Artificial intelligence engineering

6팀
반려견 행동 분석 모델 학습

강영미 송주호 양유석 이유진 이한규 현기호

인공지능공학

기말 팀프로젝트

Dog Behavior Analysis Dataset

Contents

목차

Dog Behavior Analysis Dataset

EDA

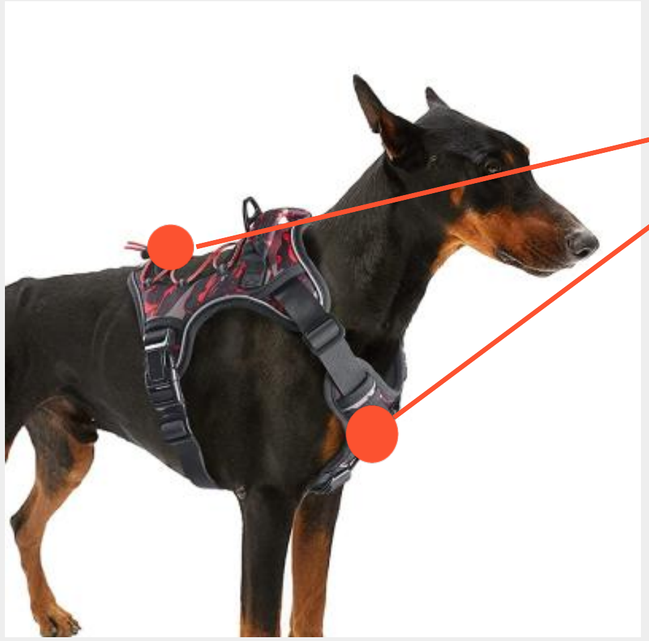
Exploratory Data Analysis

인공지능공학

Exploratory Data Analysis

01. 데이터 설명

0.01초마다 측정



Dog : 3축 가속도계 및 3축 자이로스코프 센서를 포함하는 2개의 활동센서 착용 중

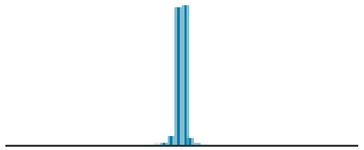
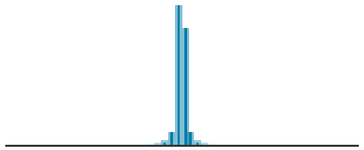
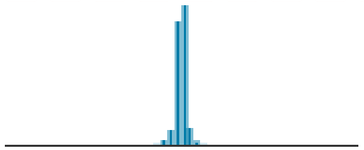
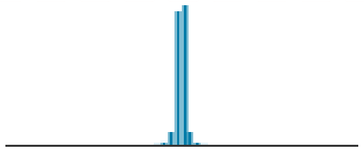
20개의 동적 동작

```
['<undefined>' 'Bowing' 'Carrying object' 'Drinking' 'Eating'
 'Extra_Synchronization' 'Galloping' 'Jumping' 'Lying chest' 'Pacing'
 'Panting' 'Playing' 'Shaking' 'Sitting' 'Sniffing' 'Standing'
 'Synchronization' 'Trotting' 'Tugging' 'Walking']
```

Exploratory Data Analysis

02. 자이로스코프센서

중력을 기준으로 얼마나 기울어져있는지의 값 측정

# GBack_x	# GBack_y	# GBack_z	# GNeck_x
 -2k 2천	 -2k 2천	 -2k 2천	 -2k 2천
-17.639161	-22.766115	7.44629	-7.934571
-15.075685	-11.413575	4.821778	-3.90625
-12.207032	-0.12207	2.807617	-0.488281
-9.46045	7.995606	1.586914	1.159668
-8.361817	14.587403	-1.037598	4.577637
-9.826661	19.653322	-5.493164	12.939454
-11.657716	15.869142	-7.751465	22.70508

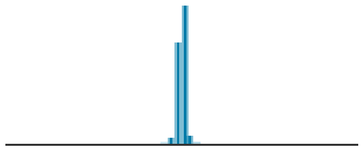
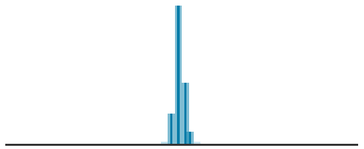
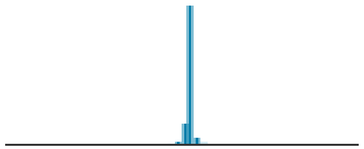
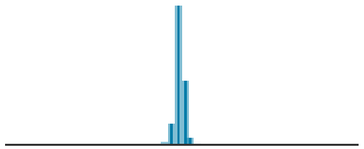
한쪽 방향으로 쏠렸을 때, 기울어지는 각도를 측정

가속도센서로 측정할 수 없는 방위각과 모든 축에 대한 회전각 측정

Exploratory Data Analysis

03. 가속도계

**움직이는 방향,
가속도 측정**

# ABack_x	# ABack_y	# ABack_z	# Aneck_x
			
-16 16	-16 16	-16 16	-16 16
0.041504	0.938965	-0.015137	-0.067871
0.041992	0.941895	-0.02002	-0.128906
0.040527	0.939453	-0.004395	-0.158691
0.021484	0.946289	0.007813	-0.12207
-0.000977	0.951172	0.033691	-0.053711
-0.006348	0.955078	0.038574	0.019043
0.001953	0.959473	0.035156	0.045898

**Dog의 어떤부위가
어떤 방향으로 움직이는지
측정**

**x, y, z 좌표로 센서값을
받으므로 하나의 벡터값으로 계산**

Exploratory Data Analysis

04. 행동

Behavior_1		Behavior_2		Behavior_3		PointEvent	
<undefined>	38%	<undefined>	72%	<undefined>	98%	<undefined>	100%
Lying chest	10%	Panting	9%	Galloping	1%	Bark	0%
Other (5542568)	52%	Other (2035805)	19%	Other (113099)	1%		
<undefined>		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	

Dog의 1초이상 실행한 행동을 비디오 녹화를 통해 발견함

(동시에 여러 동작 가능)

