1. dataset 준비하기

· 코랩 파일:

https://colab.research.google.com/drive/1xdzpe1Qy7nqrjKOajR6uVvYBCb_ZYs0G#scrollTo=WZLYrWEbsHb9

	A	B	C
1	file_name	emotion	fold
2	angry_000	angry	5
3	angry_001	angry	5
4	angry_002	angry	1
5	angry_003	angry	4
6	angry_004	angry	2
7	angry_005	angry	1
8	angry_006	angry	4
9	angry_007	angry	5
10	angry_008	angry	2
11	angry_009	angry	3
12	angry_010	angry	4
13	angry_011	angry	3
14	angry_012	angry	3
15	angry_013	angry	4
16	angry_014	angry	4
17	angry_015	angry	1

feelings_skfold2

준비 접근성: 사용할 수 없음

2. Model 정의하기

- 사용한 모델: Keras의 Sequential 모델

Tensorflow Hub 활용

-> https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b0/feature_vector/2

- input_shape: batch_size는 일단 지정 x, width/height는 모두 48, 색상: 3

- optimizer: Adam, learning rate = 0.0001

- loss 함수: sparse_categorical_crossentropy

-> 다중 분류 손실 함수

-> one - hot - encoding을 하지 않고 정수 형태로 넣어줌,

-> 한 샘플에 여러 클래스가 있거나 label이 soft(확률)일 경우 사용

=> Label의 경우 각각의 감정에 번호를 부여해 정수로 제공할 계획

+ ***categorical_crossentropy*** 활용하는 방법도 생각해 봄

-> one-hot encoding 문제인지 모델링 문제인지 model.fit( )에서 에러 발생(확인 필요)

* 손실함수 관련 reference: https://www.tensorflow.org/api_docs/python/tf/keras/losses

- metrics: accuracy(사실 이건 조금 헷갈려요. 그저 복붙함..^-^;)

- 모델 구조(summary)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              5919312

 dense (Dense)               (None, 7)                 8967

=================================================================
Total params: 5,928,279
Trainable params: 5,867,671
Non-trainable params: 60,608
_________________________________________________________________
```

- Albumentations: 수업 때 진행한 내용과 동일(별다른 수정 거치지 x)

```python
import albumentations as A

# Albumentation class 생성하기
class Augmentation:
    def __init__(self, size, mode='train'):
        if mode == 'train':
            self.transform = A.Compose([
                # 수평 전환
                A.HorizontalFlip(p=0.5),
                # 이동, 크기, 회전을 설정
                A.ShiftScaleRotate(
                    p=0.5,
                    shift_limit=0.05,
                    scale_limit=0.05,
                    rotate_limit=15
                ),
                # 구멍을 dropout 하게됨
                A.CoarseDropout(
                    p=0.5,
                    max_holes=8, # 최대 8개의 구멍을 dropout 시킴
                    max_height=int(0.1 * size),
                    max_width=int(0.1 * size)
                ),
                A.RandomBrightnessContrast(p=0.2) # 밝기 대비
            ])

    def __call__(self, **kwargs): # callback 함수: 클래스의 객체를 생성한 이후 객체를 호출할 때 자동으로 실행되는 함수
        if self.transform:
            augmented = self.transform(**kwargs) # **kwargs : 가변 인수(파라미터의 개수에 제한을 두지 않겠다.)
            img = augmented['image'] # 증폭된 이미지
            return img
```

- DataGenerator 만들기: 수업 때 진행한 내용과 유사 + 몇 개의 부분만 변경

(변경된 부분 위주로 표시하였습니다.)

1) <u>감정 labeling</u>

- 딕셔너리 형태를 이용해 감정(key)와 label 정수(value) 쌍으로 묶어줌

```python
csv_path = './feelings_skfold2.csv'

LABEL_INT_DICT = np.unique(pd.read_csv(csv_path)['emotion'])
pprint(LABEL_INT_DICT) # 데이터의 타입과 형태 등도 같이 보여준다.(조금 더 예쁘게 출력해준다?)
LABEL_STR_DICT = {k:v for v,k in enumerate(LABEL_INT_DICT)}
pprint(LABEL_STR_DICT)# Keras의 Sequential model 이용
```

