CAPTCHA : 모방 이미지를 이용한 학습시도 총정리

개요 : 앞선 예측 시도를 통해, 어떤 CAPTCHA든 잘 맞추는 범용적인 예측 모델을 얻기는 불가능하거나 매우 어렵다는 사실을 깨달음. 따라서 **특정 사이트의 CAPTCHA에 대해서 높은 정확도를 얻을 수 있는 예측 모델**을 훈련시켜보기로 함.

→ 이때 곧바로 떠오르는 방식은 실제 사용되는 CAPTCHA를 직접 수집 및 레이블링하여 훈련을 시켜보는 것인데, 이는 **가능할지언정 매우 시간이 오래걸리고 고된 작업**이므로 발상을 바꾸어 실제 CAPTCHA를 **모방한 이미지**를 이용해 훈련을 시도해보기로 함.

▼ DC inside 매크로 작성 방지 CAPTCHA

선정 이유 : 2글자 CAPTCHA라서 처음 시도하기에 가장 난이도가 낮아서 적합하다고 생각했음.

시도 과정

1차

2글자 CAPTCHA를 수동 생성하여 학습한 후 별도로 얻은 2글자 CAPTCHA를 해석할 수 있는지 테스트 해봄 → 결과 : 실패(전혀 맞히지 못함)

[훈련 이미지(아래 결과 부분만 사용해도 무방)]

```
[71]: #학습

model = build_model()

epochs = 1000
stopping_patience = 100 # 10번의 epoch동안 개선이 없을 시 종료
#restore_best_weights : 조기종료 후 지금까지 최적값을 냈던 가중치를 다시 선택함
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=stopping_patience, restore_best_weights=True
)

history = model.fit(
    train_dataset,
    validation_data = test_dataset,
    epochs = epochs,
    callbacks=[early_stopping],
)
```

```
Epoch 510/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0178 - val_loss: 1.1328
Epoch 511/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0079 - val_loss: 1.0615
Epoch 512/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0114 - val_loss: 0.9524
Epoch 513/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0128 - val_loss: 1.0420
Epoch 514/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0909 - val_loss: 1.2322
Epoch 515/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0579 - val_loss: 1.1218
Epoch 516/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0267 - val_loss: 0.9678
Epoch 517/1000
73/73 [==============================] - 3s 36ms/step - loss: 0.0091 - val_loss: 1.0082
Epoch 518/1000
73/73 [==============================] - 3s 35ms/step - loss: 0.0161 - val_loss: 0.9283
```

[학습에 사용한 2글자 CAPTCHA]

[실제 대상 CAPTCHA 예시]

[예측 결과] : 일부 글자를 맞히긴 했으나 인식률이 매우 좋지 않았다

```
prediction : ['mx'] / Label : 23
prediction : ['my'] / Label : 3q
prediction : ['fx'] / Label : 46
prediction : ['u8'] / Label : 52
prediction : ['4u'] / Label : 7d
prediction : ['4m'] / Label : 7n
prediction : ['mx'] / Label : 8z
prediction : ['f[UNK]'] / Label : 9b
prediction : ['48'] / Label : a6
prediction : ['f[UNK]'] / Label : am
prediction : ['f8'] / Label : bh
prediction : ['48'] / Label : cv
prediction : ['vx'] / Label : e4
prediction : ['4h'] / Label : mg
prediction : ['mx'] / Label : mq
prediction : ['m8'] / Label : n7
prediction : ['mg'] / Label : nh
prediction : ['rf'] / Label : p5
prediction : ['mx'] / Label : pv
prediction : ['4u'] / Label : qu
prediction : ['ug'] / Label : vg
prediction : ['u8'] / Label : x4
prediction : ['vx'] / Label : xp
prediction : ['48'] / Label : xy
prediction : ['4x'] / Label : yd
prediction : ['u8'] / Label : z4
prediction : ['fx'] / Label : zx
```

실패원인 추정 : CAPTCHA의 스타일이 너무 달랐다. 직접 생성한 CAPTCHA는 라인 노이즈와 함께 심한 왜곡, 전체적인 Pepper&Salt 노이즈가 들어가 있는 반면, 대상이 CAPTCHA는 중앙에 2글자가 배치되고 그 주위로만 Pepper&Salt 노이즈가 배치된 방식이었음. 이러한 차이로 인해 해당 CAPTCHA에 대해 정답을 예측하지 못하는 모델이 훈련된 것이 아닐까 추정하였음.

2차

더 비슷하게 생긴 CAPTCHA를 수동생성하여 훈련시킴. 컬러로 인한 영향을 배제하기 위해 훈련 data와 대상 CAPTCHA 모두 배경색을 흰 색으로, 문자색을 검정색으로 통일하는 전처리를 거침.

