멀티모달 1차 과제 2Model

생체인증보안

사이버보안전공
1871085
주유연

목차

01

Data

- Read Data
- Data Augmentation

02

Split Data

- train_test_split

03

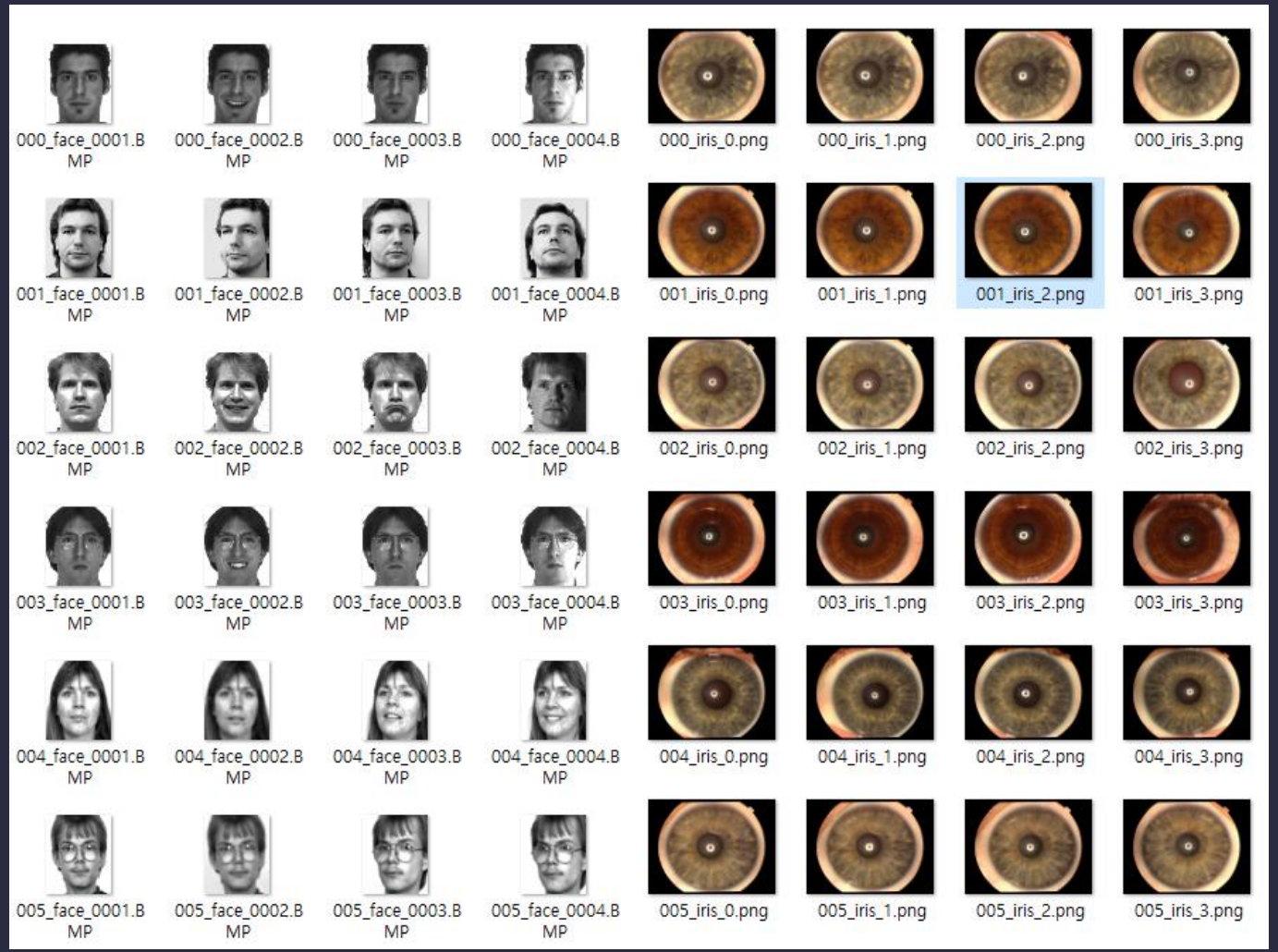
Model

- Pre-trained Model
- Training Result

04

Evaluate Model

- Confusion Matrix



Given Data

- 64명의 얼굴, 홍채 데이터가 1명당 4개씩 총 256쌍의 이미지 데이터가 주어짐.
- 각 파일명의 앞쪽 숫자는 label을 의미함.

```
# 이미지 목록
images_face = sorted(glob.glob('./04_multimodal_training/*.BMP'))
images_iris = sorted(glob.glob('./04_multimodal_training/*.png'))

print(len(images_face), len(images_iris))
```

```
256 256
```

```
r = re.compile('₩d+')

img_face = [] # 이미지
img_iris = [] # 이미지
label_face = [] # 라벨
label_iris = [] # 라벨

for fname in images_face:
    l = r.findall(fname)[1]
    label_face.append(l)
    im = pilimg.open(fname)
    pix = np.array(im)/255.
