인공지능 팀 프로젝트 리포트

과제 제목: CNN Model을 이용해 AI를 포함한 21개의 화가 Class 그림 분류

학번: B811134
이름: 이승환

1. 과제 개요

1.1. 주제

CNN Model을 이용해 AI를 포함한 21명의 화가 그림 분류

1.2. 주제 선정 이유

CNN은 image 인식에서 최고의 model이라고 알려져 있다. CNN을 공부하던 도중 CNN이 명화의 세밀한 특징까지도 학습해서 분류해 낼 수 있는지를 알아보고자 했다. 또한 challenge적인 요소로 AI의 그림도 분류 할 수 있는지도 알아보고자 하였다. AI의 그림은 학습을 기반으로 만들어지기 때문에 그림을 그릴 때 학습한 그림을 일부 붙여넣고 부자연스러운 경계 처리를 하는 등의 특징이 있다. 이러한 부자연스러운 특징도 알아낼 수 있는지를 알아보고자 위와 같은 주제를 선정하게 되었다.

2. 구현 환경

2.1. Data 전처리

MacOS local 환경에서 Jupyter로 실행

사용한 라이브러리 버전

matplotlib	3.5.2
numpy	1.21.5
Pillow	9.2.0
tensorflow	2.10.0
keras	2.10.0
Keras-Preprocessing	1.1.2

2.2. Model train, evaluation 등

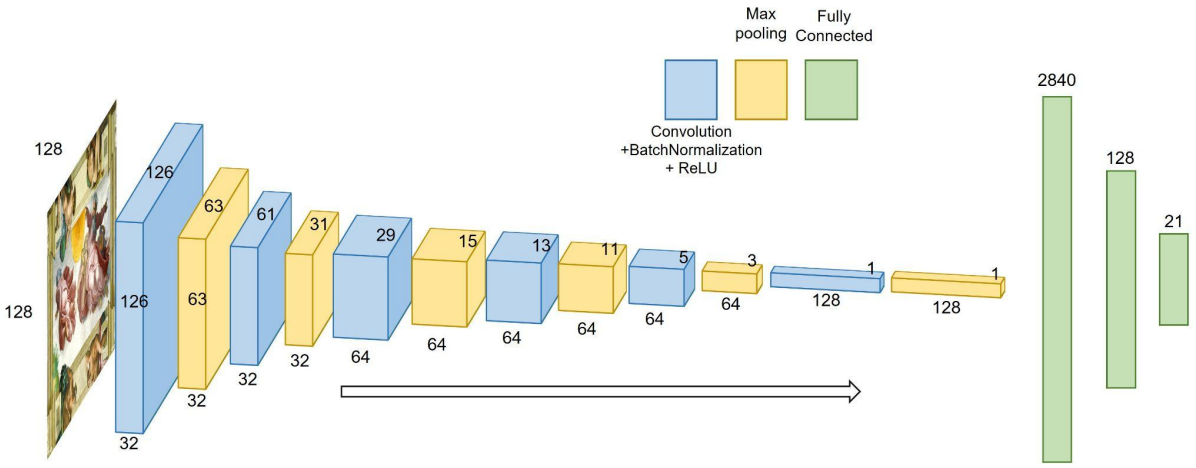
Google Colab 사용

사용한 라이브러리 버전

matplotlib	3.2.2
pandas	1.3.5
numpy	1.21.6
cv2	1.0
python	3.7.15
glob	0.7
sklearn-pandas	1.8.0
keras	2.9.0

3. 알고리즘에 대한 설명

3.1. CNN Model Architecture



3.1.1. Model 구성 layer

- Convolution 2D layer: 이미지를 classification 하는 데에 필요한 feature 정보를 추출한다. 이 layer에 있는 filter들을 통해 feature를 추출할 수 있다. 우리가 설계한 model에서 filter의 크기는 (3, 3)이고, filter로 정보를 추출하기 때문에 convolution layer를 거칠 때마다 output의 크기는 줄어들고 channel 수는 늘어난다.
- BatchNormalization: 학습하는 과정을 전체적으로 안정화하여 학습 속도를 가속하고 Local optimum 문제에 빠지는 가능성을 줄일 수 있는 방법이다. Gradient vanishing, Gradient Exploding이 일어나는 문제를 방지할 수 있다.
- Activation function: node에 입력된 값들을 비선형 함수에 통과시킨 후 다음 layer로 전달하는데, 이 때 사용하는 함수가 활성화함수(activation function)이다. 학습 과정에서는 ReLU 사용, 마지막 Dense layer에서는 Sigmoid 함수를 사용한다.
    - ReLU: 가장 많이 사용되는 활성화 함수 중 하나로, Sigmoid, Tanh의 Gradient vanishing 문제를 해결하기 위한 함수이다. 0보다 크면 기울기가 1인 직선, 0보다 작으면 함수 값이 0이 된다. Sigmoid, Tanh 함수보다 학습이 빠르고 연산 비용이 적다.
    - Sigmoid: 범위가 0에서 1 사이의 값으로 출력하는 S자형 함수이다. return 값이 확률값이기 때문에 결과를 확률로 해석할 때 유용하다. 우리 프로젝트는 multi-classification 문제이기 때문에 마지막 layer에서 Sigmoid 함수를 적용하였다.
- MaxPooling2D layer: convolution layer의 출력 이미지에서 주요 값만 뽑아 크기가 작은 출력 영상을 만든다.
- Dense layer: dense layer 의 각 뉴런이 이전 계층의 모든 뉴런으로부터 입력을 받게 됩니다. 추출된 정보를 하나의 layer로 모으고 우리가 원하는 차원으로 축소시켜서 표현하기 위한 layer이다.

- Flatten layer: convolution layer 나 maxpooling layer를 반복적으로 거치면 주요 특징만 추출되고 추출된 주요 특징을 전 결합층에 전달되어 학습된다. 전 결합층에 전달하기 위해 1차원 자료로 바꾸어 줄 때 사용하는 layer이다.
- Dropout 기법: Overfitting 을 방지하기 위해 연결된 신경망에서 0에서 1 사이의 확률로 뉴런을 Drop 하는 기법이다. 본 프로젝트에서 원본 이미지의 양이 적기 때문에 Data augmentation을 진행한 dataset으로 학습을 하였는데 그럼에도 불구하고 Train dataset이 많지 않다. Overfitting이 우려되는 사항이라 Dropout기법을 적용하여 방지하고자 하였다. 이 모델에서는 0.3의 확률로 랜덤하게 제거 여부가 결정된다.