```
array(['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise'],
      dtype=object)
{'angry': 0,
 'disgust': 1,
 'fear': 2,
 'happy': 3,
 'neutral': 4,
 'sad': 5,
 'surprise': 6}
```

LABEL_INT_DICT

LABEL_STR_DICT

2) DataGenerator

```python
class DataGenerator(keras.utils.Sequence):
    def __init__(self, batch_size, csv_path, fold, image_size, mode='train', shuffle=True):
        self.batch_size = batch_size
        self.fold = fold
        self.image_size = image_size
        self.mode = mode
        self.shuffle = shuffle

        self.df = pd.read_csv(csv_path)

        if self.mode == 'train':
            self.df = self.df[self.df['fold'] != self.fold]
        elif self.mode == 'val':
            self.df = self.df[self.df['fold'] == self.fold]

        self.transform = Augmentation(image_size, mode)

        self.on_epoch_end()
```

```python
    def on_epoch_end(self):
        if self.shuffle: # shuffle = True라면(df 앞의 인덱스를 지워주는 기능
            self.df = self.df.sample(frac=1).reset_index(drop=True)
    # len()
    def __len__(self):
            return math.ceil(len(self.df) / self.batch_size)

    def __getitem__(self, idx):
        strt = idx * self.batch_size
        fin = (idx + 1) * self.batch_size
        data = self.df.iloc[strt:fin]
        batch_x, batch_y = self.get_data(data)
        return np.array(batch_x), np.array(batch_y)
```

```python
    def get_data(self, data):
        batch_x = [] #사진
        batch_y = [] # label (감정)
                          넘파이 함수
        for _, r in data.iterrows():
            file_name = r['file_name']
            img_folder = r['emotion'] # type = np.str_
                                                                                  흑백 사진
            image = cv2.imread(f'datasets/{img_folder}/{file_name}.jpg',cv2.IMREAD_GRAYSCALE)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, (self.image_size, self.image_size))

            if self.mode == 'train':
                image = image.astype('uint8') # 정수로 형변환
                image = self.transform(image=image)

            image = image.astype('float32') # 실수로 다시 형변환
            image = image / 255. # 0~1 사이의 값을 가지도록


            emotion = str(img_folder) # 사진에 해당하는 감정 찾기(angry, fear, ... )
            emotion = LABEL_STR_DICT[emotion] # 감정(str) → label(정수)
                                                                  0~6
            batch_x.append(image)
            batch_y.append(emotion)

        return batch_x, batch_y
```

3) train_generator와 valid_generator 객체 생성

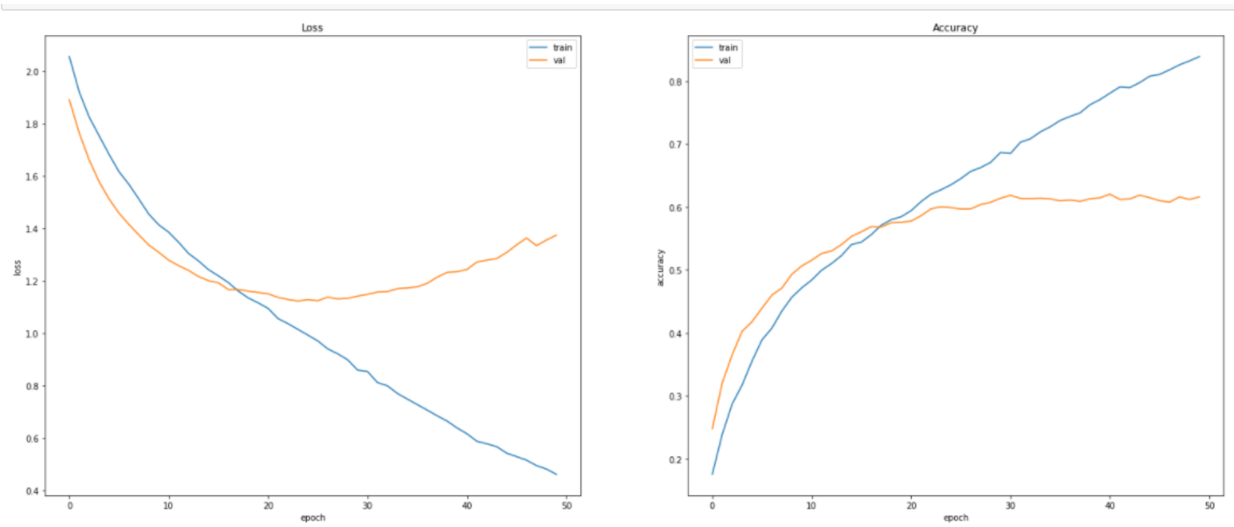
```
csv_path = './feelings_skfold2.csv'