    pix = pix.reshape(pix.shape[0], pix.shape[1], 1)
    pix = tf.image.grayscale_to_rgb(tf.convert_to_tensor(pix)) # ResNet50 위해 rgb 이미지로 변환
    img_face.append(pix)

for fname in images_iris:
    l = r.findall(fname)[1]
    label_iris.append(l)
    im = pilimg.open(fname)
    im = im.resize((int(im.width*0.3), int(im.height*0.3))) # 이미지 줄이기
    pix = np.array(im)/255. # Normalize
    img_iris.append(pix)
```

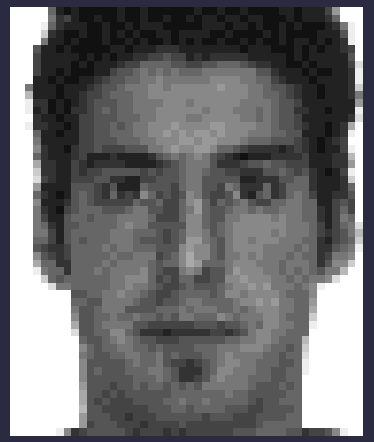
```
X_face = np.array(img_face)
X_iris = np.array(img_iris)
print(X_face.shape, X_iris.shape)
```

```
(256, 56, 46, 3) (256, 172, 230, 3)
```

데이터 불러오기

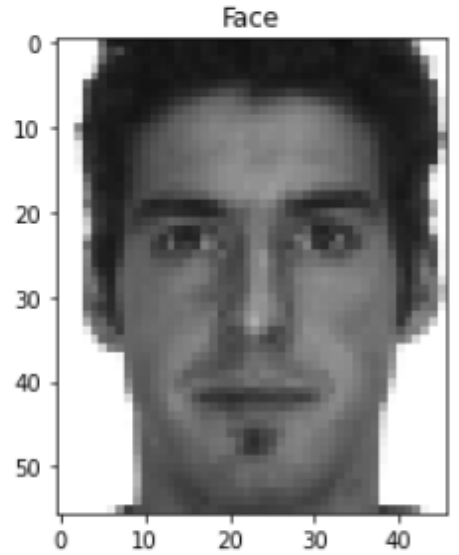
- glob 함수를 이용해 파일명을 불러오고, 파일명에서 label을 읽어 별도의 리스트에 저장함.
- Out of memory 문제를 해결하기 위한 방법으로 이미지의 크기를 줄임 (30%)
- 얼굴 이미지의 경우 grayscale로 되어 있는데, Pre-trained 모델을 이용하기 위해 RGB 이미지로 변환함.