Dog의 행동은 최대 3가지까지 동시에 가능,
짖는 행동 따로 표시

Exploratory Data Analysis

05. 행동 분류

Behavior_1		Behavior_2		Behavior_3		PointEvent	
<undefined>	38%	<undefined>	72%	<undefined>	98%	<undefined>	100%
Lying chest	10%	Panting	9%	Galloping	1%	Bark	0%
Other (5542568)	52%	Other (2035805)	19%	Other (113099)	1%		
<undefined>		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	
Synchronization		<undefined>		<undefined>		<undefined>	

undefined 상태가 가장 적게 나타나는 Behavior_1이 학습에 적합하다고 판단

Exploratory Data Analysis

06. 예상 적합 모델

	DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	ANeck_x	ANeck_y	ANeck_z	GBack_x	GBack_y	GBack_z	GNeck_x	GNeck_y	GNeck_z	Task	Behavior_1
0	16	1	0.00	0.041504	0.938965	-0.015137	-0.067871	-0.510254	-0.934570	-17.639161	-22.766115	7.446290	-7.934571	6.347657	13.427735	<undefined>	<undefined>
1	16	1	0.01	0.041992	0.941895	-0.020020	-0.128906	-0.494141	-0.913086	-15.075685	-11.413575	4.821778	-3.906250	4.394532	16.540528	<undefined>	Synchronization
2	16	1	0.02	0.040527	0.939453	-0.004395	-0.158691	-0.480469	-0.911133	-12.207032	-0.122070	2.807617	-0.488281	-1.953125	26.794435	<undefined>	Synchronization
3	16	1	0.03	0.021484	0.946289	0.007813	-0.122070	-0.486816	-0.880371	-9.460450	7.995606	1.586914	1.159668	-5.676270	38.085940	<undefined>	Synchronization
4	16	1	0.04	-0.000977	0.951172	0.033691	-0.053711	-0.500000	-0.807129	-8.361817	14.587403	-1.037598	4.577637	4.089356	41.503909	<undefined>	Synchronization
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10611063	74	2	1928.19	0.018066	0.436523	0.808105	-0.620605	0.326172	-0.647949	23.620607	75.317387	77.148442	-124.511726	107.727057	-122.497566	Task treat-search	Sniffing
10611064	74	2	1928.20	-0.021973	0.515625	0.813965	-0.671875	0.241211	-0.660645	34.729006	63.293461	79.956060	-116.455085	97.534186	-123.229988	Task treat-search	Sniffing
10611065	74	2	1928.21	-0.041504	0.507324	0.782227	-0.674805	0.244629	-0.627930	39.978030	49.316409	84.655767	-104.858405	95.458990	-124.816902	Task treat-search	Sniffing
10611066	74	2	1928.22	-0.032715	0.484375	0.824219	-0.685059	0.211426	-0.578613	38.024905	32.775881	89.538580	-87.463384	101.257330	-128.601082	Task treat-search	Sniffing
10611067	74	2	1928.23	0.033691	0.432617	0.785645	-0.664063	0.254395	-0.527344	26.306154	19.897462	89.477545	-64.025883	93.688971	-128.906258	Task treat-search	Sniffing

10611068 rows × 20 columns

0.01초 단위로 측정된 시계열 데이터이기 때문에 순환신경망 기법의 하나인 LSTM 모델이 적합할 것이라고 판단함

Dog Behavior Analysis Dataset

데이터 전처리

Data Preprocessing

인공지능공학

Data Preprocessing

01. 데이터 로드

	DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	ANeck_x	ANeck_y	ANeck_z	GBack_x	GBack_y	GBack_z	GNeck_x	GNeck_y	GNeck_z	Task	Behavior_1
0	16	1	0.00	0.041504	0.938965	-0.015137	-0.067871	-0.510254	-0.934570	-17.639161	-22.766115	7.446290	-7.934571	6.347657	13.427735	<undefined>	<undefined>
1	16	1	0.01	0.041992	0.941895	-0.020020	-0.128906	-0.494141	-0.913086	-15.075685	-11.413575	4.821778	-3.906250	4.394532	16.540528	<undefined>	Synchronization
2	16	1	0.02	0.040527	0.939453	-0.004395	-0.158691	-0.480469	-0.911133	-12.207032	-0.122070	2.807617	-0.488281	-1.953125	26.794435	<undefined>	Synchronization
3	16	1	0.03	0.021484	0.946289	0.007813	-0.122070	-0.486816	-0.880371	-9.460450	7.995606	1.586914	1.159668	-5.676270	38.085940	<undefined>	Synchronization
4	16	1	0.04	-0.000977	0.951172	0.033691	-0.053711	-0.500000	-0.807129	-8.361817	14.587403	-1.037598	4.577637	4.089356	41.503909	<undefined>	Synchronization
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

01

각 행의 센서 데이터를 x로 설정

02

각 행의 행동1을 y로 설정

```
X = data[['ABack_x','ABack_y','ABack_z','ANeck_x', 'ANeck_y', 'ANeck_z',
          'GBack_x', 'GBack_y', 'GBack_z', 'GNeck_x', 'GNeck_y', 'GNeck_z']].to_numpy()
Y = data[['Behavior_1']].values.ravel()
```

모든 센서 측정값 colunm을 데이터로 사용하도록 설정

Data Preprocessing

02. 정답 데이터 수치화

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(Y)
Y_encoder = encoder.transform(Y)
Y_encoder
```

01 sklearn에서 제공하는 LabelEncoder를 활용하여 카테고리형 데이터를 수치형으로 변환

```
# 훈련 집합과 테스트 집합으로 분할
x_train, x_test, y_train, y_test = train_test_split(X, Y_encoder, train_size=0.6, random_state=34)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