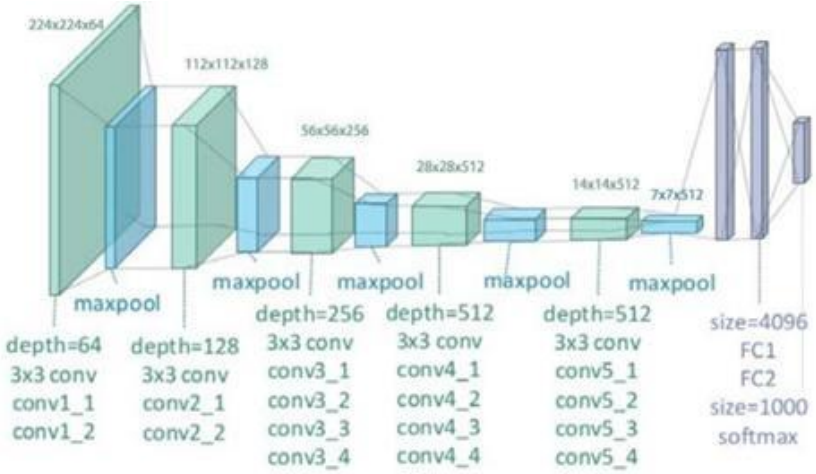
3.1.2. 구현 Model 설명

- layer는 총 6층으로 다음 과 같은 순서로 하나의 층을 구성하고 있다.
  Convolution 2D → BatchNormalization→ ReLU→ Maxpooling 2D
  이후 마지막 6번째 layer에서는 다음과 같은 순서로 구성된다.
  Convolution 2D → BatchNormalization→ ReLU→ Maxpooling 2D → Dropout
- 각 layer의 input, output의 size는 위 그림에 나와있다.
- layer 수 : 30개
- Total params: 66,996개
- Trainable params: 66,612개
- Non-trainable params: 384개

3.2. Transfer Learning Model

3.2.1. 사용 사전모델 설명

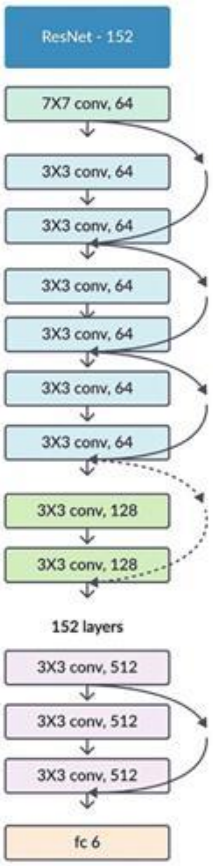
a. VGG19



(출처
:https://www.researchgate.net/profile/Clifford-Yang/publication/325137356/figure/fig2/AS:670371271413777@1536840374533/Illustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means.jpg)

- 22개 layer로 구성된다.
- 깊이의 영향만을 최대한 확인하고자 convolution layer의 filter size는 작은 3 x 3으로 고정하였다.
- 두 개 혹은 세 개의 convolution 필터와 maxpooling을 블록 단위로 연속해서 쌓는 방식 으로 진행하였다.

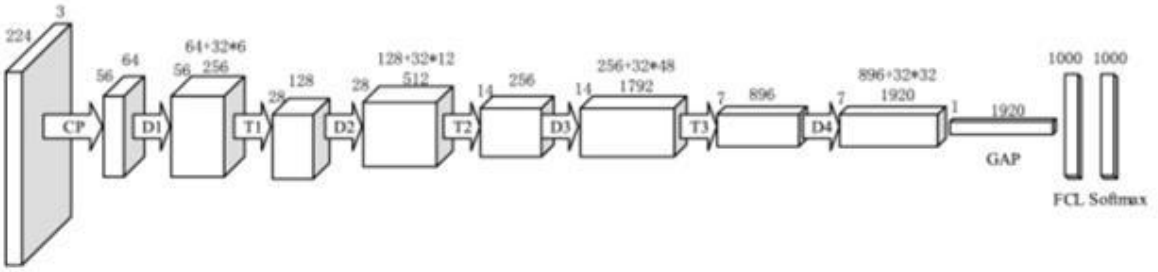
b. ResNet152v2



(출처 : https://www.researchgate.net/figure/ResNet-152-neural-network-architecture-Slika-1-Arhitektura-ResNet-152-neuronske-mreze_fig1_343615852)

- 564개 layer로 구성된다.
- bottleneck 구조 사용 : 1*1 filter를 이용하여 차원을 축소시켜 연산량을 줄여준다. 이는 더 깊은 계층의 모델을 쌓는데 핵심적인 역할을 한다.

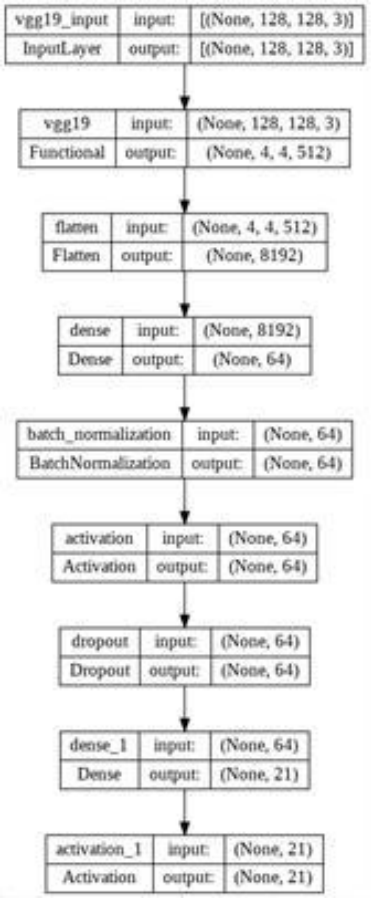
c. DenseNet201



(출처 : https://wikidocs.net/163753)

- 707개 layer로 구성
- 각 layer 들이 다른 모든 layer 들과 연결된다.
- ResNet은 다음 레이어와 skip connection시 단지 값을 더하지만 DenseNet은 concate- nation을 통해 연결한다.