얼굴 이미지 확인



```
# 0번 사람의 얼굴 이미지
plt.title('Face')
plt.imshow(X_face[0])
print(X_face[0].shape)
print(y_face[0])
```

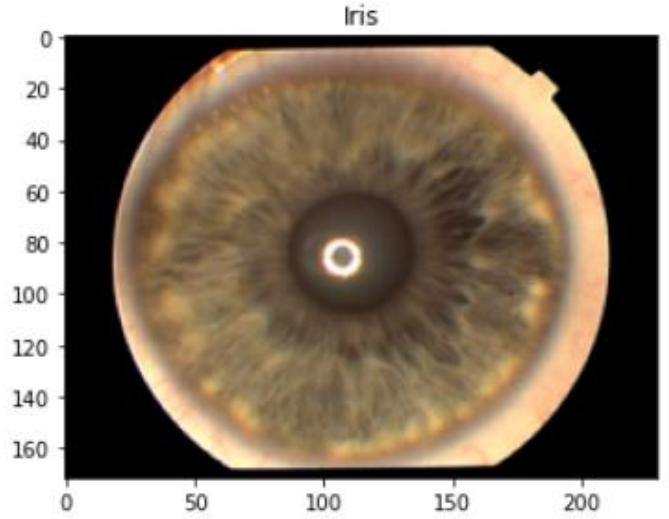
```
(56, 46, 3)
0
```



홍채 이미지 확인

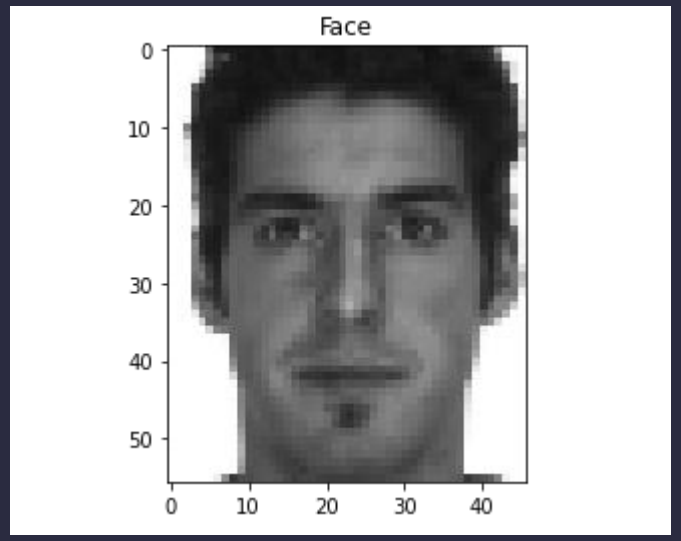
```
# 0번 사람의 홍채 이미지
plt.title('Iris')
plt.imshow(X_iris[0])
print(X_iris[0].shape)
print(y_iris[0])
```

```
(172, 230, 3)
0
```

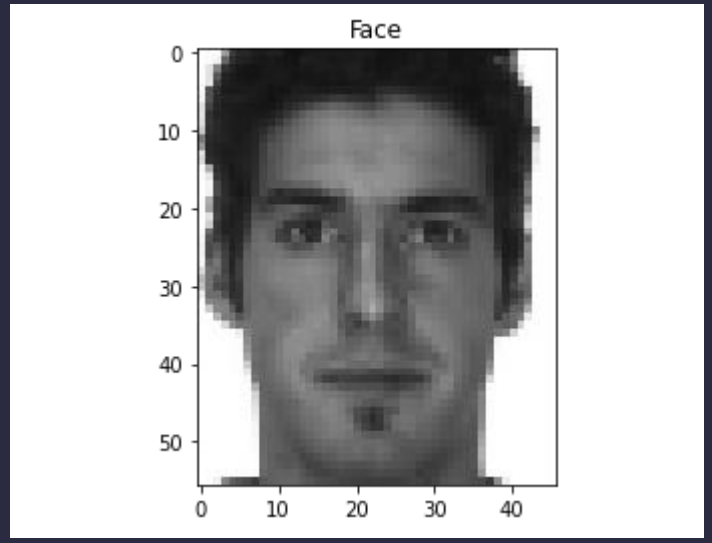


Data Augmentation

- 1명당 2개의 데이터는 얼굴/홍채를 구별하기에 부족하다고 판단.
- 이미지의 밝기 조절, 수평반전, 블러 등을 랜덤으로 적용한 image augmentation 사용.
- 각 이미지당 9개의 추가 augmented image를 생성하여 총 2560개의 데이터를 확보함.