02 sklearn에서 제공하는 train_test_split을 활용하여 학습셋과 테스트셋 분리

Dog Behavior Analysis Dataset

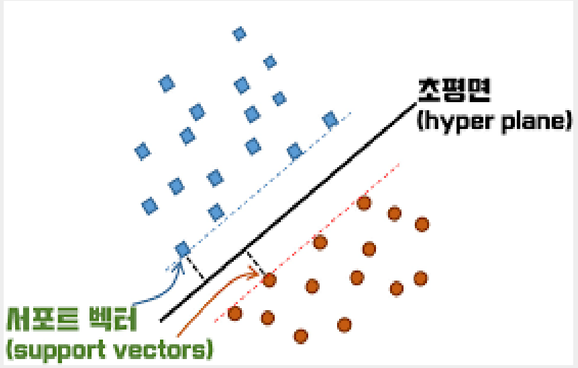
인공지능 모델링

AI Modeling

인공지능공학

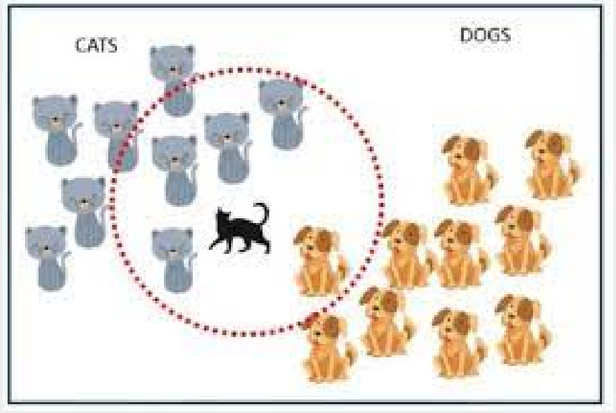
Data Preprocessing

모델

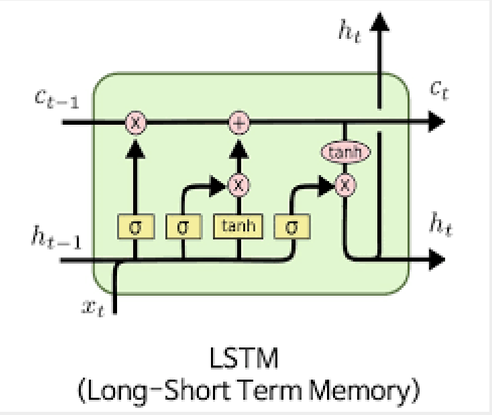


SVM 알고리즘

Ramdom Forest



KNN 알고리즘



LSTM 알고리즘

Dog Behavior Analysis Dataset

SVM

Support Vector Machine

인공지능공학

Support Vector Machine

01. SVM모델 학습

```
1 from sklearn.svm import SVC
2
3 s = SVC(gamma=0.1,C=10)
4 s.fit(train_x,train_y)
```

01 SVM 모델 임포트

02 SVM 모델 학습
gamma = 0.1
C=10

BUT!

입력 데이터셋이 너무 많은 관계로 러닝 타임이 너무 길어졌고
12시간 돌아간후 구글 코렙의 런타임이 중지되어 제대로 학습을 하지 못하였음.

Dog Behavior Analysis Dataset

랜덤 포레스트

RandomForestClassifier

인공지능공학

RandomForestClassifier

01. 랜덤 포레스트 모델 선정 이유

랜덤 포레스트란? RandomForest는 앙상블 학습 방법 일종으로, 훈련 과정에서 구성한 다수의 결정 트리로부터 분류 또는 평균 예측치 (회귀분석)를 출력하여 동작합니다.

	DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	ANeck_x	ANeck_y	ANeck_z	GBack_x	GBack_y	GBack_z	GNeck_x	GNeck_y	GNeck_z	Task	Behavior_1
0	16	1	0.00	0.041504	0.938965	-0.015137	-0.067871	-0.510254	-0.934570	-17.639161	-22.766115	7.446290	-7.934571	6.347657	13.427735	<undefined>	<undefined>
1	16	1	0.01	0.041992	0.941895	-0.020020	-0.128906	-0.494141	-0.913086	-15.075685	-11.413575	4.821778	-3.906250	4.394532	16.540528	<undefined>	Synchronization
2	16	1	0.02	0.040527	0.939453	-0.004395	-0.158691	-0.480469	-0.911133	-12.207032	-0.122070	2.807617	-0.488281	-1.953125	26.794435	<undefined>	Synchronization
3	16	1	0.03	0.021484	0.946289	0.007813	-0.122070	-0.486816	-0.880371	-9.460450	7.995606	1.586914	1.159668	-5.676270	38.085940	<undefined>	Synchronization
4	16	1	0.04	-0.000977	0.951172	0.033691	-0.053711	-0.500000	-0.807129	-8.361817	14.587403	-1.037598	4.577637	4.089356	41.503909	<undefined>	Synchronization
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10611063	74	2	1928.19	0.018066	0.436523	0.808105	-0.620605	0.326172	-0.647949	23.620607	75.317387	77.148442	-124.511726	107.727057	-122.497566	Task treat-search	Sniffing
10611064	74	2	1928.20	-0.021973	0.515625	0.813965	-0.671875	0.241211	-0.660645	34.729006	63.293461	79.956060	-116.455085	97.534186	-123.229988	Task treat-search	Sniffing
10611065	74	2	1928.21	-0.041504	0.507324	0.782227	-0.674805	0.244629	-0.627930	39.978030	49.316409	84.655767	-104.858405	95.458990	-124.816902	Task treat-search	Sniffing
10611066	74	2	1928.22	-0.032715	0.484375	0.824219	-0.685059	0.211426	-0.578613	38.024905	32.775881	89.538580	-87.463384	101.257330	-128.601082	Task treat-search	Sniffing
10611067	74	2	1928.23	0.033691	0.432617	0.785645	-0.664063	0.254395	-0.527344	26.306154	19.897462	89.477545	-64.025883	93.688971	-128.906258	Task treat-search	Sniffing

각 센서 데이터 값마다 특정한 움직임을가질 것이라 예측하여 트리로 분류하는 랜덤 포레스트 모델을 사용함

RandomForestClassifier

02. 랜덤 포레스트 모델 학습

```
from sklearn.metrics import classification_report

rfc = RandomForestClassifier(n_estimators=10, n_jobs=-1)
model = rfc.fit(x_train,y_train)
y_pred = rfc.predict(x_test)

print(classification_report(y_test,y_pred))
```

```
(n_estimators=10, n_jobs=-1)
(n_estimators=20, n_jobs=-1)
(n_estimators=30, n_jobs=-1)
(n_estimators=40, n_jobs=-1)
(n_estimators=50, n_jobs=-1)
```

sklearn에서 제공하는 RandomForestClassifier를 트리 수를 10개씩 늘려 설정하고 모델 생성

RandomForestClassifier

03. 랜덤 포레스트 학습 결과

	precision	recall	f1-score	support
<undefined>	0.69	0.84	0.76	1211693
Bowing	0.94	0.22	0.35	153
Carrying object	0.87	0.05	0.09	5318
Drinking	0.88	0.68	0.77	19510
Eating	0.78	0.29	0.42	50226
Extra_Synchronization	0.90	0.23	0.37	78
Galloping	0.74	0.04	0.08	3179
Jumping	0.91	0.03	0.05	1176
Lying chest	0.95	0.91	0.93	309890
Pacing	0.68	0.14	0.23	23046
Panting	0.91	0.83	0.87	250855
Playing	0.68	0.55	0.61	258716
Shaking	0.84	0.72	0.77	12246
Sitting	0.94	0.88	0.91	152750
Sniffing	0.79	0.87	0.83	307204
Standing	0.95	0.77	0.85	134384
Synchronization	0.95	0.56	0.71	4949
Trotting	0.75	0.72	0.73	215595
Tugging	0.90	0.01	0.01	4150
Walking	0.67	0.40	0.50	218203
accuracy			0.77	3183321
macro avg	0.84	0.49	0.54	3183321
weighted avg	0.77	0.77	0.76	3183321

