생체인식보안

지문 인식

사이버보안전공 1871095 문수빈

INDEX

01 Image Augmentation

02 Build Model

03 Training Data

04 Test Data

01. Image Augmentation

01. Image Augmentation

1차 제출에서 높은 Accuracy를 보였으며 실제 정답 결과가 좋아 코드에 이전에 시도해보려 생각했던 Image Augmentation을 추가했다.

모든 지문 정보가 항상 정방향으로 들어오지 않으며, 일부는 잘려서 들어온다는 점을 고려했다.

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array

케라스의 이미지 전처리 모듈을 불러온다.

```
datagen = ImageDataGenerator(
                rotation_range = 40,
               width_shift_range = 0.1,
               height_shift_range = 0.1)
```

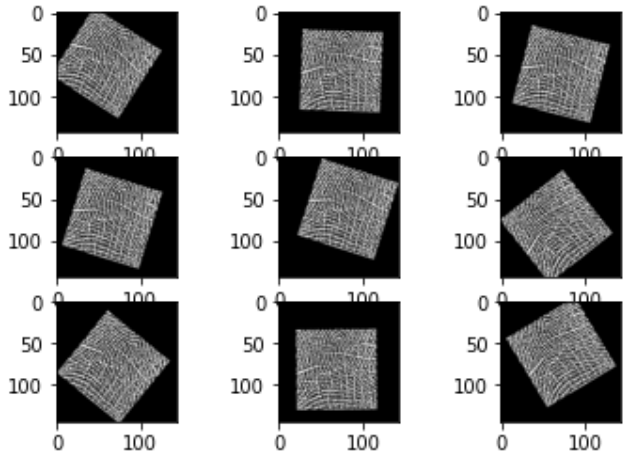
지문의 특징점 길이가 달라지거나, 왜곡될 필요가 없어

ImageDataGenerator에서 shear, zoom 등의 속성은 제외하고 rotation과 shift 인자만 이용해 데이터를 추가했다.

01. Image Augmentation

01. 이미지가 어떻게 변화되는지 보기 위해 3번의 지문을 Augmentation 해 9개로 늘리고, 임시로 출력한 결과이다.

```
new_image = img_to_array(image)
samples = expand_dims(new_image, 0)

it= datagen.flow(samples, batch_size=1)

for i in range(9):
    batch = it.next()
    image = batch[0].astype('uint8')
    plt.subplot(3, 3, i+1)
    np_image = np.array(image)
    plt.imshow(image)
```



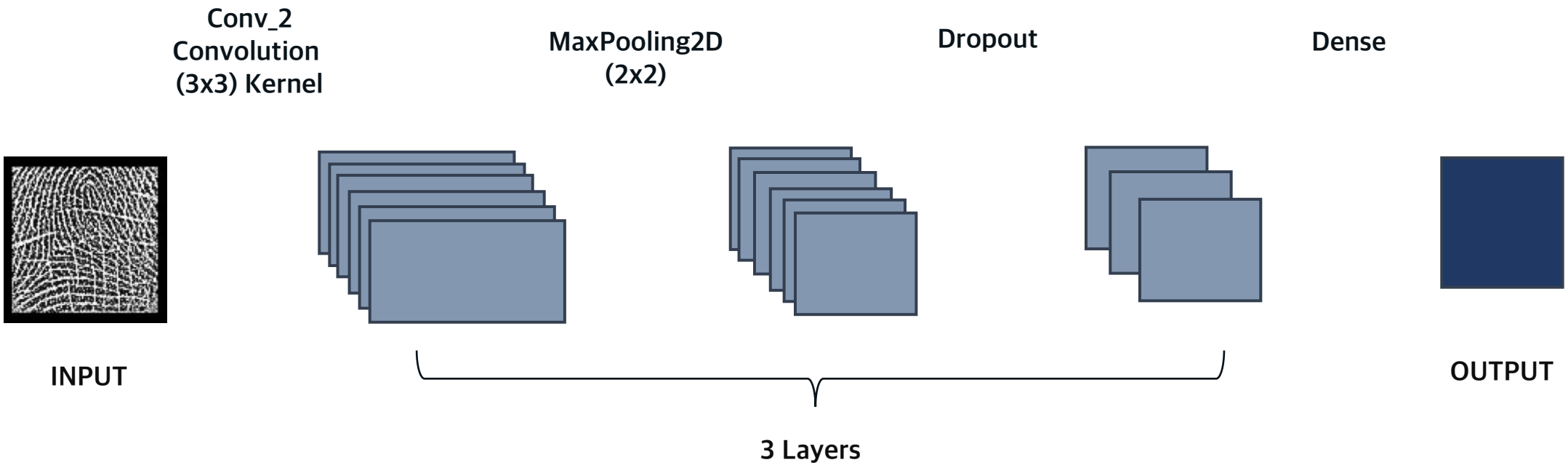
02. 1의 방법처럼 기존 학습 데이터를 불러와 data를 변환하고, 각 데이터에 label을 붙이는 과정에서 30번의 반복을 통해 각 이미지 당 30개의 회전 이미지를 추가로 생성했다.

```
for filename in files:
    category = filename.split(" ")[0]
    image = Image.open('./Finger_Training/'+filename).convert('L')
    image = image.resize((IMAGE_WIDTH, IMAGE_HEIGHT))

    new_image = img_to_array(image)
    samples = expand_dims(new_image, 0)
```

```
it= datagen.flow(samples, batch_size=1)

for i in range(30):
    batch = it.next()
    image = batch[0].astype('uint8')
    np_image = np.array(image)
    data.append(np_image)
```