- Add(0, 1)
- Fliplr(0.5)
- GaussianBlur(sigma=(0.0, 0.9))



최종 Data

```
# 기존 이미지
print(X_face.shape)
print(y_face.shape)
print(X_iris.shape)
print(y_iris.shape)

# 생성한 이미지
face_x_d = np.array(face_x_d)
face_y_d = np.array(face_y_d)
print(face_x_d.shape)
print(face_y_d.shape)
iris_x_d = np.array(iris_x_d)
iris_y_d = np.array(iris_y_d)
print(iris_x_d.shape)
print(iris_y_d.shape)
```

```
(256, 56, 46, 3)
(256,)
(256, 172, 230, 3)
(256,)
(2304, 56, 46, 3)
(2304,)
(2304, 172, 230, 3)
(2304,)
```

```
# 기존 이미지, 생성 이미지 합치기
X_face = np.concatenate([X_face, face_x_d], axis=0)
y_face = np.concatenate([y_face, face_y_d], axis=0)
X_iris = np.concatenate([X_iris, iris_x_d], axis=0)
y_iris = np.concatenate([y_iris, iris_y_d], axis=0)
print(X_face.shape)
print(X_iris.shape)
```

```
(2560, 56, 46, 3)
(2560, 172, 230, 3)
```

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_face, X_test_face, y_train_face, y_test_face = train_test_split(X_face,y_face, test_size=0.2, shuffle=True, stratify=y_face,
X_iris_face, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris,y_iris, test_size=0.2, shuffle=True, stratify=y_iris,
```

label 똑같이 나누어진 것 확인

```
print(np.array_equal(y_train_face,y_train_iris))
print(np.array_equal(y_test_face, y_test_iris))
```

```
True
True
```

```
print(X_train_face.shape, y_train_face.shape, X_test_face.shape, y_test_face.shape)
print(X_iris_face.shape, y_train_iris.shape, X_test_iris.shape, y_test_iris.shape)
```

```
(2048, 56, 46, 3) (2048,) (512, 56, 46, 3) (512,)
(2048, 172, 230, 3) (2048,) (512, 172, 230, 3) (512,)
```

모델 학습

- Stratified K-Fold Cross Validation으로 모델을 평가함.
- 데이터셋을 split하기 위해 label의 one-hot encoding은 학습 직전에 함수를 적용함

```python
#Cross validation
from sklearn.model_selection import KFold
from keras.utils import to_categorical

kf = KFold(n_splits=4, shuffle=True, random_state=42)
all_history_face = [] # face 결과 저장
all_history_iris = [] # iris 결과 저장

def score_model(cv=None):
    if cv is None:
        cv = KFold(n_splits=4, random_state=42, shuffle=True)

    i=0
    for train_fold_index, val_fold_index in cv.split(X_train_face, y_train_data):
        # index를 split하는 것이기 때문에 face, iris에 동일하게 적용 가능 (random_state)

        i=i+1
        print('Fold #',i)

        # Get the training data
        X_train_face_fold = X_train_face[train_fold_index]
        X_train_iris_fold = X_train_iris[train_fold_index]
        y_train_fold = y_train_data[train_fold_index]

        # Get the validation data
        X_val_face_fold = X_train_face[val_fold_index]
        X_val_iris_fold = X_train_iris[val_fold_index]
        y_val_fold = y_train_data[val_fold_index]

        print(X_train_face_fold.shape)
        print(X_train_iris_fold.shape)
        print(y_val_fold.shape)

        # 원핫인코딩
        y_train_fold = to_categorical(y_train_fold, num_classes=64)
        y_val_fold = to_categorical(y_val_fold, num_classes=64)
```

```python
# Fit the model - Face
print('<Train Face Model>')
facemodel = build_model_face()
facemodel_obj = facemodel.fit(X_train_face_fold, y_train_fold,
                              epochs=1000,
                              batch_size=128,
                              validation_data=(X_val_face_fold, y_val_fold),
                              verbose=1)
facemodel.save('./model/face_model3_'+str(i)+'.h5')
all_history_face.append(facemodel_obj.history)