[전처리된 CAPTCHA들 간의 비교(위 : 직접 생성, 아래 : 원본)]

[추가적으로 Median Blur처리를 통해 노이즈를 제거한 이미지도 테스트함]

[250 에포크 후 `loss: 0.2000 - val_loss: 1.6690일 때의 결과` **]**

위 : 노이즈 그대로 / 아래 : Median Blur처리를 통해 노이즈를 제거한 이미지

→ 결과 : 두 종류의 이미지를 모두 포함해 단 한 개 맞힘. 다만 몇몇 글자에 대한 인식률이 높아진 것을 체감해 추가적인 전처리를 통해 개선해보기로 함.

```
prediction : 25  /  Actual label : 23
prediction : 23  /  Actual label : 23 정답!
prediction : 34  /  Actual label : 3q
prediction : 24  /  Actual label : 3q
prediction : zv  /  Actual label : 46
prediction : 43  /  Actual label : 46
prediction : e2  /  Actual label : 52
prediction : 83  /  Actual label : 52
prediction : 74  /  Actual label : 7d
prediction : 34  /  Actual label : 7d
prediction : z8  /  Actual label : 7n
prediction : 48  /  Actual label : 7n
prediction : 9g  /  Actual label : 8z
prediction : 95  /  Actual label : 8z
prediction : 8[UNK]  /  Actual label : 9b
prediction : 8[UNK]  /  Actual label : 9b
prediction : 4j  /  Actual label : a6
prediction : 34  /  Actual label : a6
prediction : 56  /  Actual label : am
prediction : 8[UNK]  /  Actual label : am
prediction : 68  /  Actual label : bh
prediction : f9  /  Actual label : bh
prediction : c3  /  Actual label : cv
prediction : e2  /  Actual label : cv
prediction : e3  /  Actual label : e4
prediction : 83  /  Actual label : e4
prediction : 83  /  Actual label : mg
prediction : 87  /  Actual label : mg
prediction : z4  /  Actual label : mq
prediction : 8[UNK]  /  Actual label : mq
prediction : zi  /  Actual label : n7
prediction : v8  /  Actual label : n7
prediction : w[UNK]  /  Actual label : nh
prediction : 8[UNK]  /  Actual label : nh
prediction : 93  /  Actual label : p5
prediction : 98  /  Actual label : p5
prediction : y3  /  Actual label : pv
prediction : 98  /  Actual label : pv
prediction : zc  /  Actual label : qu
prediction : e6  /  Actual label : qu
prediction : 85  /  Actual label : vg
prediction : g7  /  Actual label : vg
prediction : 23  /  Actual label : x4
prediction : 83  /  Actual label : x4
prediction : 3i  /  Actual label : xp
prediction : 8[UNK]  /  Actual label : xp
prediction : 23  /  Actual label : xy
prediction : 58  /  Actual label : xy
prediction : z3  /  Actual label : yd
prediction : 84  /  Actual label : yd
prediction : s3  /  Actual label : z4
prediction : g3  /  Actual label : z4
prediction : gs  /  Actual label : zx
prediction : g5  /  Actual label : zx
```

3차

원본과 보다 유사하게 만들기 위해 두 글자의 높낮이가 랜덤하게 배치되도록 하는 코드를 추가

[글자의 높낮이가 다른 훈련 Data의 예시]

위와 같은 이미지 4881개로 학습한 결과 → 노이즈 O / 노이즈 X 이미지를 모두 포함해 5개 정답

```
prediction : g3  /  Actual label : 23
prediction : 23  /  Actual label : 23 정답!
prediction : 3q  /  Actual label : 3q 정답!
prediction : 34  /  Actual label : 3q
prediction : 8o  /  Actual label : 46
prediction : 9o  /  Actual label : 46
prediction : 53  /  Actual label : 52
prediction : 8g  /  Actual label : 52
prediction : 74  /  Actual label : 7d
prediction : dq  /  Actual label : 7d
prediction : 4m  /  Actual label : 7n
prediction : 4m  /  Actual label : 7n
prediction : u3  /  Actual label : 8z
prediction : be  /  Actual label : 8z
prediction : db  /  Actual label : 9b
prediction : d8  /  Actual label : 9b
prediction : c6  /  Actual label : a6
prediction : 38  /  Actual label : a6
prediction : cp  /  Actual label : am
prediction : 9o  /  Actual label : am
prediction : b9  /  Actual label : bh
prediction : b9  /  Actual label : bh
prediction : o5  /  Actual label : cv
prediction : 6o  /  Actual label : cv
prediction : 94  /  Actual label : e4
prediction : ko  /  Actual label : e4
prediction : m3  /  Actual label : mg
prediction : mg  /  Actual label : mg 정답!
prediction : q[UNK]  /  Actual label : mq
prediction : mq  /  Actual label : mq 정답!
prediction : 45  /  Actual label : n7
prediction : n4  /  Actual label : n7
prediction : m6  /  Actual label : nh
prediction : mo  /  Actual label : nh
prediction : p5  /  Actual label : p5 정답!
prediction : b5  /  Actual label : p5
prediction : bo  /  Actual label : pv
prediction : uq  /  Actual label : pv
prediction : qd  /  Actual label : qu
prediction : dq  /  Actual label : qu
prediction : p3  /  Actual label : vg
prediction : 63  /  Actual label : vg
prediction : 54  /  Actual label : x4
prediction : 9[UNK]  /  Actual label : x4
prediction : 5q  /  Actual label : xp
prediction : uo  /  Actual label : xp
prediction : 56  /  Actual label : xy
prediction : 56  /  Actual label : xy
prediction : s4  /  Actual label : yd
prediction : w4  /  Actual label : yd
prediction : a3  /  Actual label : z4
prediction : wq  /  Actual label : z4
prediction : z5  /  Actual label : zx
prediction : e3  /  Actual label : zx
```