(n_estimators=10, n_jobs=-1)

	precision	recall	f1-score	support
<undefined>	0.71	0.85	0.77	1211693
Bowing	1.00	0.17	0.29	153
Carrying object	0.97	0.04	0.08	5318
Drinking	0.91	0.70	0.79	19510
Eating	0.86	0.29	0.44	50226
Extra_Synchronization	0.94	0.21	0.34	78
Galloping	0.93	0.03	0.06	3179
Jumping	0.97	0.03	0.05	1176
Lying chest	0.96	0.92	0.94	309890
Pacing	0.81	0.13	0.23	23046
Panting	0.92	0.85	0.89	250855
Playing	0.71	0.57	0.63	258716
Shaking	0.84	0.74	0.79	12246
Sitting	0.95	0.90	0.93	152750
Sniffing	0.79	0.91	0.84	307204
Standing	0.97	0.79	0.87	134384
Synchronization	0.97	0.59	0.73	4949
Trotting	0.76	0.75	0.75	215595
Tugging	1.00	0.01	0.01	4150
Walking	0.73	0.42	0.53	218203
accuracy			0.78	3183321
macro avg	0.88	0.50	0.55	3183321
weighted avg	0.79	0.78	0.78	3183321

(n_estimators=20, n_jobs=-1)

	precision	recall	f1-score	support
<undefined>	0.72	0.85	0.78	1211693
Bowing	0.97	0.19	0.32	153
Carrying object	0.99	0.04	0.08	5318
Drinking	0.91	0.70	0.79	19510
Eating	0.87	0.29	0.44	50226
Extra_Synchronization	1.00	0.26	0.41	78
Galloping	0.91	0.03	0.07	3179
Jumping	0.96	0.02	0.04	1176
Lying chest	0.96	0.92	0.94	309890
Pacing	0.86	0.13	0.22	23046
Panting	0.93	0.86	0.89	250855
Playing	0.72	0.58	0.64	258716
Shaking	0.84	0.76	0.80	12246
Sitting	0.96	0.91	0.93	152750
Sniffing	0.79	0.91	0.85	307204
Standing	0.97	0.79	0.87	134384
Synchronization	0.97	0.60	0.74	4949
Trotting	0.76	0.76	0.76	215595
Tugging	1.00	0.01	0.01	4150
Walking	0.75	0.43	0.54	218203
accuracy			0.79	3183321
macro avg	0.89	0.50	0.56	3183321
weighted avg	0.80	0.79	0.78	3183321

(n_estimators=30, n_jobs=-1)

RandomForest

03. 랜덤 포레스트 학습 결과

	precision	recall	f1-score	support
<undefined>	0.72	0.86	0.78	1211693
Bowing	0.97	0.19	0.32	153
Carrying object	0.98	0.04	0.08	5318
Drinking	0.92	0.70	0.80	19510
Eating	0.88	0.29	0.44	50226
Extra_Synchronization	1.00	0.29	0.46	78
Galloping	0.93	0.03	0.06	3179
Jumping	1.00	0.01	0.03	1176
Lying chest	0.96	0.92	0.94	309890
Pacing	0.87	0.13	0.23	23046
Panting	0.93	0.86	0.90	250855
Playing	0.72	0.59	0.65	258716
Shaking	0.84	0.76	0.80	12246
Sitting	0.96	0.91	0.93	152750
Sniffing	0.79	0.92	0.85	307204
Standing	0.97	0.79	0.87	134384
Synchronization	0.98	0.61	0.75	4949
Trotting	0.76	0.77	0.77	215595
Tugging	1.00	0.01	0.01	4150
Walking	0.77	0.43	0.55	218203
accuracy			0.79	3183321
macro avg	0.90	0.51	0.56	3183321
weighted avg	0.80	0.79	0.78	3183321

```
(n_estimators=40, n_jobs=-1)
```

	precision	recall	f1-score	support
<undefined>	0.72	0.86	0.78	1211693
Bowing	1.00	0.20	0.34	153
Carrying object	1.00	0.04	0.08	5318
Drinking	0.92	0.70	0.80	19510
Eating	0.89	0.29	0.44	50226
Extra_Synchronization	1.00	0.32	0.49	78
Galloping	0.94	0.03	0.06	3179
Jumping	1.00	0.01	0.03	1176
Lying chest	0.96	0.92	0.94	309890
Pacing	0.88	0.13	0.22	23046
Panting	0.93	0.87	0.90	250855
Playing	0.73	0.59	0.65	258716
Shaking	0.83	0.77	0.80	12246
Sitting	0.96	0.91	0.94	152750
Sniffing	0.79	0.92	0.85	307204
Standing	0.97	0.80	0.88	134384
Synchronization	0.98	0.61	0.75	4949
Trotting	0.76	0.78	0.77	215595
Tugging	1.00	0.01	0.01	4150
Walking	0.77	0.43	0.55	218203
accuracy			0.79	3183321
macro avg	0.90	0.51	0.56	3183321
weighted avg	0.80	0.79	0.79	3183321

```
(n_estimators=50, n_jobs=-1)
```

n_estimators=40부터는 정확도 0.79에서 변화 없음

n_estimators=50 초과는 하드웨어 성능부족으로 학습 할 수 없음

Dog Behavior Analysis Dataset

KNN

K-Nearest Neighbor

인공지능공학

K-Nearest Neighbor

01. KNN 모델 선정 이유

KNN이란?

