얼굴인식 1차 과제

생체인증보안

사이버보안전공

1871085

주유연

목차

01

Data

- Read Data
- Data Augmentation

02

Split Data

- train_test_split

03

Model

- CNN Model
- Hyperparameter tuning
- Training Result

04

Evaluate Model

- Confusion Matrix

Given Data

- 350명의 얼굴 데이터가 1명당 3개씩 총 1050개의 얼굴 이미지 데이터가 주어짐.
- 각 파일명의 앞쪽 숫자는 label을 의미함.

```
# 이미지 목록
images = glob.glob('./02_face_training/*.BMP')
len(images)
```

```
1050
```

```
r = re.compile('₩d+')

img = [] # 이미지
label = [] # 라벨

for fname in images:
    l = r.findall(fname)[1]
    label.append(l)
    im = pilimg.open(fname)
    pix = np.array(im)/255. # Normalize
    img.append(pix)
```

```
X = np.array(img)
X = X.reshape(X.shape[0],56,46,1)
X.shape # img shape
```

```
(1050, 56, 46, 1)
```

```
y = np.array(label, dtype='int32')
y # label
```

```
array([128, 127, 128, ..., 336, 335, 336], dtype=int32)
```

데이터 불러오기

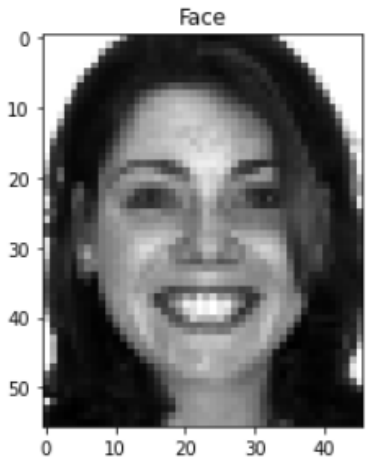
- glob 함수를 이용해 파일명을 불러오고, 파일명에서 label을 읽어 별도의 리스트에 저장함.
- 불러온 이미지를 numpy array로 변환함.

얼굴 이미지 확인

- Normalization

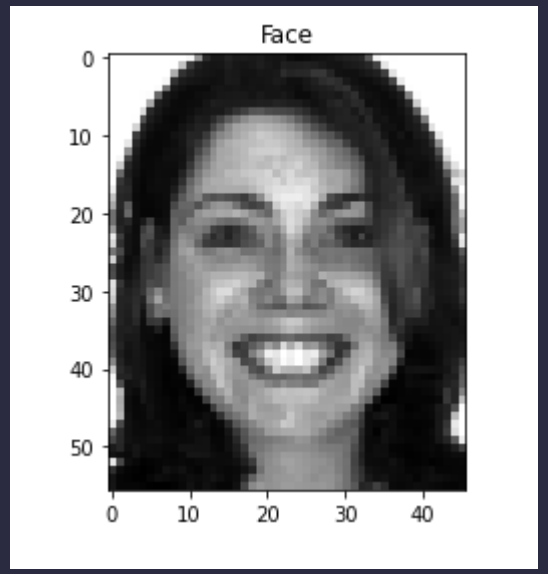
```
# 얼굴 이미지
plt.title('Face')
plt.imshow(X[0].reshape(56,46), cmap='gray')
print(X[0].shape)
print(y[0])
```

```
(56, 46, 1)
128
```

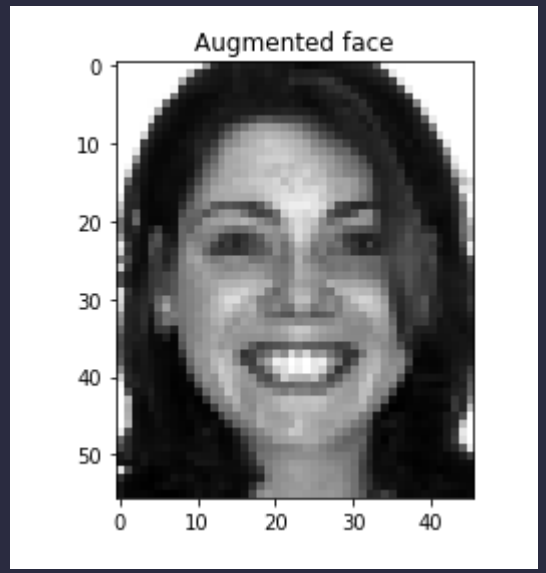


Data Augmentation

- 1명당 3개의 데이터는 얼굴을 구별하기에 부족하다고 판단.
- 얼굴 이미지를 blur처리하고 이미지의 각도를 비틀어 새 데이터를 생성함. (크게 변형되지 않도록)
- 각 이미지당 9개의 추가 augmented image를 생성하여 총 9450개의 데이터를 확보함.