개선방안 : sine wave distortion(글자가 물결치는 모양새가 되도록 하는 왜곡)을 넣는 등 더 정교하게 흉내냄으로써 성능 개선을 시도할 수는 있겠으나 전체적으로 정답률이 낮은 상황을 타파하기는 어려울 것으로 생각함. → 좀 더 모방하기 쉬운 간단한 CAPTCHA를 이용하여 다시 시도해보기로 함.

3차 훈련 당시에 기록한 주피터 노트북은 아래 파일 참고.

DcInside2CharBest.ipynb

▼ NETIS 공유기 로그인 보안 CAPTCHA

선정 이유 : 매우 간단한 형태의 노이즈와 왜곡이 없는 정직한 영문+숫자의 조합이라 맞히기 용이할 것으로 예상함.

[CAPTCHA들 간의 비교(위 : 직접 생성, 아래 : 원본 예시)]

1-1차 시도 : train_loss = 0.257, val_loss = 0.335

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 805ms/step
prediction : 3o0t[UNK]  /  Actual label : 2kh0i
1/1 [==============================] - 0s 57ms/step
prediction : 3ofot  /  Actual label : 3othj
1/1 [==============================] - 0s 14ms/step
prediction : e0oer  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : o9eb[UNK]  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : r89f[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : r83m[UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 16ms/step
prediction : s0otv  /  Actual label : s0otr
```

```
// 텍스트 형태로 저장하기 위한 것이니 필요하지 않다면 무시
1/1 [==============================] - 1s 805ms/step
prediction : 3o0t[UNK]  /  Actual label : 2kh0i
1/1 [==============================] - 0s 57ms/step
prediction : 3ofot  /  Actual label : 3othj
1/1 [==============================] - 0s 14ms/step
prediction : e0oer  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : o9eb[UNK]  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : r89f[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : r83m[UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 16ms/step
prediction : s0otv  /  Actual label : s0otr
```

1-2차 시도 : 같은 모델로 30 epoch 다시 돌려본 결과(새로 학습)

```
Epoch 26/30
282/282 [==============================] - 15s 54ms/step - loss: 0.0170 - val_loss: 0.0016
Epoch 27/30
282/282 [==============================] - 15s 54ms/step - loss: 0.0340 - val_loss: 8.3664e-04
Epoch 28/30
282/282 [==============================] - 15s 54ms/step - loss: 0.0124 - val_loss: 6.4756e-04
Epoch 29/30
282/282 [==============================] - 15s 53ms/step - loss: 0.0071 - val_loss: 4.9702e-04
Epoch 30/30
282/282 [==============================] - 15s 54ms/step - loss: 0.0080 - val_loss: 0.0022
```

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 730ms/step
prediction : 2b0t[UNK]  /  Actual label : 2kh0i
1/1 [==============================] - 0s 14ms/step
prediction : 3otbi  /  Actual label : 3othj
1/1 [==============================] - 0s 15ms/step
prediction : k0os[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : o94eb  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : 89yy[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 15ms/step
prediction : 8h2[UNK][UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 15ms/step
prediction : s0ot[UNK]  /  Actual label : s0otr
```

→ 알 수 없는 이유로 loss가 대폭 개선되고, 인식하는 글자가 달라짐


```
1/1 [==============================] - 1s 730ms/step
prediction : 2b0t[UNK]  /  Actual label : 2kh0i
1/1 [==============================] - 0s 14ms/step
prediction : 3otbi  /  Actual label : 3othj
1/1 [==============================] - 0s 15ms/step
prediction : k0os[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : o94eb  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : 89yy[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 15ms/step
prediction : 8h2[UNK][UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 15ms/step
prediction : s0ot[UNK]  /  Actual label : s0otr
```


이후 동일한 학습 데이터로 5 epoch씩 학습해가며, 학습의 정도가 다른 각각의 모델을 만들어 예측을 시도해보았으나 유의미한 차이는 없었다.