3.2.2. Transfer Learning Model(최종)



- conv_base 아래 모델은 sequential로 직접 구하였으며 pre-train model로 ResNet152v2, VGG19, DenseNet201 등을 이용하여 다양하게 실험해보았으나 최종적으로 VGG19를 선택하였다. transfer learning 모델 또한 activation(ReLU, softmax), BatchNormalization layer, MaxPooling layer, Dense layer를 사용하였고 추가적으로 Dropout layer를 사용하였다.
- 레이어 수 : 29개
- Total params: 20,550,357
- Trainable params: 9,965,077
- Non-trainable params: 10,585,280

4. 데이터에 대한 설명

4.1. Input Feature

- 데이터 출처: 데이콘 화가분류(https://dacon.io/competitions/official/236006/data)
- data type: int8 형식으로 각 원소마다 0부터 255까지의 픽셀값이 들어있는 넘파이 배열
- feature의 차수 : (65248, 128, 128, 3)
- 값의 의미 : 파일 낱개로는 한 장의 명화 이미지 파일이며 해당 이미지 파일은 픽셀들의 값으로 구성되어 있다.
- 본 프로젝트에서 사용한 데이터셋은 원본 이미지 데이터 5911개에 AI 데이터 200장 추가한 것이며, 원본 데이터셋의 클래스별 데이터 수의 불균형을 어느 정도 해소하기 위해 클래스 별 데이터 평균 개수인 200개를 기준으로 200개보다 개수가 적은 클래스의 데이터를 무작위로 복제하여 200개로 맞추는 오버샘플링 작업을 진행하였다.
- 데이터 전처리 방식은 두 가지로 진행하였고, 첫 번째로는 keras 라이브러리를 이용해 증식하였다. 하지만 이는 테두리에 노이즈가 생겨 학습에 악영향을 끼쳤다. 두 번째로는 pillow 라이브러리를 이용해 직접 증식하여 노이즈를 제거하였다. 초기에는 첫 번째 방식으로 진행하였으나 최종적으로는 두 번째 방식으로 진행하였다.
- 초기 실험은 class 10개부터 진행하였으며 점차 클래스의 개수를 늘려 화가 클래스 20개에 AI가 그린 그림까지 총 21개의 클래스로 진행하였다.

4.2. Target Output

output은 input image에 대해 최종적으로 분류하고자 했던 화가 class 21개에 대한 확률값을 출력한다.

5. 소스코드에 대한 설명

5.1. 메인 code 파일

```
f = open('/content/drive/MyDrive/2022-2_기계학습팀플/train_artists_class20_AI.csv', 'r', encoding='utf-8')
rdr = csv.reader(f)
all = []
for line in rdr:
    all.append(line)
f.close()
all = all[1:]
print(all) # 수정된 파일을 대상으로 모든 정보 불러옴

all = np.array(all) # 행렬연산을 위해 numpy array로 변경
filepath = all[:,1] # 이미지 파일 불러오기 위해 파일의 경로만 저장
filepath = list(map(lambda x:x[1:], filepath))
artists = all[:,2] # 수정된 데이터에 대한 화가 이름
artists_class = all[:,3] # 수정된 데이터에 대한 화가 클래스

X_dataset = np.load(
    '/content/drive/MyDrive/2022-2_기계학습팀플/임시/oversampling/class21/ImageDataset_resize_class20_AI.npy')
print(X_dataset.shape)

tmp = []
for artist in artists_class:
  for _ in range(16):
    tmp.append(artist)

artists_class = np.array(tmp)
print(artists_class)
print(len(artists_class))
```

- 실행에 필요한 여러 가지 Package를 Import한다.
- 필요한 데이터셋은 Google Drive에 준비해 두었기 때문에 mount를 진행한다.
- Colab에서 진행하지 않는다면 google.colab package import와 하단의 mount는 생략한다.

```
import cv2, os, time, csv, os.path
import gc
import numpy as np
import pandas as pd
import tensorflow as tf

from google.colab import drive
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
from glob import glob
from PIL import Image

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

from keras import models, layers
from keras.models import Sequential
from keras import optimizers
from keras import regularizers
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, Activation, BatchNormalization
from keras.layers.convolutional import MaxPooling2D


drive.mount('/content/drive')
```

```
# 분할 시 증식한 16장 단위로 나누어야 하기 때문에 reshape 진행
X_dataset_edit = np.reshape(X_dataset, (-1, 16, 128, 128, 3))
# original_augmentation = (64, 128, 128, 3) -> reshape(-1, 16, 128, 128, 3) => (4, 16, 128, 128, 3)
Y_dataset_edit = np.reshape(artists_class, (-1, 16))
print(X_dataset_edit.shape)
print(Y_dataset_edit.shape)

from sklearn.model_selection import train_test_split

# Train, Test, Validation 분할

X_train, X_test, Y_train, Y_test= train_test_split(
    X_dataset_edit, Y_dataset_edit, test_size=0.1, shuffle=True, stratify=Y_dataset_edit, random_state=34)

X_train = np.reshape(X_train, (-1,128,128,3))
X_test = np.reshape(X_test, (-1,128,128,3))
Y_train = np.reshape(Y_train, (-1))
Y_test = np.reshape(Y_test, (-1))

gc.collect()

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

val_index = 8160

X_val = X_train[-val_index:]
X_train = X_train[:-val_index]
print(X_train.shape) # (65248, 128, 128, 3)
print(X_val.shape) # (8160, 128, 128, 3)
print(X_test.shape) # (8160, 128, 128, 3)

Y_val = Y_train[-val_index:]
Y_train = Y_train[:-val_index]
print(Y_train.shape) # (65248, 128, 128, 3)
print(Y_val.shape) # (8160, 128, 128, 3)
print(Y_test.shape) # (8160, 128, 128, 3)
```

- 해당 코드는 증식, 오버샘플링 등 전처리가 완료된 데이터를 분할하는 코드이므로 별도로 제출한 완성본 데이터셋 사용시 실행하지 않아도 된다.
- 증식 과정에서 기반으로 사용한 파일과 label에 대한 csv 파일을 불러온 뒤 적절하게 파싱하는 과정을 진행한다.
- 증식된 데이터셋은 16배가 되었으나 불러온 label은 그렇지 않으므로 똑같이 16배로 증식한다.
- 분할은 train_test_split을 통해 진행하였고, 동일한 크기로 validation도 분리해준다.

```
# path: dataset file이 있는 폴더 경로
path = '/content/drive/MyDrive/2022-2_기계학습팀플/임시/oversampling/class21'
X_train = np.load(path+'/X_train.npy')
X_test = np.load(path+'/X_test.npy')
X_val = np.load(path+'/X_val.npy')
Y_train = np.load(path+'/Y_train.npy')
Y_test = np.load(path+'/Y_test.npy')
Y_val = np.load(path+'/Y_val.npy')
```