# Fit the model - Iris
print('<Train Iris Model>')
irismodel = build_model_iris()
irismodel_obj = irismodel.fit(X_train_iris_fold, y_train_fold,
                              epochs=100,
                              #batch_size=32,
                              validation_data=(X_val_iris_fold, y_val_fold),
                              verbose=1)
irismodel.save('./model/iris_model3_'+str(i)+'.h5')
all_history_iris.append(irismodel_obj.history)
```

얼굴 모델

- 얼굴 이미지의 크기가 홍채에 비해 작으므로 ResNet보다 구조가 단순한 Pre-trained 모델인 VGG19 네트워크를 이용함.
- 오버피팅을 방지해주기 위해 L2 kernel_regularizer를 적용함.

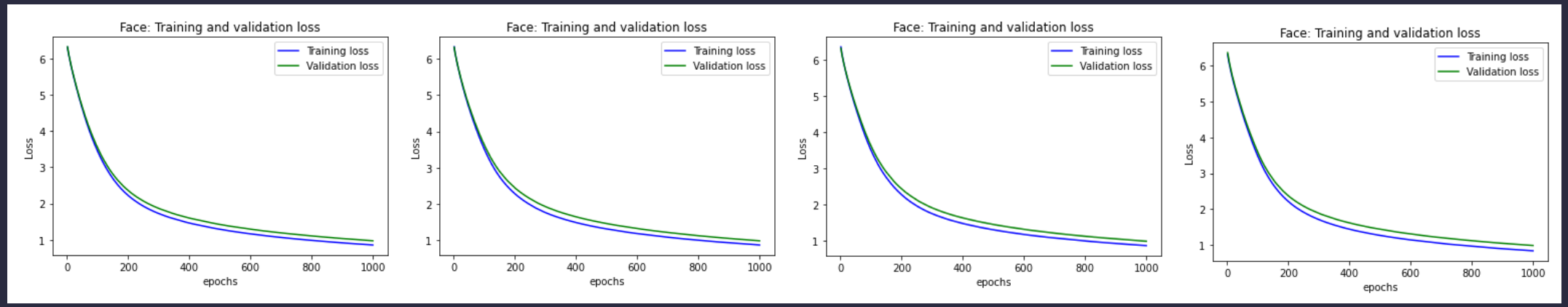
```
def build_model_face():
    learning_rate = 0.00005
    METRICS = [
      tf.keras.metrics.CategoricalAccuracy(name='accuracy')
    ]
    input_ = Input(shape=X_face[0].shape, name='face_input')
    base_model = VGG19(include_top=False, input_tensor=input_)
    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = Flatten()(x)
    x = Dense(128, activation='relu', kernel_regularizer='l2')(x)
    output = Dense(64, activation='softmax')(x)

    model = Model(input_, output)

    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=learning_rate),
    return model
```

홍채 모델

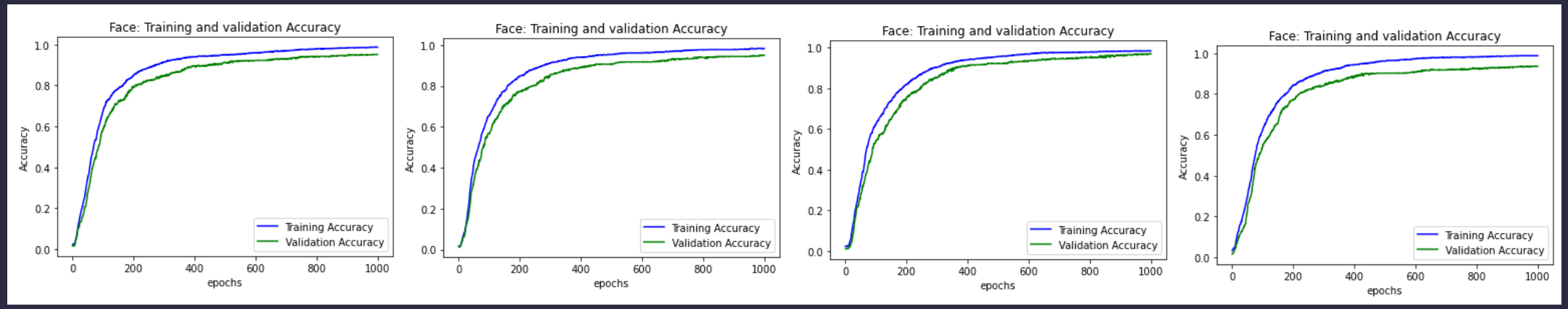
- Pre-trained 모델인 ResNet50V2 네트워크로 학습.
- 오버피팅을 방지해주기 위해 L2 kernel_regularizer를 적용함.
- 얼굴과 달리 홍채는 오버피팅이 심하게 발생하여 Dropout 레이어를 추가해줌.