KNN(K-Nearest Neighbor)이란 지도학습 알고리즘으로, 주변 데이터를 보고 더 많은 데이터가 포함되어있는 범주로 분류하는 알고리즘.

	DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	ANeck_x	ANeck_y	ANeck_z	GBack_x	GBack_y	GBack_z	GNeck_x	GNeck_y	GNeck_z	Task	Behavior_1
0	16	1	0.00	0.041504	0.938965	-0.015137	-0.067871	-0.510254	-0.934570	-17.639161	-22.766115	7.446290	-7.934571	6.347657	13.427735	<undefined>	<undefined>
1	16	1	0.01	0.041992	0.941895	-0.020020	-0.128906	-0.494141	-0.913086	-15.075685	-11.413575	4.821778	-3.906250	4.394532	16.540528	<undefined>	Synchronization
2	16	1	0.02	0.040527	0.939453	-0.004395	-0.158691	-0.480469	-0.911133	-12.207032	-0.122070	2.807617	-0.488281	-1.953125	26.794435	<undefined>	Synchronization
3	16	1	0.03	0.021484	0.946289	0.007813	-0.122070	-0.486816	-0.880371	-9.460450	7.995606	1.586914	1.159668	-5.676270	38.085940	<undefined>	Synchronization
4	16	1	0.04	-0.000977	0.951172	0.033691	-0.053711	-0.500000	-0.807129	-8.361817	14.587403	-1.037598	4.577637	4.089356	41.503909	<undefined>	Synchronization
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10611063	74	2	1928.19	0.018066	0.436523	0.808105	-0.620605	0.326172	-0.647949	23.620607	75.317387	77.148442	-124.511726	107.727057	-122.497566	Task treat-search	Sniffing
10611064	74	2	1928.20	-0.021973	0.515625	0.813965	-0.671875	0.241211	-0.660645	34.729006	63.293461	79.956060	-116.455085	97.534186	-123.229988	Task treat-search	Sniffing
10611065	74	2	1928.21	-0.041504	0.507324	0.782227	-0.674805	0.244629	-0.627930	39.978030	49.316409	84.655767	-104.858405	95.458990	-124.816902	Task treat-search	Sniffing
10611066	74	2	1928.22	-0.032715	0.484375	0.824219	-0.685059	0.211426	-0.578613	38.024905	32.775881	89.538580	-87.463384	101.257330	-128.601082	Task treat-search	Sniffing
10611067	74	2	1928.23	0.033691	0.432617	0.785645	-0.664063	0.254395	-0.527344	26.306154	19.897462	89.477545	-64.025883	93.688971	-128.906258	Task treat-search	Sniffing

각 센서 데이터 값이 행동마다 비슷한 양상을 띄고 있을것이라 판단하여 모델 사용

02.

데이터 전처리
(Data preprocessing)

Dog Behavior Analysis Dataset

해당 데이터셋은 반려견의 등과 목에 부착한 두개의 센서를 통해 0.01초마다 반려견의 행동을 감지하여 움직임을 측정하여 기록되었다.

```
1 from sklearn.preprocessing import StandardScaler, MinMaxScaler
2 sc = StandardScaler()
3 train_x = sc.fit_transform(train_x)
4 test_x = sc.transform(test_x)
```

표준 스케일러를 이용해 데이터를 데이터를 표준화해줌

```
[[ 4.39318455e+00, -2.09267968e+00,  1.99371905e+00, ...,
   1.06126164e+00,  9.78473210e-01,  9.22351486e-02],
 [ 2.22785761e-01,  2.85225352e-02,  3.15477191e-01, ...,
   2.01285146e-01,  2.65664513e-01,  2.01925999e-01],
 [ 3.91562664e-01, -2.66474035e-01,  2.08978064e-01, ...,
  -2.64253315e-01, -1.51232713e-02, -6.62909348e-01],
 ...,
 [ 5.88469326e-01,  5.50062879e-01,  1.11354168e-01, ...,
   8.70166089e-02, -3.16401988e-02,  6.75000311e-04],
 [ 7.50013804e-01, -8.76838123e-01, -1.01841919e+00, ...,
   9.83389762e-01, -6.67541946e-01, -2.85763299e+00],
 [ 2.04301432e-01,  1.54848610e-02,  1.92116399e-01, ...,
  -1.44906190e-01,  6.95259853e-02,  1.06468999e-02]])
```

K-Nearest Neighbor

03. KNN 모델 학습

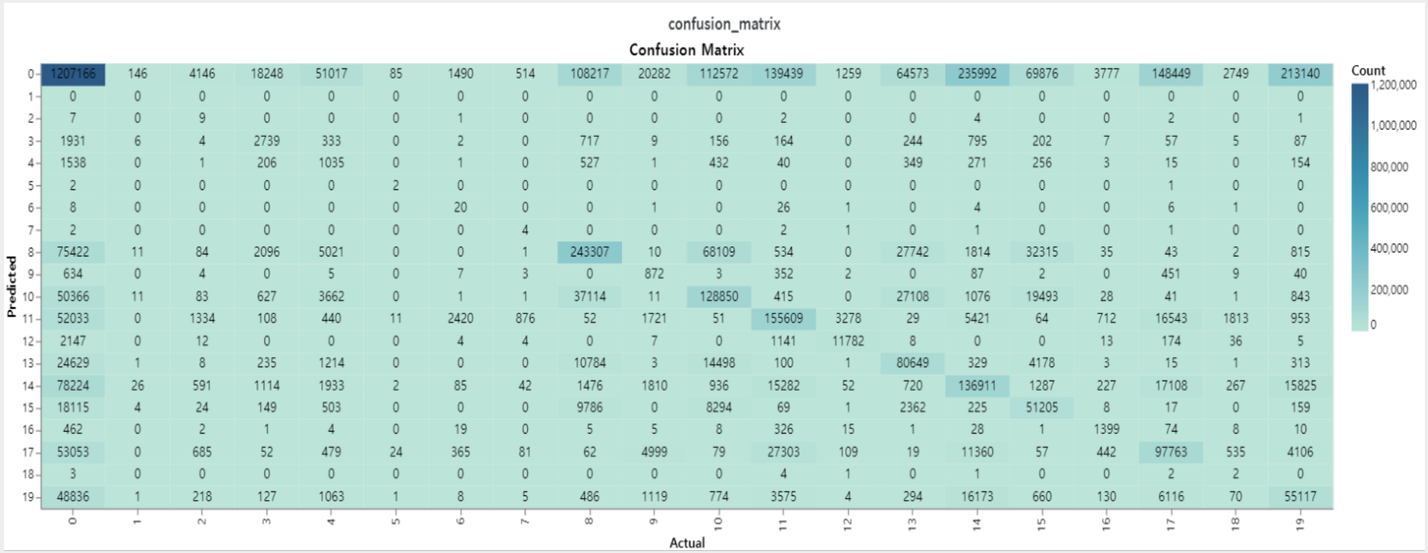
```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=15)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

01 KNN 모델 임포트

02 KNN 모델 학습
n_neighbors = 15
p=default(Euclidean)