- 위의 과정을 마친 최종 데이터셋이다. 별도로 다운로드하여 진행가능하다.
- 해당 코드를 통해 사용할 데이터셋을 불러온다. 데이터와 label값 각각 Train, Test, Validation으로 분리하여 총 6개의 파일을 불러온다.

```
y_train = np.asarray(Y_train).astype('float32').reshape((-1,1))
y_test = np.asarray(Y_test).astype('float32').reshape((-1,1))
y_val = np.asarray(Y_val).astype('float32').reshape((-1,1))

enc=OneHotEncoder()

enc.fit(y_train)
y_train_onehot=enc.transform(y_train).toarray()

enc.fit(y_test)
y_test_onehot=enc.transform(y_test).toarray()

enc.fit(y_val)
y_val_onehot=enc.transform(y_val).toarray()
```

- 불러온 label 값은 현재 [14, 20, 1, 6, …] 과 같은 해당 클래스의 숫자이므로 one-hot encoding을 진행한다.
- 그 결과 y_train은 (65248,)에서 (65248, 21)로, y_test와 y_val은 (8160,)에서 (8160, 21)으로 shape이 변경된다.

```
n_filter = 32 #convolution filter #
conv_r = 3  #convolution kernel row
conv_c = 3  #convolution kernel column
activation_function = 'relu'  #activation function define
stride = 2
epoch = 10
batch_size = 128

model=Sequential()

model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c),  padding='same', input_shape=(128, 128, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))
```

```
model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))


model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(n_filter, kernel_size=(conv_r, conv_c), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(rate=0.3))


model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Activation('relu'))

model.add(Dense(20, kernel_regularizer=regularizers.l2(0.001)))
model.add(Activation('sigmoid'))



model.compile(optimizer=optimizers.Adam(lr=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()


start = time.time()
history = model.fit(X_train, y_train_onehot, batch_size=128, epochs=10, validation_data = (X_val, y_val_onehot))
print('fit time=', time.time()-start)

score = model.evaluate(X_test, y_test_onehot, verbose=0, batch_size = batch_size)
print(score)
```

- Basic CNN Model을 구성하는 코드이다.
- 하단의 fit 함수를 통해 학습을 진행하고, evaluate를 통해 테스트 데이터셋에 대해 평가한다.

```
resnet152v2 = tf.keras.applications.ResNet152V2(
              include_top=False,
              weights="imagenet",
              input_tensor=None,
              input_shape=(128,128,3),
              pooling=None,
              classes=21,
              classifier_activation="softmax",
          )

VGG19 = tf.keras.applications.VGG19(
        include_top=False,
        weights="imagenet",
        input_tensor=None,
        input_shape=(128,128,3),
        pooling=None,
        classes=21,
    )

DenseNet201 = tf.keras.applications.DenseNet201(
              include_top=False,
              weights="imagenet",
              input_tensor=None,
              input_shape=(128,128,3),
              pooling=None,
              classes=21,
          )
```

- 사용한 사전 모델 세 가지 load하는 코드이다.
- weights : 해당 모델의 가중치를 어떤 Dataset으로 학습한 가중치를 불러올지 지정한다.
- include_top : classifier 부분 포함 여부 → 프로젝트 모델에 맞춰야 하므로 제외한다.
- input_shape : 입력 shape이다.
- input_tensor : Input으로 특정 tensor 사용하고 싶다면 설정한다.
- pooling : include_top이 false일 때 None, avg, max 중 선택한다.
- classes : include_top이 true일 때 output class 수, 해당 프로젝트에선 의미 없음
- classifier_activation : 위와 동일하다.

```
from keras import models, layers
from keras.callbacks import EarlyStopping

# conv_base = resnet152v2
conv_base = VGG19
# conv_base = DenseNet201
i = 0
freeze = 5

# Freeze all the layers
for layer in conv_base.layers[:]:
    layer.trainable = False
    i += 1
    print(i) # layer 개수 세기

i = 0
# UnFreeze all the layers
for layer in conv_base.layers[-freeze:]:
    print(i)
    layer.trainable = True
    i += 1

# Check the trainable status of the layers
for layer in conv_base.layers:
    print(layer, layer.trainable)
```

- pre-trained model을 사용하고 fine-tuning을 진행한다.
- 우선 conv_base 부분을 다운로드 받은 사전모델로 지정한 뒤, 모든 레이어를 학습불가능하게 지정한다.
- 그 다음 끝에서 k개의 레이어만 학습 가능하게 변경한다.
    - 이 때 보통 끝에서 10% 안팎 정도의 레이어를 지정하였다.
    - 일반적인 묶음 단위로 학습 허용 범위를 설정하였다. (convolution 이후 activation 레이어부터 학습가능하게 설정하는 것과 같은 경우를 피함)
- 마지막으로 현재 모든 레이어들의 학습 가능 여부를 출력하여 확인한다.

```
import time

batch_size = 64
epoch = 20

model2 = models.Sequential()
model2.add(conv_base)

model2.add(Flatten())
model2.add(Dense(64, kernel_regularizer=regularizers.l2(0.001)))
model2.add(BatchNormalization())
model2.add(Activation('relu'))
model2.add(Dropout(0.4))

model2.add(Dense(21, kernel_regularizer=regularizers.l2(0.0001)))
model2.add(Activation('softmax'))

model2.compile(optimizer=optimizers.Adam(lr=0.00001), loss='categorical_crossentropy', metrics=['accuracy'])
model2.summary()

gc.collect()
```

- convolution base 부분은 사전모델을 사용하였으나 classifier 부분은 프로젝트에 맞게 수정하여야 하기 때문에 직접 구성하였다.
- 여러 가지 레이어들을 쌓고, class 수 21개에 맞추어 최종 output을 설정하였다.

```
start = time.time() # 시간 측정
early_stopping = EarlyStopping(monitor='val_loss', patience=3, mode='min')
history = model2.fit(X_train, y_train_onehot, batch_size=batch_size, epochs=epoch,#
                     validation_data = (X_val, y_val_onehot), callbacks = [early_stopping])
print('fit time=', (time.time()-start)/60, ':', (time.time()-start)%60) # 소요 시간 출력
model2.evaluate(X_test, y_test_onehot, batch_size=batch_size)
```

- 전체 학습시간을 측정하기 위하여 time package 사용하였다.
- 더 이상 학습이 되지 않고 val_loss가 줄어들지 않는다면 early stopping을 이용하여 중단하였다.
- 학습 종료 후 test dataset에 대해 평가 진행하였다.

```
#accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

#loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

- Train과 Validation의 loss와 accuracy 변화 추이를 matplotlib를 이용하여 그래프로 시각화 하였다.

5.2. 전처리 code 파일