- GaussianBlur
- Affine – translate, rotate



최종 Data

```
# 생성한 이미지
x_d = np.array(x_d)
y_d = np.array(y_d)
print(x_d.shape)
print(y_d.shape)
```

```
(9450, 56, 46, 1)
(9450,)
```

```
# 기존 이미지
print(X.shape)
print(y.shape)
```

```
(1050, 56, 46, 1)
(1050,)
```

```
# 기존 이미지, 생성 이미지 합치기
X_data = np.concatenate([X, x_d], axis=0)
y_data = np.concatenate([y, y_d], axis=0)
print(X_data.shape)
print(y_data.shape)
```

```
(10500, 56, 46, 1)
(10500,)
```

Split Data

- sklearn.model_selection의 train_test_split 함수 이용.
- Train : Test의 비율은 8:2로 하고, stratify 옵션을 주어 label이 균등하게 나뉘도록 함.
- X_test, y_test는 모델 훈련 후 evaluate으로 이용함.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data,y_data, test_size=0.2, shuffle=True, stratify=y_data, random_state=101)
```

```
X_train=X_train.astype('float32')
y_train=y_train.astype('int32')
X_test=X_test.astype('float32')
y_test=y_test.astype('int32')
```

```
X_train = X_train.reshape(X_train.shape[0],56,46,1)
X_test = X_test.reshape(X_test.shape[0],56,46,1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(8400, 56, 46, 1) (8400,) (2100, 56, 46, 1) (2100,)
```

Label one-hot encoding (원-핫 인코딩)

- 모델 학습 시 loss function으로 categorical_crossentropy를 사용하기 위해 label을 원-핫 인코딩해줌.
- Label 값은 1~350 범위로 저장되어 있는데 이를 희소행렬로 변환해준다.
- 이 때, 0에 해당하는 data가 없기에 학습 자체에는 영향을 주지 못한다고 판단하여 따로 label을 변경하진 않았음.

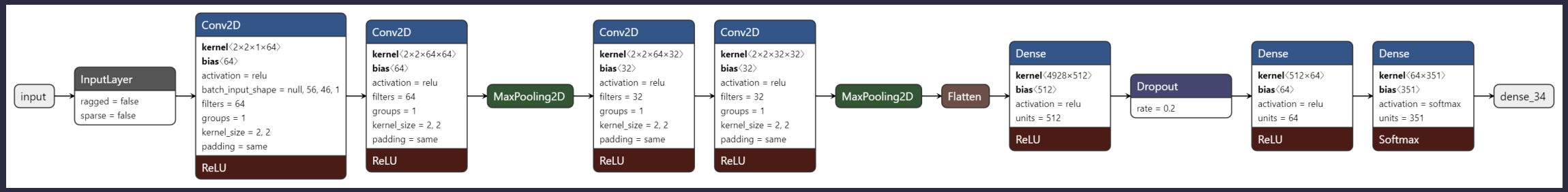
```
from keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes=351)
y_test = to_categorical(y_test, num_classes=351)
```

시도해본 모델

	Testset rate	Learning rate	Layer	Dropout (Conv/MaxPool)	Dropout (Dense)	regularizer	optimizer	epoch*fold
01	0.2	0.00001	1층	X	X	X	Nadam	10*4
02	0.2	0.001	1층	X	X	X	adam	15*4
03	0.2	0.00001	2층	0.5 (층마다)	X	X	adam	20*4
04	0.2	0.00001	2층	0.5 (층마다)	X	Dense(64) l2	Adam	20*4
05	0.1	0.00001	2층	X	X	Dense(64) l2	Adam	10*4
06	0.1	0.00001	2층	X	0.1 * 2번	Dense(64) l2	Adam	10*4
07	0.1	0.00001	2층	X	0.1 * 2번	Dense(64) l2	Adam	15*4
08	0.2	0.00001	2층	X	0.1 * 1번	Dense(64) l2	Adam	10*4
09	0.2	0.00001	2층	X	X	Dense(512) l2	Adam	10*4
10	0.1	0.00001	2층	X	0.2 * 2번	Dense(64) l2	Adam	10*4

CNN (Convolution Neural Network)

- Convolution-MaxPooling2D 레이어를 2번 쌓아 얼굴 이미지에서 특징을 추출할 수 있게 함.
- 이후 Flatten, Dense 레이어를 넣어 최종적으로 softmax함수를 적용해 350개의 노드를 출력하도록 함.
- Dropout 및 kernel_regularizer으로 오버피팅을 줄이고자 했음.



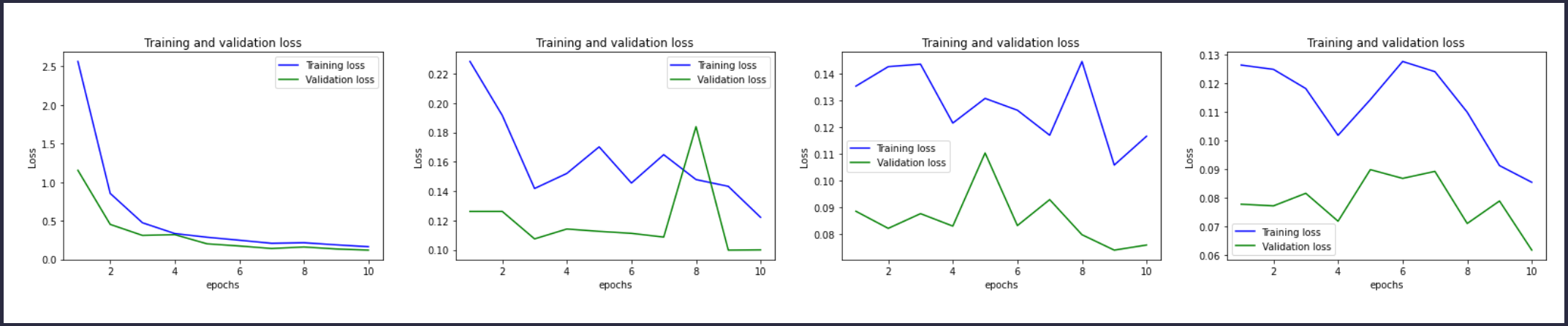
Model summary