```
wandb.sklearn.plot_confusion_matrix(test_y, y_pred, labels=clf.classes_)
```

03 혼동행렬 생성
(confusion_matrix)



K-Nearest Neighbor

03. KNN 모델 학습

```
1 from sklearn.metrics import classification_report, confusion_matrix
2 print(classification_report(test_y, y_pred))
```

04 정확도 확인
classification_report

	precision	recall	f1-score	support
0	0.50	0.75	0.60	1614578
1	0.00	0.00	0.00	206
2	0.35	0.00	0.00	7205
3	0.37	0.11	0.17	25702
4	0.21	0.02	0.03	66709
5	0.40	0.02	0.03	125
6	0.30	0.00	0.01	4423
7	0.36	0.00	0.01	1531
8	0.53	0.59	0.56	412533
9	0.35	0.03	0.05	30850
10	0.48	0.38	0.43	334762
11	0.64	0.45	0.53	344383
12	0.77	0.71	0.74	16506
13	0.59	0.40	0.47	204098
14	0.50	0.33	0.40	410492
15	0.56	0.29	0.38	179596
16	0.59	0.21	0.31	6784
17	0.49	0.34	0.40	286878
18	0.15	0.00	0.00	5493
19	0.41	0.19	0.26	231568
accuracy			0.51	4244428
macro avg	0.43	0.24	0.27	4244428
weighted avg	0.51	0.51	0.49	4244428

05 51%정도의 정확도가 나옴

Dog Behavior Analysis Dataset

LSTM

Long Short-Term Memory

인공지능공학

Long Short-Term Memory

01. LSTM 모델 선정 이유

LSTM이란?

LSTM(Long Short-Term Memory)은 **순환 신경망(RNN) 기법의 하나**로 셀, 입력 게이트, 출력 게이트, 망각 게이트를 이용해 기존 순환 신경망의 문제인 **기울기 소멸(vanishing gradient) 문제를 방지**하도록 개발되었다.

	DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	ANeck_x	ANeck_y	ANeck_z	GBack_x	GBack_y	GBack_z	GNeck_x	GNeck_y	GNeck_z	Task	Behavior_1
0	16	1	0.00	0.041504	0.938965	-0.015137	-0.067871	-0.510254	-0.934570	-17.639161	-22.766115	7.446290	-7.934571	6.347657	13.427735	<undefined>	<undefined>
1	16	1	0.01	0.041992	0.941895	-0.020020	-0.128906	-0.494141	-0.913086	-15.075685	-11.413575	4.821778	-3.906250	4.394532	16.540528	<undefined>	Synchronization
2	16	1	0.02	0.040527	0.939453	-0.004395	-0.158691	-0.480469	-0.911133	-12.207032	-0.122070	2.807617	-0.488281	-1.953125	26.794435	<undefined>	Synchronization
3	16	1	0.03	0.021484	0.946289	0.007813	-0.122070	-0.486816	-0.880371	-9.460450	7.995606	1.586914	1.159668	-5.676270	38.085940	<undefined>	Synchronization
4	16	1	0.04	-0.000977	0.951172	0.033691	-0.053711	-0.500000	-0.807129	-8.361817	14.587403	-1.037598	4.577637	4.089356	41.503909	<undefined>	Synchronization
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10611063	74	2	1928.19	0.018066	0.436523	0.808105	-0.620605	0.326172	-0.647949	23.620607	75.317387	77.148442	-124.511726	107.727057	-122.497566	Task treat-search	Sniffing
10611064	74	2	1928.20	-0.021973	0.515625	0.813965	-0.671875	0.241211	-0.660645	34.729006	63.293461	79.956060	-116.455085	97.534186	-123.229988	Task treat-search	Sniffing
10611065	74	2	1928.21	-0.041504	0.507324	0.782227	-0.674805	0.244629	-0.627930	39.978030	49.316409	84.655767	-104.858405	95.458990	-124.816902	Task treat-search	Sniffing
10611066	74	2	1928.22	-0.032715	0.484375	0.824219	-0.685059	0.211426	-0.578613	38.024905	32.775881	89.538580	-87.463384	101.257330	-128.601082	Task treat-search	Sniffing
10611067	74	2	1928.23	0.033691	0.432617	0.785645	-0.664063	0.254395	-0.527344	26.306154	19.897462	89.477545	-64.025883	93.688971	-128.906258	Task treat-search	Sniffing

10611068 rows × 20 columns

0.01초 단위로 측정된 시계열 데이터이기 때문에 순환신경망 기법의 하나인 LSTM 모델이 적합할 것이라고 판단함

Long Short-Term Memory

02.

데이터 전처리
(Data preprocessing)

Dog Behavior Analysis Dataset

해당 데이터셋은 반려견의 등과 목에 부착한 두개의 센서를 통해 0.01초마다 반려견의 행동을 감지하여 움직임을 측정하여 기록되었다.

DogID	TestNum	t_sec	ABack_x	ABack_y	ABack_z	...	Behavior_1
16	1	0.00	0.041504	0.938965	-0.015137		<undefined>
16	1	0.01	0.041992	0.941895	-0.020020		Synchronization
16	1	0.02	0.040527	0.939453	-0.004395		Synchronization
16	1	0.03	0.021484	0.946289	0.007813		Synchronization
16	1	0.04	-0.000977	0.951172	0.033691		Synchronization
...	...	...	...	...	...		...
16	1	0.96	-0.425781	-1.404297	-0.863770		Synchronization
16	1	0.97	-0.605469	-1.412598	-1.012695		Synchronization
16	1	0.98	-0.576172	-1.491211	-0.982910		Synchronization
16	1	0.99	-0.481934	-1.519531	-1.020508		Synchronization
16	1	1.00	-0.407715	-1.541992	-1.055176		Synchronization

01

1초 단위로 시퀀스 데이터를 구성

02

1초 단위로 데이터를 나누었을 때,
마지막 행동을 y로 설정

Long Short-Term Memory

02.

데이터 전처리
(Data preprocessing)

Dog Behavior Analysis Dataset

해당 데이터셋은 반려견의 등과 목에 부착한 두개의 센서를 통해 0.01초마다 반려견의 행동을 감지하여 움직임을 측정하여 기록되었다.