```
'''
    함수 이름: default_resize
    기능: 이미지를 입력받아서 원하는 사이즈로 비율 상관없이 크기 조정(단 정사각형으로만)
    매개변수: img(이미지),size(원하는 가로세로 길이)
    반환: 크기 조정된 이미지
'''


def default_resize(img, size):

    return img.resize((size,size))

'''
    함수 이름: crop_resize
    기능: 이미지를 입력받아서 원하는 사이즈로 비율을 유지하며 크롭후 크기 조정(단 정사각형으로만)
    매개변수: img(이미지),size(원하는 가로세로 길이)
    반환: 크기 조정된 이미지
'''
def crop_resize(img, size):
    width = img.width
    height = img.height

    if width == height:
        return img.resize((size,size))
    elif width > height:
        center_X = width / 2
        center_Y = height / 2
        area = (center_X - math.floor(center_Y), 0, center_X + center_Y, height)
        cropped = img.crop(area)

        return cropped.resize((size,size))
    else:
        center_X = width / 2
        center_Y = height / 2
        area = (0, center_Y - math.floor(center_X), width, center_Y + center_X)
        cropped = img.crop(area)

        return cropped.resize((size,size))

'''
    함수 이름: padding_resize
    기능: 이미지를 입력받아서 원하는 사이즈로 모자란 부분은 검은색 패딩을 입혀 크기 조정(단 정사각형으로만)
    매개변수: img(이미지),size(원하는 가로세로 길이)
    반환: 크기 조정된 이미지
'''

def padding_resize(img, size):
    width = img.width
    height = img.height
    center_X = img.width / 2
    center_Y = img.height / 2

    if width == height:
        return img.resize((size,size))
    elif width > height:
            padding_img = Image.new(mode = 'RGB', size = (width, width), color = (0,0,0))
            padding_img.paste(img, (0, round((width-height)/2)))

            return padding_img.resize((size,size))
    else:
        padding_img = Image.new(mode = 'RGB', size = (height, height), color = (0,0,0))
        padding_img.paste(img, (round((height-width)/2),0))

        return padding_img.resize((size,size))

'''
    함수 이름: resize_custom
    기능: 이미지를 입력받아서 원하는 방식으로 원하는 사이즈로 크기 조정(단 정사각형으로만)
    매개변수: img(이미지),size(원하는 가로세로 길이), style(크기조정 방식)
    반환: 크기 조정된 이미지
'''
def resize_custom(image, size, style):
    if style == 'resize':
    # 비율 신경안쓰고 resize
        return default_resize(image, size)

    elif style == 'crop':
    # 그냥 crop
        return crop_resize(image, size)

    elif style == 'padding':
    # padding 추가해서 정사각형으로 만들기

        return padding_resize(image,size)

    else:
        print("Type correct style!")
```

- 이미지 리사이징 관련 함수들이다.
- resize, crop, padding 세가지 방식의 함수들과 한번에 호출하는 resize_custom 함수로 이루어져 있다.

```
'''
    함수 이름: crop_random
    기능: 이미지를 입력받아서 원하는 방식으로 원하는 사이즈만큼 랜덤한 위치에서 뽑아냄
    매개변수: img(이미지),size(원하는 가로세로 길이)
    반환: 랜덤 크롭된 이미지
'''
def crop_random(img, size):

    center_X = img.width
    center_Y = img.height

    ran_X = random.randrange(0, center_X - size)
    ran_Y = random.randrange(0, center_Y - size)

    cropped = img.crop((ran_X, ran_Y, ran_X + size, ran_Y + size))

    return cropped

'''
    함수 이름: crop_random_seed
    기능: 이미지를 입력받아서 원하는 방식으로 원하는 사이즈만큼 랜덤한 위치에서 뽑아냄
    단, 시드값을 입력해서 항상 같은 이미지가 반환되게 고정함
    매개변수: img(이미지),size(원하는 가로세로 길이), seed(시드값)
    반환: 랜덤 크롭된 이미지
'''
def crop_random_seed(img, size, seed):

    random.seed(seed)
    center_X = img.width
    center_Y = img.height

    ran_X = random.randrange(0, center_X - size)
    ran_Y = random.randrange(0, center_Y - size)

    cropped = img.crop((ran_X, ran_Y, ran_X + size, ran_Y + size))

    return cropped
'''
    함수 이름: crop_center
    기능: 이미지를 입력받아서 원하는 방식으로 원하는 사이즈만큼 중앙에서 뽑아냄
    매개변수: img(이미지),size(원하는 가로세로 길이)
    반환: 중앙 크롭된 이미지
'''
def crop_center(img, size):

    center_X = img.width // 2
    center_Y = img.height // 2
    area = (center_X - (size // 2), center_Y - (size // 2), center_X + (size // 2), center_Y + (size // 2))

    return img.crop(area)
```

- 이미지 크롭 관련 함수들이다.
- 랜덤크롭과 중앙 크롭함수로 이루어져있다.
- 참고 출처: https://stackoverflow.com/questions/57221754/how-to-autocrop-randomly-using-pil

```
import csv
import numpy as np
from PIL import Image
import os


#csv 읽어온 후 변수에 저장
f = open('./train_artists_class20_AI.csv', 'r', encoding='utf-8')
rdr = csv.reader(f)
all = []
for line in rdr:
    all.append(line)
f.close()

#필요없는 첫번째 행 잘라낸 후 넘파이 배열로 저장
all = all[1:]
all = np.array(all)
print(all[0])

#artist 리스트 저장
artist = all[:,2]
print(artist)

#artist_class 리스트 저장
artists_class = all[:,3] # 수집된 데이터에 대한 최기 클래스
print(artists_class)

#파일 경로 저장
filepath = all[:,1]
filepath = list(map(lambda x:x[1:], filepath))