2차 시도: 훈련 데이터를 10000개 추가 생성하여 데이터 총 2만개로 같은 과정을 반복함.

총 25 epoch(`loss: 0.1480 - val_loss: 0.1011`)에서 결과

→ 읽을 수 있는 글자가 달라짐

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 705ms/step
prediction : 6qhpt  /  Actual label : 2kh0i
1/1 [==============================] - 0s 13ms/step
prediction : qotbv  /  Actual label : 3othj
1/1 [==============================] - 0s 13ms/step
prediction : poer[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 14ms/step
prediction : apdvb  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : c8g09  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : r8hqa  /  Actual label : r8h2m
1/1 [==============================] - 0s 14ms/step
prediction : epotz  /  Actual label : s0otr
```


```
1/1 [==============================] - 1s 705ms/step
prediction : 6qhpt  /  Actual label : 2kh0i
1/1 [==============================] - 0s 13ms/step
prediction : qotbv  /  Actual label : 3othj
1/1 [==============================] - 0s 13ms/step
prediction : poer[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 14ms/step
prediction : apdvb  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : c8g09  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : r8hqa  /  Actual label : r8h2m
1/1 [==============================] - 0s 14ms/step
prediction : epotz  /  Actual label : s0otr
```


3차 시도 : 10000개 추가 생성하여 총 데이터 3만개로 같은 과정을 반복 →

결과적으로 그리 큰 개선은 없었음. (`loss: 0.6023 - val_loss: 0.2770`)


```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```


```
1/1 [==============================] - 1s 709ms/step
prediction : 3so0t  /  Actual label : 2kh0i
1/1 [==============================] - 0s 14ms/step
prediction : 3otou  /  Actual label : 3othj
1/1 [==============================] - 0s 16ms/step
prediction : s0osc  /  Actual label : k0osr
1/1 [==============================] - 0s 16ms/step
prediction : o94zo  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : c39t[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : c33o[UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 14ms/step
prediction : s0oto  /  Actual label : s0otr
```

[데이터 3만개 기준 best 결과 (`loss: 0.6023 - val_loss: 0.2770`)]

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

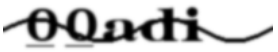
```
1/1 [==============================] - 1s 690ms/step
prediction : 3klti  /  Actual label : 2kh0i
1/1 [==============================] - 0s 17ms/step
prediction : 3otbb  /  Actual label : 3othj
1/1 [==============================] - 0s 15ms/step
prediction : k0osz  /  Actual label : k0osr
1/1 [==============================] - 0s 14ms/step
prediction : o94xb  /  Actual label : n94vb
1/1 [==============================] - 0s 14ms/step
prediction : v89ll  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : v8hto  /  Actual label : r8h2m
1/1 [==============================] - 0s 14ms/step
prediction : s0otv  /  Actual label : s0otr
```

→ 상당히 많은 글자를 맞춘다는 점에서 고무적이었으나, 좀처럼 정답을 내지 못했음.

→ 폰트에 대한 Overfitting 경향이 분명히 있으며, 인간이 보기에도 닮은 글자를 잘 구별해내지 못한다는 점을 깨달았다. r과 v를 구분하지 못하는 모습이 반복적으로 관측됨. → 노이즈를 넣어 과적합을 막으면 해결될 것인가?

4차 시도 : Overfitting을 완화하기 위해 훈련용 이미지에 Gaussian blur효과를 적용해봄

[블러처리된 훈련 데이터 예시]



결과 → 최초로 정답을 맞히는 데 성공함(`loss: 0.7100 - val_loss: 0.3675`)