```
def seq2dataset(seq,window,horizon):
    X=[]; Y=[]
    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        y=dogmove_data['Behavior_1'][i+window-1] #윈도우 크기의 데이터. 마지막 행동을 y로 설정
        X.append(x); Y.append(y)
    return np.array(X), np.array(Y)

w=100 # 윈도우 크기 # (0.01 * 100) = 1초동안의 데이터를 하나의 윈도우로 묶음
h=1 # 수평선 계수

X,Y = seq2dataset(seq,w,h)
```

시계열 데이터를 윈도우 단위로 자르는 함수를 생성하고
윈도우의 크기를 100으로 설정하여 1초동안의 데이터를 하나의 윈도우로 묶음

윈도우에서 마지막 행동 데이터('Behavior_1 Column')를 y로 설정

Long Short-Term Memory

02.

데이터 준비
(Data prep

데이터셋의 용량이 너무 커서 Google Colab에서 세션이 다운되는 문제 발생

Dog Behavior Analysis

해당 데이터셋은 반려견의 등과 목에
를 통해 0.01초마다 반려견의 행동을
측정하여 기록되었다.

```
f=open("/content/drive/MyDrive/data/DogMoveData_csv_format/DogMoveData.csv","r")
dogmove_data=pd.read_csv(f,header=0)
split_data = dogmove_data.groupby(['DogID', 'TestNum']).get_group((16, 1))
seq=split_data[['ABack_x','ABack_y','ABack_z','ANeck_x', 'ANeck_y', 'ANeck_z',
                'GBack_x', 'GBack_y', 'GBack_z', 'GNeck_x', 'GNeck_y', 'GNeck_z']].to_numpy()
```

데이터 셋을 불러오고, 모든 센서 측정값 column을 데이터로 사용하도록 설정

```
def seq2dataset(seq,window,horizon):
    X = []; Y = []
    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        y=dogmove_data['Behavior_1'][i+window] #윈도우 크기의 데이터. 마지막 행동을 y로 설정
        X.append(x); Y.append(y)
    return np.array(X), np.array(Y)
```

사용 가능한 RAM을 모두 사용한 후 세션이 다운되었습니다. 런타임 로그 보기 ×

```
X,Y = seq2dataset(seq,w,h)
```

시계열 데이터를 윈도우 단위로 자르는 함수를 생성하고
윈도우의 크기를 100으로 설정하여 1초동안의 데이터를 하나의 윈도우로 묶음

윈도우에서 마지막 행동 데이터('Behavior_1 Column')를 y로 설정

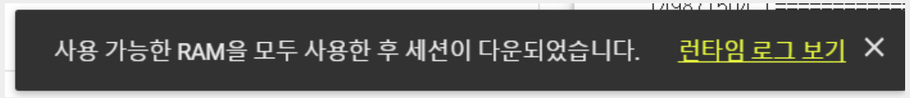
02.

데이터 전처리
(Data preprocessing)

Dog Behavior Analysis Dataset

해당 데이터셋은 반려견의 등과 목에 부착한 두개의 센서를 통해 0.01초마다 반려견의 행동을 감지하여 움직임을 측정하여 기록되었다.

데이터셋의 용량이 너무 커서 Google Colab에서 세션이 다운되는 문제 발생



```
f=open("/content/drive/MyDrive/data/DogMoveData_csv_format/DogMoveData.csv","r")
dogmove_data=pd.read_csv(f,header=0)
split_data = dogmove_data.groupby(['DogID', 'TestNum']).get_group((16, 1))
seq=split_data[['ABack_x','ABack_y','ABack_z','ANeck_x', 'ANeck_y', 'ANeck_z',
                'GBack_x', 'GBack_y', 'GBack_z', 'GNeck_x', 'GNeck_y', 'GNeck_z']].to_numpy()
```

데이터 양을 줄이기 위해
DogID가 16이고 TestNum이 1인 데이터로만 학습을 진행함

Long Short-Term Memory

03. LSTM 모델 학습

```
hidden_units = 30
num_classes = len(set(Y)) # seq2dataset 메소드를 통해 나눈 class의 종류를 의미

model = Sequential()
model.add(LSTM(hidden_units))
model.add(Dense(num_classes, activation='softmax'))
```

01

hidden layer를 30개로 설정하고
데이터 전처리를 통해 나누어진 class 개수를
이용해 모델 생성

Long Short-Term Memory

03. LSTM 모델 학습

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('hidden30_best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

02

Early Stopping

validation set의 loss가 최소화되도록 모니터링하고
4번의 epoch 동안 성능이 증가하지 않으면
학습을 멈추도록 설정

03

Model Checkpoint

validation accuracy를 모니터링하면서
이전 epoch에 비해 validation performance가 좋은 경우
해당 모델을 저장하도록 설정

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
history = model.fit(x_train, y_train, batch_size=128, epochs=100, callbacks=[es, mc], validation_data=(x_test, y_test))
```

04 모델 컴파일 및 학습 진행

Long Short-Term Memory

04.

LSTM 학습 결과

Training Parameter

- hidden Layer : 30
- batch size : 128
- Loss Function
: Categorical Crossentropy

```
Epoch 1/100
769/769 [==============================] - ETA: 0s - loss: 1.2558 - acc: 0.5897
Epoch 1: val_acc improved from -inf to 0.69475, saving model to hidden30_best_model.h5
769/769 [==============================] - 64s 80ms/step - loss: 1.2558 - acc: 0.5897 - val_loss: 0.8964 - val_acc: 0.6947
Epoch 2/100
768/769 [==============================>.] - ETA: 0s - loss: 0.7845 - acc: 0.7286
Epoch 2: val_acc improved from 0.69475 to 0.75925, saving model to hidden30_best_model.h5
769/769 [==============================] - 52s 67ms/step - loss: 0.7845 - acc: 0.7286 - val_loss: 0.6881 - val_acc: 0.7593
```

•
•
•

```
Epoch 53/100
768/769 [==============================>.] - ETA: 0s - loss: 0.1189 - acc: 0.9614
Epoch 53: val_acc did not improve from 0.96257
769/769 [==============================] - 54s 70ms/step - loss: 0.1189 - acc: 0.9614 - val_loss: 0.1266 - val_acc: 0.9574
Epoch 53: early stopping
```

Epoch 53

53 epoch에서
Early Stopping진행

학습 초기에 비해
Loss 감소
Accuracy 증가

모델 성능 증가

Long Short-Term Memory

04.