#리스트에 경로중에 사진이름 네자리 숫자만 저장
#예를들어 ./train/0000.jpg 라면 0000만 저장
onlyNum = []
for i in range(len(filepath)):
    onlyNum.append(filepath[i][7:11])
```

- csv파일을 읽어오고 내용들을 리스트에 저장한다.

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os


path = "./train/" #이미지 파일이 있는 경로
file_lst = os.listdir(path) #경로에 있는 파일 리스트를 모두 저장
file_lst = [dir for dir in file_lst if not dir.startswith ('.')]
#이 코드는 맥에서만 필요한 코드니 신경쓰지 않으셔도 됩니다

file_lst.sort() #리스트를 오름차순으로 정렬


#증식기에 설정해줄 값
datagen = ImageDataGenerator(

                rescale=1./255,
                rotation_range=15,
                width_shift_range=0.1,
                height_shift_range=0.1,
                shear_range=0.5,
                zoom_range=[0.7,1.3],
                horizontal_flip=True,
                vertical_flip=True,
                fill_mode='nearest'

                        )


#경로 안의 모든파일들에 대해 연산
for j in file_lst:
    img = image.load_img(path + j)
    img_arr = image.img_to_array(img)
    img_arr = img_arr.reshape((1,)+img_arr.shape)
    #증식기에 집어넣기 위해 이미지를 불러오고 numpy로 바꾸후 모양 변경

    i=0

    prefix = j[0:4] #0000.jpg 가 있다면 앞의 0000만 추출해서 prefix로 사용

    ##save_to_dir은 증식한 파일을 저장할 경로
    for batch in datagen.flow(img_arr,batch_size=1, save_to_dir='./new_aug/', save_prefix=prefix+'_',save_format='jpg'):
        i+=1
        if i > 15: #증식갯수 16개
            print(j+"finished")
            break
```

- 리스트에 들어있는 내용에 해당하는 파일만 keras로 증식시킨다.
- Keras 증식기 오류로 인해 파일이 원하는 갯수만큼 생성되지 않는 경우가 있음 그럴때는 해당하는 파일 번호만 다시 증식한다.
- 참고 출처: https://diane-space.tistory.com/174

```
##증식된 모든 파일에 대하여
path = "./new_aug/"
file_lst = os.listdir(path)
file_lst = [dir for dir in file_lst if not dir.startswith ('.')]


file_lst.sort()


X_dataset = [] # 모든 사진들을 넘파이 배열로 바꾼 후 저장할 리스트

for j in file_lst:
    if j[:4] not in onlyNum:
        continue
    img = image.load_img('./new_aug/'+j)
    img = resize_custom(img, 128, 'resize')
    #img = resize_custom(img, 128, 'crop')
    #img = resize_custom(img, 128, 'padding') # 이 지리에 resize method 중 하나사용
    X_dataset.append(np.array(img))
    print(j)



print("finished")
```

```
# 넘파이 배열 형태 바꾸기
print(type(X_dataset[0]))
print(len(X_dataset))
print(X_dataset[0].shape)


for i in range(len(X_dataset)):
      if X_dataset[i].ndim == 2:
        tmp = X_dataset[i].reshape(128,128,1)
        tmp1 = np.concatenate((tmp, tmp), axis=2)
        tmp2 = np.concatenate((tmp1, tmp), axis=2)
        X_dataset[i] = tmp2
npData = np.array(X_dataset)
print(npData.shape)
```

```
#.npy 파일로 저장
plt.imshow(npData[903])
plt.show()

np.save('./ImageDataset_padding_class21_AI.npy', npData)
```

- 증식된 사진을 넘파이 배열로 바꾸고 형태를 변경한 후 저장한다.

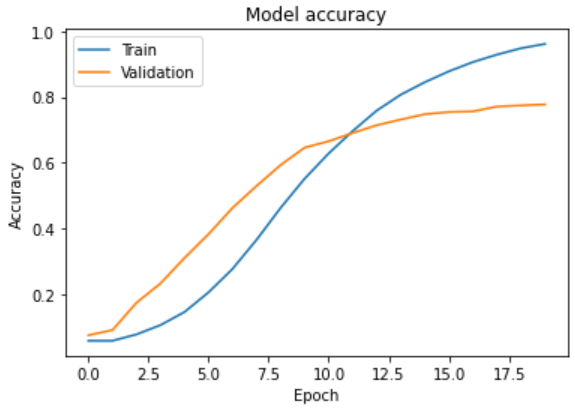
```
X_dataset = []


for path in filepath:
    #1 원본
    img = Image.open('.'+path)
    custom = resize_custom(img, 128, 'resize')
    X_dataset.append(np.array(custom))

    ### 이미지 너비랑 높이 저장변수 선언
    img_width = img.width
    img_height = img.height
    if img_width>img_height:
        key_length = img_height
    else:
        key_length = img_width

    #2 좌우반전
    leftright = img.transpose(Image.FLIP_LEFT_RIGHT)
    custom = resize_custom(leftright, 128, 'resize')
    X_dataset.append(np.array(custom))

    #3 상하반전
    updown = img.transpose(Image.FLIP_TOP_BOTTOM)
    custom = resize_custom(updown, 128, 'resize')
    X_dataset.append(np.array(custom))

    #4 상하좌우반전
    leftright = leftright.transpose(Image.FLIP_TOP_BOTTOM)
    custom = resize_custom(leftright, 128, 'resize')
    X_dataset.append(np.array(custom))

    #5 시계방향 90도
    rotate = img.rotate(90, expand = True)
    custom = resize_custom(rotate, 128, 'resize')
    X_dataset.append(np.array(custom))

    #6 90도 후 좌우반전
    leftright = rotate.transpose(Image.FLIP_LEFT_RIGHT)
    custom = resize_custom(leftright, 128, 'resize')
    X_dataset.append(np.array(custom))

    #7 90도 후 상하반전
    updown = rotate.transpose(Image.FLIP_TOP_BOTTOM)
    custom = resize_custom(updown, 128, 'resize')
    X_dataset.append(np.array(custom))

    #8 90도 후 상하좌우반전
    leftright = leftright.transpose(Image.FLIP_TOP_BOTTOM)
    custom = resize_custom(leftright, 128, 'resize')
    X_dataset.append(np.array(custom))
```

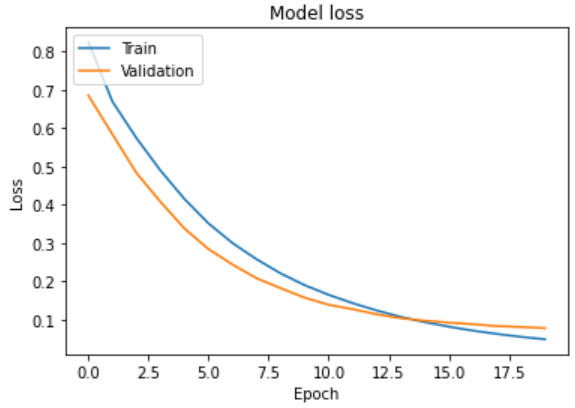
```
    #9 랜덤크롭1
    random.seed(100)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(img, length, 1000)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #10 랜덤크롭2

    random.seed(1)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(img, length, 10)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #11 랜덤크롭3
    random.seed(2)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(img, length, 20)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #12 랜덤크롭4
    random.seed(3)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(img, length, 30)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #13 90도 후 랜덤크롭1
    random.seed(200)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(rotate, length, 2000)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #14 90도 후 랜덤크롭2
    random.seed(4)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(rotate, length, 40)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #15 90도 후 랜덤크롭3
    random.seed(5)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(rotate, length, 50)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))

    #16 90도 후 랜덤크롭4
    random.seed(6)
    length = random.randrange(200, key_length)
    crop = crop_random_seed(rotate, length, 60)
    custom = resize_custom(crop, 128, 'resize')
    X_dataset.append(np.array(custom))


    print(path[7:])
    del img, leftright, updown, crop, rotate, custom, img_width, img_height, key_length, length
    gc.collect()
    ##메모리 가비지컬렉션코드

print("finished")
```