train_generator = DataGenerator(
    batch_size = 128,
    csv_path = csv_path,
    fold = 1,  #valid_data로 fold=1인 자료들 이용
    image_size = 48, #우리 사진이 48X48 이니까!
    mode = 'train',
    shuffle = True
)

valid_generator = DataGenerator(
    batch_size = 128,
    csv_path = csv_path,
    fold = 1,
    image_size = 48,
    mode = 'val',
    shuffle = True
)
```

4) model.fit

- epoch: 50



- 약간씩 valid accuracy가 흔들리다가 35~40 정도 이후부터 계속 흔들림(0.01 정도?)

  => 이 시점 이후로 overfitting된 건가...?

- 정확도 조금 더 높일 필요가 있어 보인다.

```
Epoch 1/50
loss: 2.0564 - accuracy: 0.1756 - val_loss: 1.8919 - val_accuracy:
0.2477
Epoch 2/50
loss: 1.9213 - accuracy: 0.2389 - val_loss: 1.7646 - val_accuracy:
0.3205
Epoch 3/50
loss: 1.8254 - accuracy: 0.2872 - val_loss: 1.6619 - val_accuracy:
0.3654
```

Epoch 4/50
loss: 1.7541 - accuracy: 0.3172 - val_loss: 1.5787 - val_accuracy:
0.4027
Epoch 5/50
loss: 1.6830 - accuracy: 0.3555 - val_loss: 1.5130 - val_accuracy:
0.4179
Epoch 6/50
loss: 1.6169 - accuracy: 0.3890 - val_loss: 1.4587 - val_accuracy:
0.4393
Epoch 7/50
loss: 1.5685 - accuracy: 0.4077 - val_loss: 1.4155 - val_accuracy:
0.4602
Epoch 8/50
loss: 1.5124 - accuracy: 0.4348 - val_loss: 1.3755 - val_accuracy:
0.4712
Epoch 9/50
loss: 1.4549 - accuracy: 0.4567 - val_loss: 1.3369 - val_accuracy:
0.4932
Epoch 10/50
loss: 1.4141 - accuracy: 0.4717 - val_loss: 1.3092 - val_accuracy:
0.5065
Epoch 11/50
loss: 1.3860 - accuracy: 0.4841 - val_loss: 1.2793 - val_accuracy:
0.5155
Epoch 12/50
loss: 1.3474 - accuracy: 0.4997 - val_loss: 1.2584 - val_accuracy:
0.5262
Epoch 13/50
loss: 1.3048 - accuracy: 0.5103 - val_loss: 1.2401 - val_accuracy:
0.5302
Epoch 14/50
loss: 1.2764 - accuracy: 0.5229 - val_loss: 1.2170 - val_accuracy:
0.5403
Epoch 15/50
loss: 1.2432 - accuracy: 0.5403 - val_loss: 1.2012 - val_accuracy:
0.5533
Epoch 16/50
loss: 1.2194 - accuracy: 0.5443 - val_loss: 1.1928 - val_accuracy:
0.5607
Epoch 17/50
loss: 1.1938 - accuracy: 0.5562 - val_loss: 1.1673 - val_accuracy:
0.5688
Epoch 18/50
- loss: 1.1614 - accuracy: 0.5712 - val_loss: 1.1675 - val_accuracy:
0.5683
Epoch 19/50
loss: 1.1352 - accuracy: 0.5799 - val_loss: 1.1609 - val_accuracy:
0.5751
Epoch 20/50
step - loss: 1.1164 - accuracy: 0.5845 - val_loss: 1.1563 -
val_accuracy: 0.5759
Epoch 21/50
loss: 1.0953 - accuracy: 0.5942 - val_loss: 1.1507 - val_accuracy:
0.5776
Epoch 22/50
loss: 1.0564 - accuracy: 0.6085 - val_loss: 1.1373 - val_accuracy:
0.5861