```python
def build_model():
    learning_rate = 0.00001
    width = 56
    height = 46
    METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy')
    ]
    model = Sequential()
    model.add(Conv2D(filters=64, kernel_size=(2,2), padding='same', activation='relu', input_shape=(width,
    model.add(Conv2D(filters=64, kernel_size=(2,2), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters=32, kernel_size=(2,2), padding='same', activation='relu'))
    model.add(Conv2D(filters=32, kernel_size=(2,2), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.1))
    model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
    model.add(Dropout(0.1))
    model.add(Dense(351, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

model = build_model()
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_40 (Conv2D)           (None, 56, 46, 64)        320
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 56, 46, 64)        16448
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 28, 23, 64)        0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 28, 23, 32)        8224
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 28, 23, 32)        4128
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 14, 11, 32)        0
_________________________________________________________________
flatten_10 (Flatten)         (None, 4928)              0
_________________________________________________________________
dense_32 (Dense)             (None, 512)               2523648
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_33 (Dense)             (None, 64)                32832
_________________________________________________________________
dense_34 (Dense)             (None, 351)               22815
=================================================================
Total params: 2,608,415
Trainable params: 2,608,415
Non-trainable params: 0
_________________________________________________________________
```

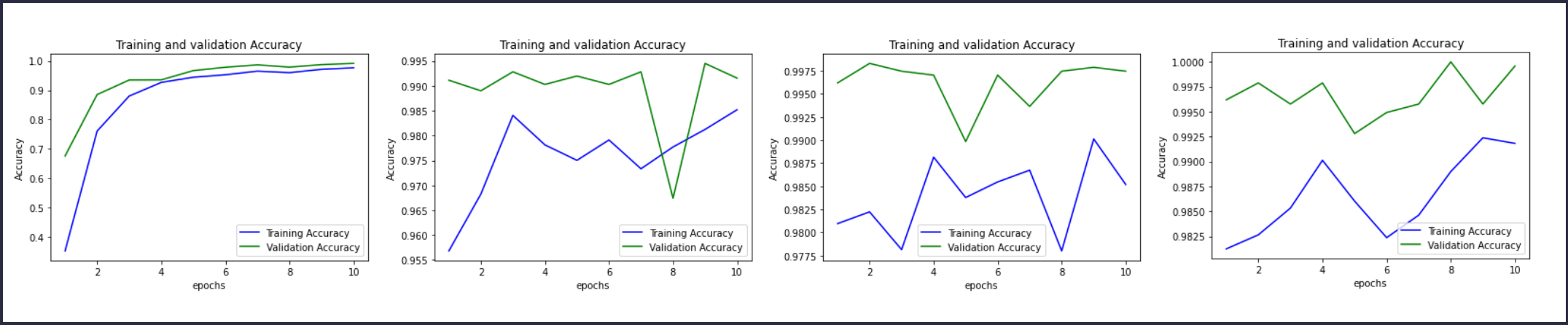
Training and Validation Loss

- KFold (k=4) Validation
- Fold 별 Train Loss와 Validation Loss