02. Build Model

02. 모델 설계



3개의 레이어층으로 **CNN 모델**을 구축.

(Convolution2D -> MaxPooling -> Dropout -> Convolution2D -> Dense)

다중분류를 위해 손실함수로 categorical_crossentropy를 사용.

02. Build Model

02. 모델 설계

```
# 모델 형성

def build_model():

    learning_rate = 0.0001
    model = Sequential()
    model.add(Convolution2D(filters = 16, kernel_size = (3, 3), padding='same', activation='relu',
                            input_shape = (IMAGE_WIDTH, IMAGE_HEIGHT, 1)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Convolution2D(filters = 64, kernel_size = (3, 3),  activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Convolution2D(filters = 64, kernel_size = (3, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(8, activation = 'softmax'))

    model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=learning_rate), metrics=['accuracy'])

    return model
```

learning_rate를 0.0001로 설정해 학습을 진행

optimizer로 Adam을 사용

03. Training Data

03. 데이터 훈련

1. 과적합을 막기 위해 10번의 cross validation을 수행한다.

```
StratifiedKFold(n_splits=10) k = 10
for train_index, test_index in skf.split(X, Y):
```

2. 훈련데이터와 테스트데이터를 8:2 비율로 나눠 훈련을 진행한다.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=10)
```

3. 나눠진 X_train과 X_test를 학습 가능하게 reshape 한다.

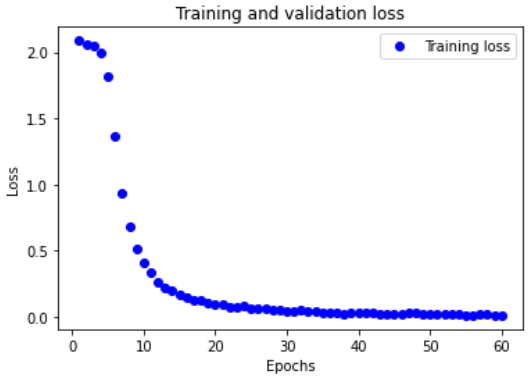
```
X_train = np.reshape(X_train, (len(X_train), IMAGE_WIDTH, IMAGE_HEIGHT, 1))
X_test = np.reshape(X_test, (len(X_test), IMAGE_WIDTH, IMAGE_HEIGHT, 1))
```

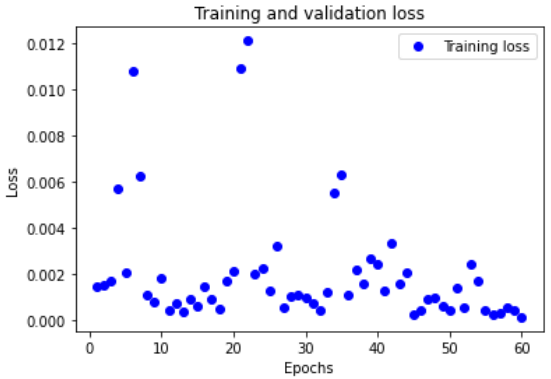
03. Training Data

03. 데이터 훈련

1. history = model.fit(X_train, Y_train, epochs = 60, batch_size = 32, verbose = 1)

처음 Image Augmentation 없이 훈련을 했을 때처럼 epoch = 60, batch_size = 32로 두고 학습을 진행한 결과