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 852ms/step
prediction : 29b0t  /  Actual label : 2kh0i
1/1 [==============================] - 0s 17ms/step
prediction : 3otbt  /  Actual label : 3othj
1/1 [==============================] - 0s 22ms/step
prediction : 96ose  /  Actual label : k0osr
1/1 [==============================] - 0s 20ms/step
prediction : o94xb  /  Actual label : n94vb
1/1 [==============================] - 0s 17ms/step
prediction : e806[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 15ms/step
prediction : e86o[UNK]  /  Actual label : r8h2m
1/1 [==============================] - 0s 21ms/step
prediction : s0otr  /  Actual label : s0otr
```

추가적으로 학습시키자, 아쉽게도 다시 정답을 맞히지 못하게 되었으나 새로운 종류의 글자를 인식할 수 있게 되었다.

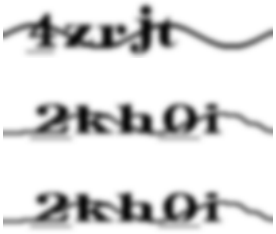
```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 848ms/step
prediction : 29b06  /  Actual label : 2kh0i
1/1 [==============================] - 0s 16ms/step
prediction : 3otbt  /  Actual label : 3othj
1/1 [==============================] - 0s 16ms/step
prediction : g0osr  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : o93xb  /  Actual label : n94vb
1/1 [==============================] - 0s 15ms/step
prediction : e886[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 18ms/step
prediction : e8d2o  /  Actual label : r8h2m
1/1 [==============================] - 0s 15ms/step
prediction : s0our  /  Actual label : s0otr
```

5차 시도 : 4차 시도에서 성능 개선 효과가 있었던 Gaussian blur효과를 더 강하게 적용해 봄

→ 읽을 수 없었던 몇몇 글자를 읽을 수 있게 되었다. 또한 blur_level = 1인 이미지와 blur_level = 2인 이미지에 대한 예측은 항상 동일하게 나왔다.

[강하게 블러처리 된 이미지 예시 (위 : 학습 데이터 / 중간과 아래 : 원본 이미지)] → 원본 중 위는 blur_level = 1 / 아래는 blur_level = 2가 적용된 결과이나 육안으로 보기에 큰 차이는 없음



[위 : 원본 이미지 + blur_level = 1 / 아래 : 원본 이미지 + blur_level = 2]

```
test_images, test_labels = images, labels = load_image_with_label("NETIS_Labled_Gaussian",'png')
```

```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 827ms/step
prediction : 35h6i  /  Actual label : 2kh0i
1/1 [==============================] - 0s 16ms/step
prediction : 3otht  /  Actual label : 3othj
1/1 [==============================] - 0s 16ms/step
prediction : 16er[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : n94zb  /  Actual label : n94vb
1/1 [==============================] - 0s 17ms/step
prediction : v89l[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 15ms/step
prediction : r922o  /  Actual label : r8h2m
1/1 [==============================] - 0s 19ms/step
prediction : z0c6v  /  Actual label : s0otr
```

```
test_images2, test_labels2 = images, labels = load_image_with_label("NETIS_Labled_Gaussian_2x",'png')
```

```
for i in range(len(test_labels2)):
    test_with_sample_image(prediction_model,test_images2[i],test_labels2[i])
```

```
1/1 [==============================] - 0s 18ms/step
prediction : 35h6i  /  Actual label : 2kh0i
1/1 [==============================] - 0s 16ms/step
prediction : 3otht  /  Actual label : 3othj
1/1 [==============================] - 0s 20ms/step
prediction : 16er[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 19ms/step
prediction : n94zb  /  Actual label : n94vb
1/1 [==============================] - 0s 15ms/step
prediction : v89l[UNK]  /  Actual label : r89ll
1/1 [==============================] - 0s 16ms/step
prediction : r922o  /  Actual label : r8h2m
1/1 [==============================] - 0s 15ms/step
prediction : z0c6v  /  Actual label : s0otr
```

추가학습 결과 (`loss: 0.2016 - val_loss: 0.0457`)에서 최고 결과 도출

→ 정답은 하나도 없지만 가장 많은 글자를 맞히는데 성공함

[98]:
```
for i in range(len(test_labels)):
    test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 864ms/step
prediction : 2gh0i  /  Actual label : 2kh0i
1/1 [==============================] - 0s 16ms/step
prediction : 3oth6  /  Actual label : 3othj
1/1 [==============================] - 0s 16ms/step
prediction : k0cs[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 16ms/step
prediction : r94zb  /  Actual label : n94vb
1/1 [==============================] - 0s 18ms/step
prediction : v89l1  /  Actual label : r89ll
1/1 [==============================] - 0s 14ms/step
prediction : v8h2o  /  Actual label : r8h2m
1/1 [==============================] - 0s 16ms/step
prediction : z0otr  /  Actual label : s0otr
```

추가학습 결과 → 가장 읽히지 않았던 2kh0i를 읽는데 성공함

```
for i in range(len(test_labels2)):
    test_with_sample_image(prediction_model,test_images2[i],test_labels2[i])
```

```
1/1 [==============================] - 0s 20ms/step
prediction : 2kh0i  /  Actual label : 2kh0i
1/1 [==============================] - 0s 15ms/step
prediction : 3othi  /  Actual label : 3othj
1/1 [==============================] - 0s 17ms/step
prediction : kd9cz  /  Actual label : k0osr
1/1 [==============================] - 0s 15ms/step
prediction : n94zb  /  Actual label : n94vb
1/1 [==============================] - 0s 18ms/step
prediction : v4898  /  Actual label : r89ll
1/1 [==============================] - 0s 16ms/step
prediction : r8h2o  /  Actual label : r8h2m
1/1 [==============================] - 0s 16ms/step
prediction : z0otv  /  Actual label : s0otr
```

추가학습 결과 (train_loss / val_loss는 유사한데 계속해서 결과가 달라짐. 새로운 정답을 맞.)