```
def build_model_iris():
    learning_rate = 0.0001
    METRICS = [
      tf.keras.metrics.CategoricalAccuracy(name='accuracy')
    ]
    input_ = Input(shape=X_iris[0].shape, name='iris_input')
    base_modelB = ResNet50V2(include_top=False, input_tensor=input_)
    for layer in base_modelB.layers[:44]:
        layer.trainable=False

    y = base_modelB.output
    y = MaxPooling2D(pool_size=(2,2))(y)
    y = Flatten()(y)
    y = Dense(512, activation='relu', kernel_regularizer='l2')(y)
    y = Dropout(0.5)(y)
    y = Dense(128, activation='relu', kernel_regularizer='l2')(y)
    output = Dense(64, activation='softmax')(y)

    model = Model(input_, output)

    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=learning_rate),
    return model
```

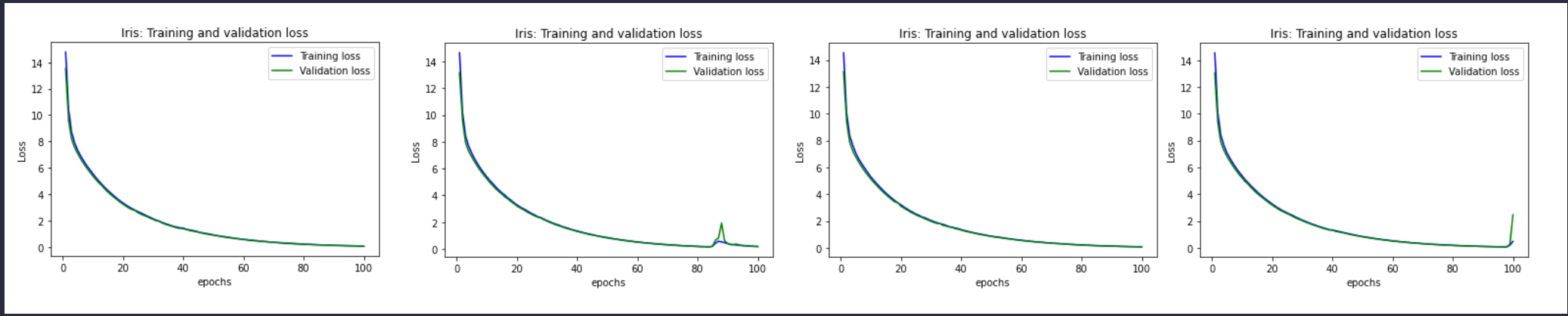
Face: Training and Validation Loss & Accuracy



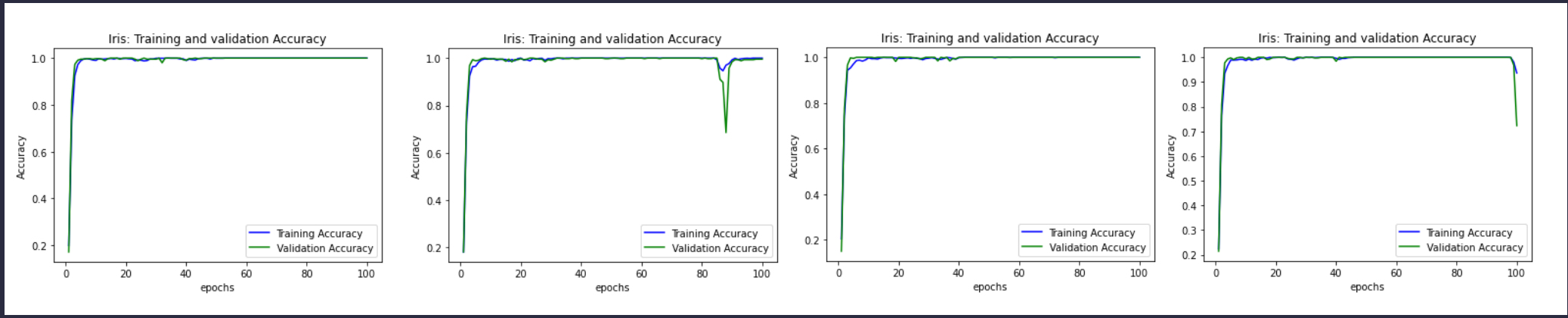
FOLD #1 FOLD #2 FOLD #3 FOLD #4



Iris: Training and Validation Loss & Accuracy



FOLD #1 FOLD #2 FOLD #3 FOLD #4



예측 결과 확인

- 두 모델의 예측 결과 중 더 높은 확률의 예측 결과를 최종 예측으로 선택함.
- Test set으로 분리한 데이터의 일부의 실제 label과 예측 label을 확인함.