```
Epoch 23/50
loss: 1.0371 - accuracy: 0.6203 - val_loss: 1.1292 - val_accuracy:
0.5971
Epoch 24/50
loss: 1.0152 - accuracy: 0.6273 - val_loss: 1.1233 - val_accuracy:
0.6005
Epoch 25/50
loss: 0.9933 - accuracy: 0.6352 - val_loss: 1.1287 - val_accuracy:
0.5990
Epoch 26/50
loss: 0.9711 - accuracy: 0.6451 - val_loss: 1.1244 - val_accuracy:
0.5968
Epoch 27/50
loss: 0.9407 - accuracy: 0.6566 - val_loss: 1.1385 - val_accuracy:
0.5971
Epoch 28/50
loss: 0.9220 - accuracy: 0.6626 - val_loss: 1.1312 - val_accuracy:
0.6038
Epoch 29/50
loss: 0.8991 - accuracy: 0.6709 - val_loss: 1.1337 - val_accuracy:
0.6075
Epoch 30/50
loss: 0.8600 - accuracy: 0.6867 - val_loss: 1.1420 - val_accuracy:
0.6140
Epoch 31/50
loss: 0.8544 - accuracy: 0.6850 - val_loss: 1.1490 - val_accuracy:
0.6185
Epoch 32/50
loss: 0.8123 - accuracy: 0.7028 - val_loss: 1.1578 - val_accuracy:
0.6137
Epoch 33/50
loss: 0.7997 - accuracy: 0.7083 - val_loss: 1.1593 - val_accuracy:
0.6134
Epoch 34/50
loss: 0.7708 - accuracy: 0.7193 - val_loss: 1.1706 - val_accuracy:
0.6140
Epoch 35/50
loss: 0.7493 - accuracy: 0.7274 - val_loss: 1.1734 - val_accuracy:
0.6129
Epoch 36/50
loss: 0.7283 - accuracy: 0.7375 - val_loss: 1.1773 - val_accuracy:
0.6098
Epoch 37/50
loss: 0.7071 - accuracy: 0.7438 - val_loss: 1.1910 - val_accuracy:
0.6112
Epoch 38/50
loss: 0.6856 - accuracy: 0.7495 - val_loss: 1.2142 - val_accuracy:
0.6089
Epoch 39/50
loss: 0.6653 - accuracy: 0.7624 - val_loss: 1.2327 - val_accuracy:
0.6131
Epoch 40/50
loss: 0.6392 - accuracy: 0.7705 - val_loss: 1.2352 - val_accuracy:
0.6146
Epoch 41/50
loss: 0.6168 - accuracy: 0.7806 - val_loss: 1.2438 - val_accuracy:
0.6205
```

```
Epoch 42/50
loss: 0.5882 - accuracy: 0.7906 - val_loss: 1.2721 - val_accuracy:
0.6120
Epoch 43/50
loss: 0.5785 - accuracy: 0.7895 - val_loss: 1.2794 - val_accuracy:
0.6129
Epoch 44/50
loss: 0.5675 - accuracy: 0.7976 - val_loss: 1.2854 - val_accuracy:
0.6188
Epoch 45/50
loss: 0.5428 - accuracy: 0.8077 - val_loss: 1.3089 - val_accuracy:
0.6148
Epoch 46/50
loss: 0.5296 - accuracy: 0.8105 - val_loss: 1.3378 - val_accuracy:
0.6100
Epoch 47/50
loss: 0.5159 - accuracy: 0.8178 - val_loss: 1.3640 - val_accuracy:
0.6078
Epoch 48/50
loss: 0.4953 - accuracy: 0.8257 - val_loss: 1.3346 - val_accuracy:
0.6160
Epoch 49/50
loss: 0.4820 - accuracy: 0.8319 - val_loss: 1.3562 - val_accuracy:
0.6120
Epoch 50/50
loss: 0.4620 - accuracy: 0.8388 - val_loss: 1.3748 - val_accuracy:
0.6160
```