```
[120]: for i in range(len(test_labels)):
           test_with_sample_image(prediction_model,test_images[i],test_labels[i])
```

```
1/1 [==============================] - 1s 820ms/step
prediction : 2kh05  /  Actual label : 2kh0i
1/1 [==============================] - 0s 17ms/step
prediction : 3oth5  /  Actual label : 3othj
1/1 [==============================] - 0s 15ms/step
prediction : kooa[UNK]  /  Actual label : k0osr
1/1 [==============================] - 0s 16ms/step
prediction : n94zb  /  Actual label : n94vb
1/1 [==============================] - 0s 17ms/step
prediction : r89ll  /  Actual label : r89ll
1/1 [==============================] - 0s 16ms/step
prediction : r8h2o  /  Actual label : r8h2m
1/1 [==============================] - 0s 15ms/step
prediction : z0otr  /  Actual label : s0otr
```

결과 : 비록 높은 정답율을 기록하는 예측 모델을 만드는 데에는 실패했으나, 모방한 이미지를 통한 학습이 가능하다는 확신을 얻을 수 있었음. 보다 모방하기 쉽거나, 소문자 z / r / v와 같이 모델이 자주 혼동하는 글자가 배제된 CAPTCHA라면 충분히 좋은 결과를 낼 수 있다고 생각함.

▼ Melon Ticket 매크로 방지 CAPTCHA

선정 이유 : 알파벳 대문자만으로 조합 + 별다른 왜곡이 없는 간단한 노이즈이기 때문에 가장 접근하기 쉬울 것을 예상함.

[Melon Ticket에서 좌석 예매 시도시 뜨는 실제 화면]

[CAPTCHA들 간의 비교(위 : 직접 생성, 아래 : 원본 예시)]

→ 동일한 폰트, 동일한 노이즈(Java로 작성된 오픈소스 라이브러리 중 Simple-Captcha 사용)

→ 육안으로 구별할 수 있는 차이가 거의 없음.

→ 원본 CAPTCHA를 관찰해본 결과 글자 간 간격을 랜덤하게 좁히는 식으로 동작하는 것 같아서 해당 메커니즘을 구현하는 코드를 포함함

▼ [참고 : 생성에 사용한 코드 / 발표엔 넣지 말아주세요! 자료 제출시에만 참고할 예정]

```
from PIL import Image, ImageDraw, ImageFont
import random

def createCaptcha(rand_text):
    width, height = 280, 80
    background_color = "white"

    image = Image.new("RGB", (width, height), background_color)
    draw = ImageDraw.Draw(image)

    text = rand_text
    text_color = "black"
    font_path = r"C:\Users\yongh\Downloads\prima-sans-oblique_fxtCr\Prima Sans Oblique\Prima Sans Oblique.otf"

    font_size = 54
    font = ImageFont.truetype(font_path, font_size)

    start_x = 20 + 5 - 4
    start_y = 40 - 54/2 -10
```

```
    spacing = random.randint(-3,1)

    for char in text:
        draw.text((start_x, start_y), char, fill=text_color, font=font)
        char_width = draw.textlength(char, font=font)
        start_x += char_width + spacing

    image.save(r"C:\Users\yongh\Desktop\MLTeamProject\resource\MelonCaptcha_original\%s.png" %(text))

char_list = list(''.join("ABCDEFGHIJKLMNOPQRSTUVWXYZ"))

for i in range(10000):
    random_string = ''.join(random.choices(char_list, k=6))
    createCaptcha(random_string)
```

```
// 이하 Java 7이하에서만 정상 작동

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;

public class Main {
    private static final Logger logger = Logger.getLogger(Main.class.getName());

    public static void makeNoise(String label){
        String imagePath = String.format("C:/Users/yongh/Desktop/MLTeamProject/resource/MelonCaptcha_original/%s.png",label);
        String outputPath = String.format("C:/Users/yongh/Desktop/MLTeamProject/resource/MelonCaptcha_noised/%s.png",label);
        File inputFile = new File(imagePath);
        BufferedImage image = null;
        try {
            image = ImageIO.read(inputFile);

            int width = image.getWidth();
            int height = image.getHeight();