- Pil로 노이즈가 없는 방식으로 증식한다.
- 최종적으로 결정된 방식이다.
- 이미지를 상하좌우 반전 4장, 90도 회전 후 상하좌우 반전 4장, 랜덤크롭 8장으로 구성된다.
- 랜덤크롭 데이터의 일관성을 위해 시드값을 적용한다.
- 이후는 keras 방식과 동일하게 넘파이 파일로 저장한다.

6. 학습 과정에 대한 설명

```
Epoch 1/20
1020/1020 [==============================] - 161s 148ms/step - loss: 0.8243 - accuracy: 0.0582 - val_loss: 0.6855 - val_accuracy: 0.0750
Epoch 2/20
1020/1020 [==============================] - 154s 151ms/step - loss: 0.6697 - accuracy: 0.0580 - val_loss: 0.5840 - val_accuracy: 0.0906
Epoch 3/20
1020/1020 [==============================] - 161s 158ms/step - loss: 0.5743 - accuracy: 0.0775 - val_loss: 0.4832 - val_accuracy: 0.1740
Epoch 4/20
1020/1020 [==============================] - 155s 152ms/step - loss: 0.4895 - accuracy: 0.1058 - val_loss: 0.4078 - val_accuracy: 0.2327
Epoch 5/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.4151 - accuracy: 0.1457 - val_loss: 0.3380 - val_accuracy: 0.3104
Epoch 6/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.3514 - accuracy: 0.2056 - val_loss: 0.2844 - val_accuracy: 0.3830
Epoch 7/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.3000 - accuracy: 0.2772 - val_loss: 0.2439 - val_accuracy: 0.4629
Epoch 8/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.2578 - accuracy: 0.3658 - val_loss: 0.2082 - val_accuracy: 0.5298
Epoch 9/20
1020/1020 [==============================] - 162s 158ms/step - loss: 0.2209 - accuracy: 0.4628 - val_loss: 0.1822 - val_accuracy: 0.5933
Epoch 10/20
1020/1020 [==============================] - 155s 152ms/step - loss: 0.1902 - accuracy: 0.5518 - val_loss: 0.1576 - val_accuracy: 0.6463
Epoch 11/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.1649 - accuracy: 0.6285 - val_loss: 0.1391 - val_accuracy: 0.6656
Epoch 12/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.1432 - accuracy: 0.6968 - val_loss: 0.1273 - val_accuracy: 0.6908
Epoch 13/20
1020/1020 [==============================] - 157s 154ms/step - loss: 0.1241 - accuracy: 0.7591 - val_loss: 0.1140 - val_accuracy: 0.7145
Epoch 14/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.1074 - accuracy: 0.8074 - val_loss: 0.1039 - val_accuracy: 0.7316
Epoch 15/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.0936 - accuracy: 0.8453 - val_loss: 0.0974 - val_accuracy: 0.7479
Epoch 16/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.0818 - accuracy: 0.8784 - val_loss: 0.0924 - val_accuracy: 0.7547
Epoch 17/20
1020/1020 [==============================] - 157s 154ms/step - loss: 0.0714 - accuracy: 0.9066 - val_loss: 0.0882 - val_accuracy: 0.7564
Epoch 18/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.0629 - accuracy: 0.9293 - val_loss: 0.0833 - val_accuracy: 0.7712
Epoch 19/20
1020/1020 [==============================] - 156s 153ms/step - loss: 0.0552 - accuracy: 0.9487 - val_loss: 0.0810 - val_accuracy: 0.7746
Epoch 20/20
1020/1020 [==============================] - 162s 159ms/step - loss: 0.0489 - accuracy: 0.9619 - val_loss: 0.0780 - val_accuracy: 0.7777
fit time= 52.436349328358965 : 26.18096089363098
128/128 [==============================] - 15s 118ms/step - loss: 0.0794 - accuracy: 0.7806
[0.07943713665008545, 0.780637264251709]
```





- 최종 모델을 학습하며 경과를 출력해보았다.
- epoch이 18이 될 때까지 accuracy가 계속해서 증가하는 추이를 관찰할 수 있다.
- 하지만 validation set에 대한 accuracy가 77%에서 더 증가하지 못하는 점 또한 볼 수 있다. 경미하게나마 마지막 epoch까지 loss는 계속해서 감소하는 것도 확인할 수 있었다.
- epoch을 더 증가시킨다고 해도 좋은 결과로 이어지지 않을 것으로 예측하여 해당 epoch에서 중단하였다.
- 그래프를 관찰 하였을 때 오버피팅이 어느 정도 발생한 것을 관찰할 수 있었으나 초기 실험 결과에 비하면 오버피팅을 상당 부분 완화하였다.

7. 결과 및 분석

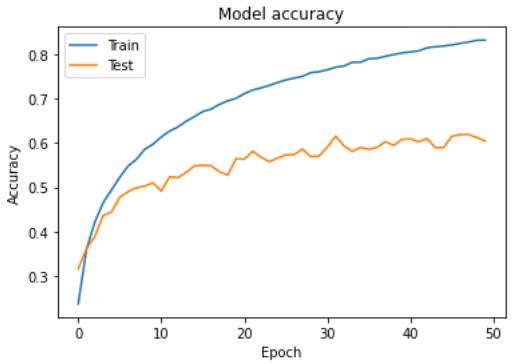
7.1. 결과

실험 결과를 정리한 excel file 링크는 다음과 같습니다.