- Fold마다 loss가 줄어들었음을 확인할 수 있음.



Training and Validation Accuracy

- KFold (k=4) Validation
- Fold 별 Train Accuracy 와 Validation Accuracy
- Fold마다 Accuracy가 증가하며 1에 수렴함을 확인할 수 있음.

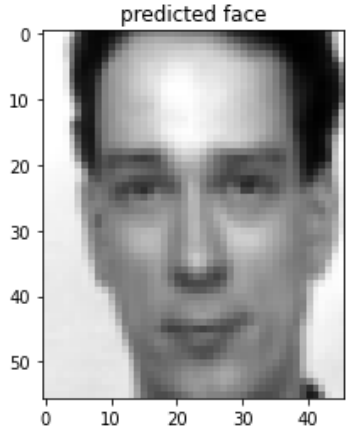


예측 결과 확인

- Test set으로 분리한 데이터의 일부의 실제 label과 예측 label을 확인함.

```
# 예측결과
plt.title('predicted face')
this_img = X_test[1].reshape(56,46)
plt.imshow(this_img, cmap='gray')
print(this_img.shape)
print('예측: ', preds[1])
print('실제: ', y_test_origin[1])
#X_test[0]
```

```
(56, 46)
예측:  342
실제:  342
```



Accuracy, Precision, Recall, F1 score

- Multi-class model임을 고려하여 평가지표들을 계산함.
- 각 label마다의 confusion matrix TP, FN, FP, TN의 평균인 Macro Average를 계산함.

```
from sklearn.metrics import multilabel_confusion_matrix
cm = multilabel_confusion_matrix(y_test_origin, preds)
```

```
cm
```

```
array([[[1047,    0],
        [   0,    3]],

       [[1047,    0],
        [   0,    3]],

       [[1047,    0],
        [   0,    3]],

       ...,

       [[1047,    0],
        [   0,    3]],

       [[1047,    0],
        [   0,    3]],

       [[1047,    0],
        [   0,    3]]])
```

```
# Macro Average로 평가지표 계산
accuracy_avg = 0
precision_avg = 0
recall_avg = 0
F1_avg = 0

for i in range(len(cm_mat)):
    accuracy_avg += accuracy(cm_mat[i])
    precision_avg += precision(cm_mat[i])
    recall_avg += recall(cm_mat[i])
    F1_avg += F1(cm_mat[i])

accuracy_avg /= len(cm_mat)
precision_avg /= len(cm_mat)
recall_avg /= len(cm_mat)
F1_avg /= len(cm_mat)

print('accuracy: ', accuracy_avg)
print('precision: ', precision_avg)
print('recall: ', recall_avg)
print('F1: ', F1_avg)
```

```
accuracy:  0.9999945578231292
precision:  0.9999972737186477
recall:  0.9999972711147497
F1:  0.9999972711141274
```

Test 예측 결과

	Image	Answer
15	1	2
28	2	270
359	3	188
248	4	214
290	5	303
...	...	...
199	696	93
402	697	182
481	698	201
534	699	15
551	700	281

700 rows × 2 columns

개선방향

- Data Augmentation 시 생성하는 데이터양을 더 늘리거나, 기존 이미지의 특성을 최대한 보존할 수 있도록 augmentation 옵션을 조절.
- 모델 학습 시 하이퍼파라미터 값 조절 (learning_rate, dropout rate, regularization)
- Cross Validation Fold 수 조절

감사합니다