            CaptchaGenerator generator = new CaptchaGenerator();
            generator.makeNoise(image);

            try {
                File output = new File(outputPath);
                ImageIO.write(image, "png", output);
            } catch (IOException e) {
                logger.severe("Failed to read or process the image: " + e.getMessage());
            }

        } catch (IOException e) {
            logger.severe("Failed to read or process the image: " + e.getMessage());
            // Additionally, handle the exception or perform error recovery here
        }
    }

    public static void main(String[] args) {
        String folderPath = "C:/Users/yongh/Desktop/MLTeamProject/resource/MelonCaptcha_original";
        String extension = "png";

        List<String> images = loadImagesWithLabel(folderPath, extension);
        for (String image : images) {
            makeNoise(image);
        }
    }

    public static List<String> loadImagesWithLabel(String folderPath, String extension) {
        File folder = new File(folderPath);
        File[] listOfFiles = folder.listFiles();
        List<String> labels = new ArrayList<>();

        if (listOfFiles != null) {
            for (File file : listOfFiles) {
                if (file.isFile() && file.getName().endsWith("." + extension)) {
                    String imageName = file.getName();
                    String label = imageName.split("\\.")[0].split("_")[0];
                    labels.add(label);
                }
            }
        }
        return labels;
```

```
    }
}
```

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.CubicCurve2D;
import java.awt.geom.PathIterator;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;
import java.security.SecureRandom;
import java.util.Random;

public class CaptchaGenerator{
    private static final Random RAND = new SecureRandom();

    private final Color _color;

    private final float _width;

    public CaptchaGenerator() {
        this(Color.BLACK, 3.0F);
    }

    public CaptchaGenerator(Color color, float width) {
        this._color = color;
        this._width = width;
    }

    public void makeNoise(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        CubicCurve2D cc = new CubicCurve2D.Float(width * 0.1F, height *
                RAND.nextFloat(), width * 0.1F, height *
                RAND.nextFloat(), width * 0.25F, height *
                RAND.nextFloat(), width * 0.9F, height *
                RAND.nextFloat());
        PathIterator pi = cc.getPathIterator(null, 2.0D);
        Point2D[] tmp = new Point2D[200];
        int i = 0;
        while (!pi.isDone()) {
            float[] coords = new float[6];
            switch (pi.currentSegment(coords)) {
                case 0:
                case 1:
                    tmp[i] = new Point2D.Float(coords[0], coords[1]);
                    break;
            }
            i++;
            pi.next();
        }
        Point2D[] pts = new Point2D[i];
        System.arraycopy(tmp, 0, pts, 0, i);
        Graphics2D graph = (Graphics2D)image.getGraphics();
        graph.setRenderingHints(new RenderingHints(
                RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON));
        graph.setColor(this._color);
        for (i = 0; i < pts.length - 1; i++) {
            if (i < 3)
                graph.setStroke(new BasicStroke(this._width));
            graph.drawLine((int)pts[i].getX(), (int)pts[i].getY(),
                    (int)pts[i + 1].getX(), (int)pts[i + 1].getY());
        }
        graph.dispose();
    }
}
```