```
facepreds = facemodel.predict(face_t)
irispreds = irismodel.predict(iris_t)

preds = np.where(irispreds>facepreds, irispreds, facepreds) # 예측 확률이 높은쪽 선택
preds = np.argmax(preds, axis=1)
preds
```

```
array([27, 36,  2, 46, 50, 10, 61, 49, 59, 56, 13, 31, 37, 30, 25,  0, 41,
       58,  9, 20, 48, 47, 22, 42, 24,  3,  6, 55, 63, 21, 16, 34, 11, 57,
       35, 32, 40, 62, 19, 52, 33, 15,  1, 29, 54, 17, 26, 53, 28, 18, 14,
       43, 60,  4, 39, 45, 38, 23, 51, 44,  5, 12,  8,  7])
```

```
# 예측결과
plt.title('predicted iris')
this_img = X_test_face[0]
plt.imshow(this_img, cmap='gray')
print(this_img.shape)
print('예측: ', preds[0])
print('실제: ', y_test_data[0])
```

```
(56, 46, 3)
예측:  41
실제:  41
```

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 classification_report
print(classification_report(y_test_data,preds, zero_division=1))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	12
1	1.00	1.00	1.00	12
2	1.00	1.00	1.00	12
3	1.00	1.00	1.00	12
4	1.00	1.00	1.00	12
5	1.00	1.00	1.00	12
6	1.00	1.00	1.00	12
7	1.00	1.00	1.00	12
8	1.00	1.00	1.00	12
9	1.00	1.00	1.00	12

	Precision	Recall	F1-score	
accuracy			1.00	768
macro avg	1.00	1.00	1.00	768
weighted avg	1.00	1.00	1.00	768

Test 예측 결과

	Image	Answer
0	0	27
11	1	31
22	2	22
33	3	57
44	4	54
...	...	...
54	59	39
56	60	38
57	61	23
58	62	51
59	63	44

64 rows × 2 columns

개선 방향

- 최종 prediction을 각 모델이 낸 정답 확률 중 더 높은 쪽의 결과를 택했는데 두 모델의 학습 결과를 적절히 섞어 반영할 수 있는 방법을 찾아보고 싶다.
- 홍채의 일부 fold에서 loss가 급격히 증가하는 부분이 있는데, 이를 해결하기 위해 lr을 낮춰봐야겠다.

감사합니다