위와 같이 학습 중 뛰는 loss 값들이 존재했다. test data의 분류 결과를 확인하니 2와 6번의 지문데이터를 많이 혼동해 batch_size와 learning rate를 조금 변경해 재학습을 진행했다.

➡ 기존 learning rate를 0.001로 사용하다, rate를 감소시켜 0.0001로 학습한 결과 기존보다 학습 결과가 좋지 못했다. 대부분 아예 틀린 결과가 나와 다시 learning rate를 0.001로 변경했다.

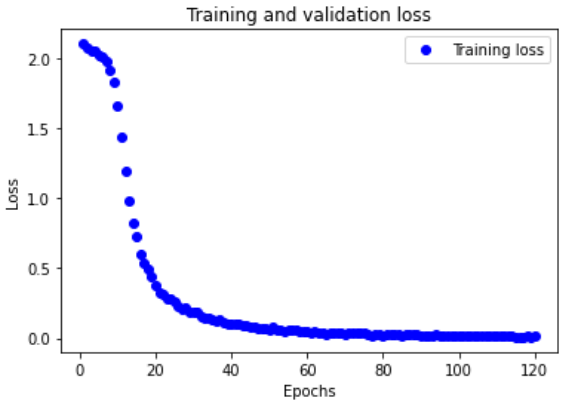
03. Training Data

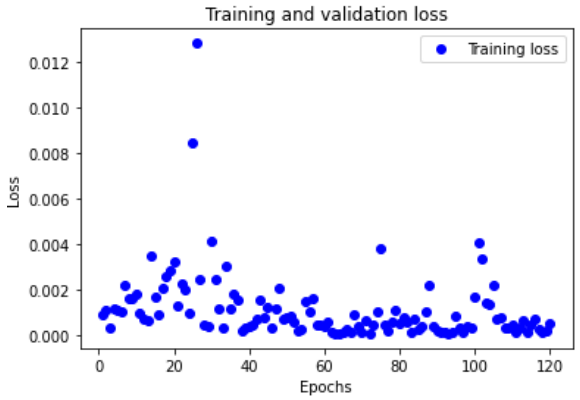
2. history = model.fit(X_train, Y_train, epochs = 60, batch_size = 128, verbose = 1)

batch_size를 128로 변경해 학습하자 혼동해 판단하는 지문 데이터들이 존재했다.

전반적으로 유사한 형태의 지문에서 특징점을 찾아내지 못하는 것 같아 epoch를 증가해 학습횟수를 늘리고, batch_size를 감소시켰다.

3. history = model.fit(X_train, Y_train, epochs = 120, batch_size = 32, verbose = 1)





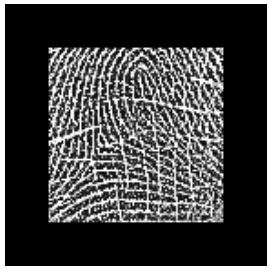

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.009299962781369686
Test accuracy: 0.9979166388511658
```

epoch을 증가해 훈련을 시키자 테스트 데이터의 정확성이 이전보다 높아졌다.

다만, 다른 지문들은 다 잘 구분했으나 지문1과 지문6의 경우 지문의 코어 형태가 유사한

탓인지 1차 학습에 비해 두 지문 판단이 뒤바뀐 결과가 몇 개 발견되었다.



지문1

지문6

03. Training Data

교차 검증별 평균 스코어

```
print(sum(all_acc)/10)
print(sum(all_f1)/10)
print(sum(all_recall)/10)
print(sum(all_precision)/10)
```

Accuracy	0.9883
F1_score	0.9885355
Recall	0.98873825
Precision	0.9887382

최종 스코어

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Test score	0.00929996278
Test accuracy	1.0

최종 데이터 훈련 결과 모델의 스코어는 1차보다 전반적으로 향상되었다.

05. Result

Image	Answer
1	7
2	1
3	2
4	6
5	1
6	2
7	5
8	1
9	5
10	3
11	1
12	2
13	5
14	2
15	1
16	6
17	8
18	5
19	7
20	3
21	1
22	1
23	2
24	8
25	8
26	7

최종적으로 result.csv 파일에서 Image 별로 모델이 분류한 사람의 번호를 확인할 수 있다.

80개의 테스트 데이터를 맞춘 1차 결과와 비교해 보았을 때, Image Augmentation 으로 다양하게 회전된 형태의 지문 인식을 시도해 보았으나 처음에 비해 잘못 인식한 데이터들이 종종 있었다.

찾아본 결과 Fingerprint Thinner이나, 정규화를 통해 이미지에 변환을 가해보면 특징점들을 조금 더 잘 찾을 수 있지 않을까 하는 생각이 든다.