1차 시도 : 10000개의 훈련 데이터를 생성하여 훈련함

→ 20개 중 4개를 맞혔으나 기대한 바에 미치지 못해 많이 아쉬웠음. 최대한 원본과 같아지도록 추가적인 이미지 전처리를 시도하였으나, 개선되는 점은 없었음.

```
1/1 [==============================] - 1s 697ms/step
prediction : YOCU[UNK][UNK]  /  Actual label : BDYDCU  /  False
```

```
1/1 [==============================] - 0s 15ms/step
prediction : BMCOP[UNK]  /  Actual label : BMZOAP  /  False
1/1 [==============================] - 0s 14ms/step
prediction : CGCCPU  /  Actual label : CZZPUT  /  False
1/1 [==============================] - 0s 15ms/step
prediction : CIST[UNK][UNK]  /  Actual label : DCPTBT  /  False
1/1 [==============================] - 0s 14ms/step
prediction : GCVOI[UNK]  /  Actual label : GEVOTF  /  False
1/1 [==============================] - 0s 15ms/step
prediction : IMMWW[UNK]  /  Actual label : IMMWFW  /  False
1/1 [==============================] - 0s 15ms/step
prediction : JNKBMT  /  Actual label : JNKBMT  /  True
1/1 [==============================] - 0s 16ms/step
prediction : JROK[UNK][UNK]  /  Actual label : JRLOKF  /  False
1/1 [==============================] - 0s 15ms/step
prediction : IKIBU[UNK]  /  Actual label : LKATBU  /  False
1/1 [==============================] - 0s 14ms/step
prediction : MYGSTZ  /  Actual label : MYGSTZ  /  True
1/1 [==============================] - 0s 15ms/step
prediction : OGNMUI  /  Actual label : OGNMUT  /  False
1/1 [==============================] - 0s 15ms/step
prediction : XO[UNK][UNK][UNK][UNK]  /  Actual label : PEXOAF  /  False
1/1 [==============================] - 0s 16ms/step
prediction : QJXZYU  /  Actual label : QJXZYU  /  True
1/1 [==============================] - 0s 15ms/step
prediction : QIOGCW  /  Actual label : QTOZWG  /  False
1/1 [==============================] - 0s 14ms/step
prediction : RQXBCY  /  Actual label : RQXBZY  /  False
1/1 [==============================] - 0s 15ms/step
prediction : UPX[UNK][UNK][UNK]  /  Actual label : UAPLFX  /  False
1/1 [==============================] - 0s 14ms/step
prediction : VDBKIK  /  Actual label : VDBKIK  /  True
1/1 [==============================] - 0s 14ms/step
prediction : XCVWMI  /  Actual label : XCVWMT  /  False
1/1 [==============================] - 0s 15ms/step
prediction : XJZOV[UNK]  /  Actual label : XJZAOV  /  False
1/1 [==============================] - 0s 15ms/step
prediction : XITMII  /  Actual label : XYMHYI  /  False
```

당시 사용한 주피터 노트북(제출용, 참고용)

Melon_Before_GaussianBlur.ipynb

2차 시도 : 정답을 맞추긴 하는 것으로 보아 특정 글자에 대한 Overfitting이 있다고 생각하여 이전 시도 때 효과를 봤던 Gaussian Blur처리를 가해 학습을 시도함.

→ 그리고 마침내 높은 정답율을 기록하는 데에 성공! (17/20 = 85%)

```
1/1 [==============================] - 1s 779ms/step
prediction : BDYDCU  /  Actual label : BDYDCU  /  True
1/1 [==============================] - 0s 18ms/step
prediction : BMZOAP  /  Actual label : BMZOAP  /  True
1/1 [==============================] - 0s 16ms/step
prediction : CZZPUT  /  Actual label : CZZPUT  /  True
1/1 [==============================] - 0s 18ms/step
prediction : DCPTBT  /  Actual label : DCPTBT  /  True
1/1 [==============================] - 0s 16ms/step
prediction : GEVOTF  /  Actual label : GEVOTF  /  True
1/1 [==============================] - 0s 16ms/step
prediction : IMMWFW  /  Actual label : IMMWFW  /  True
1/1 [==============================] - 0s 17ms/step
prediction : JNKBMT  /  Actual label : JNKBMT  /  True
1/1 [==============================] - 0s 17ms/step
prediction : JRLOKE  /  Actual label : JRLOKF  /  False
1/1 [==============================] - 0s 16ms/step
prediction : LKATBU  /  Actual label : LKATBU  /  True
1/1 [==============================] - 0s 16ms/step
prediction : MYGSTZ  /  Actual label : MYGSTZ  /  True
1/1 [==============================] - 0s 16ms/step
prediction : OGNMUT  /  Actual label : OGNMUT  /  True
1/1 [==============================] - 0s 17ms/step
prediction : PEXOAE  /  Actual label : PEXOAF  /  False
1/1 [==============================] - 0s 16ms/step
prediction : QJXZYU  /  Actual label : QJXZYU  /  True
```

```
1/1 [==============================] - 0s 16ms/step
prediction : QTOZWG  /  Actual label : QTOZWG  /  True
1/1 [==============================] - 0s 17ms/step
prediction : RQXBZY  /  Actual label : RQXBZY  /  True
1/1 [==============================] - 0s 16ms/step
prediction : UAPLFX  /  Actual label : UAPLFX  /  True
1/1 [==============================] - 0s 16ms/step
prediction : VDBKIK  /  Actual label : VDBKIK  /  True
1/1 [==============================] - 0s 16ms/step
prediction : XCVWMT  /  Actual label : XCVWMT  /  True
1/1 [==============================] - 0s 16ms/step
prediction : XJZAOV  /  Actual label : XJZAOV  /  True
1/1 [==============================] - 0s 17ms/step
prediction : XTMHYI  /  Actual label : XYMHYI  /  False
```

당시 사용한 주피터 노트북(제출용, 참고용)

Melon_After_GaussianBlur.ipynb

→ 새로 훈련시킬 때매다 정답율이 달라지나 대체로 60~85%사이를 기록하는 것을 확인