최종실험결과

7.1.1. Basic CNN Model

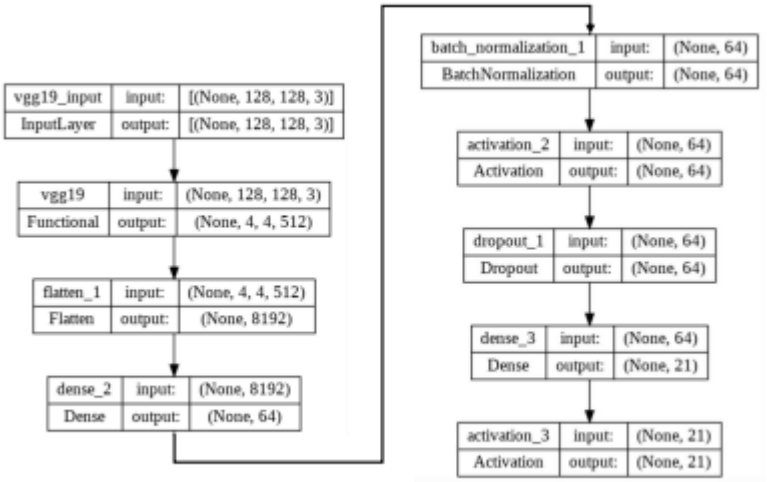
변경한 Parameter는 주로 epoch, layer 갯수, kernel size이다. 다른 parameter는 큰 변화가 없이 55~57%를 맴돌았지만, epoch이 증가할 수록 accuracy가 높아졌다. train, validation, test set 모두에 대해 증가하였기 때문에 model의 성능이 좋아졌다고 판단하였다.



epoch이 50일 때의 학습 그래프를 보면 test set에서는 점차 학습이 더디게 되고 있음을 알 수 있다. epoch이 40일 때 보다 loss값이 증가하였기 때문에 epoch이 50 일 때 실험을 중단하였다. 하지만 test accuracy가 0.6275로 class가 21개인 데 반해 높은 accuracy가 나왔다고 생각한다.

하지만 더 좋은 결과물을 내기 위해 우리의 두번째 model인 Transfer learning model을 우리의 dataset에 적용해보았다.

7.1.2. Transfer Learning Model



(실험 결과 최종적으로 선택한 모델)

- conv_base(VGG19) - flatten - Dense - BatchNormalization - ReLU - Dropout - Dense – Softmax
- 약 60회 가량의 실험을 진행한 결과 학습이 안정적으로 진행되고 정확도가 높은 모델을 최종 선정하였다.
- 위와 같은 구조로 모델을 구성하였으며 accuracy로는 약 78%의 성능을 보였다.
- 21개의 클래스를 분류하는 것인만큼 당초 예상했던 정확도보다 높은 성능을 나타냈다.

7.2. 분석

- data 수가 적음을 인지하고 이미 학습된 사전 모델을 이용하여 fine-tuning하는 방식을 택하였다.
- 사전 모델 중에서도 최신 모델, 더 깊고 넓은 모델이 좋은 성능을 낼 것으로 예상하여 ResNet152v2를 택하여 다양한 실험을 진행하였으나, 좋은 결과를 얻지 못하였다.
- 사전 모델을 변경하여 비교적 간단한 VGG19와 독특한 구조의 DenseNet201을 선정하여 여러 가지 실험을 진행해보았고, 그 중 VGG19에서 제일 좋은 실험 결과를 얻었다.
- 항상 깊은 model만이 좋은 성능을 내는 것이 아니고 특히 overfitting의 위험성이 있는 dataset에서는 더욱 영향을 받음을 예측할 수 있었다.
- 대체로 기존 Conv_base 부분을 freezing하고 learning rate를 작게하여 epoch을 늘렸을 때 성능이 제일 좋았다.
- Optimizer는 Adam, Nadam 등을 시도해 보았으나 대부분 비슷하거나 Adam에서 성능이 가장 좋았음 최종 model에도 어느정도 overfitting이 존재하는데, 이 부분까지 완벽하게 해결하지 못 한 점이 아쉬웠다.
- 추가로 한번 학습에 약 30분~1시간이 소요됨에 따라 마감 기한을 준수하기 위해 더욱 다양한 실험을 진행하지 못한 점이 아쉽다.
- 사전모델은 이번 프로젝트에서 다룬 모델 외에도 다양하고, 최신 딥러닝 기법 또한 존재하므로 다양한 모델로 실험해보고 기법들을 적용하면 더욱 발전할 수 있음을 예상한다.

8. 실행 메뉴얼

8.1. 기본 정보

코드 작성과 실험이 대부분 Google Colab에서 진행되었기 때문에 가능하다면 Colab에서 ipynb 파일을 실행하는 것을 권장합니다.

Dataset 파일 첨부링크
https://hongik-my.sharepoint.com/:f:/g/personal/jh9788_mail_hongik_ac_kr/EuCr51KVzh5CuwOdx4iqZ7kBTYLdOExwMdnp2ODauH3kaA?e=beXjf2

8.2. 실행 방법

A. 최종 데이터셋은 위의 링크에서 train, test, validation 파일 총 6개를 받아서 진행한다. 라이브러리는 colab 기본 패키지 버전으로 실행하되, colab이 아니라면 해당링크의 requirement.txt 파일을 다운 받아서 `pip install -r requirements.txt` 명령어를 실행하여 패키지를 설치한다.

B. main code ipynb 파일에 각 셀마다 번호를 매겨두었으며 이하 셀 번호로 설명한다.

C. main code에서 1번 import하는 셀과 4번 dataset 불러오기(path에 데이터셋을 다운로드 받은 경로 입력) 셀을 실행한다.

D. 5번 one-hot encoding 코드를 실행한다.

E. 이 후 Basic CNN Model에 대한 결과를 확인하고자 하면 6번 셀을 실행한다.

F. 혹은 Transfer Model에 대한 결과를 확인하고자 하면 사전학습 모델에 맞추어 7/8/9 중 알맞게 선택하여 pre-train model 다운로드 셀을 실행한다.

G. 그 다음 freezing하는 셀을 실행하는데, 주석처리 된 conv_base 부분을 다운로드 받은 사전모델에 맞게 수정한 뒤 두 번째 for문의 freezing할 layer 수를 입력한 뒤 실행한다.

H. 11번 classifier 정의, 12번 학습, 13번 그래프출력을 순서대로 실행하여 결과를 확인한다.

I. 전처리과정을 확인하기 위해서는 위의 링크에서 train.zip과 train_artists_class20_AI.csv를 다운로드 받아서 경로에 맞춰 전처리 코드파일(CNN_painting_preprocessing.ipynb)과 readme.txt 을 참고하여 실행한다.