LSTM 학습 결과

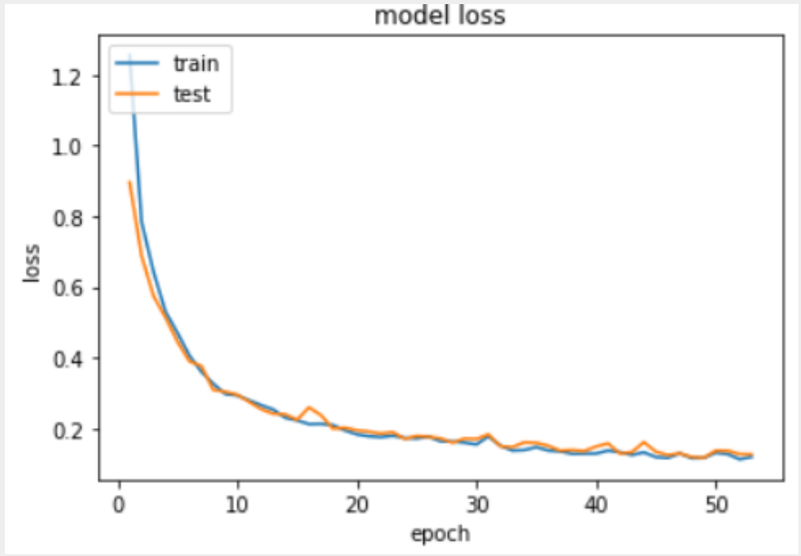
Training Parameter

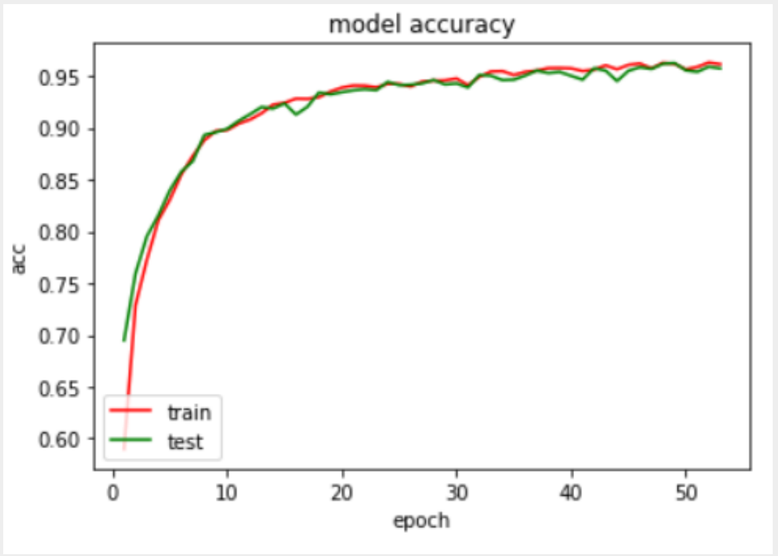
- Hidden Layer : 30
- Batch Size : 128
- Loss Function
: Categorical Crossentropy

```
Epoch 1/100
769/769 [==============================] - ETA: 0s - loss: 1.2558 - acc: 0.5897
Epoch 1: val_acc improved from -inf to 0.69475, saving model to hidden30_best_model.h5
769/769 [==============================] - 64s 80ms/step - loss: 1.2558 - acc: 0.5897 - val_loss: 0.8964 - val_acc: 0.6947
Epoch 2/100
768/769 [==============================>.] - ETA: 0s - loss: 0.7845 - acc: 0.7286
Epoch 2: val_acc improved from 0.69475 to 0.75925, saving model to hidden30_best_model.h5
769/769 [==============================] - 52s 67ms/step - loss: 0.7845 - acc: 0.7286 - val_loss: 0.6881 - val_acc: 0.7593
```

•
•
•

```
Epoch 53/100
768/769 [==============================>.] - ETA: 0s - loss: 0.1189 - acc: 0.9614
Epoch 53: val_acc did not improve from 0.96257
769/769 [==============================] - 54s 70ms/step - loss: 0.1189 - acc: 0.9614 - val_loss: 0.1266 - val_acc: 0.9574
Epoch 53: early stopping
```





Long Short-Term Memory

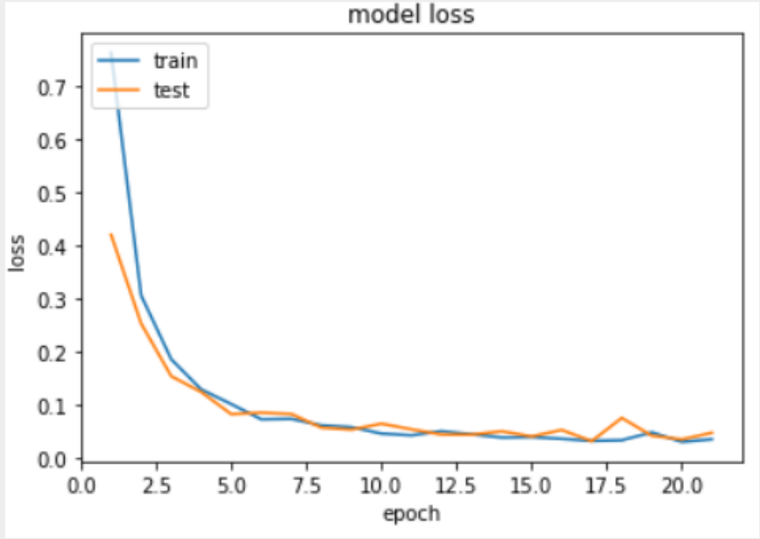
05.

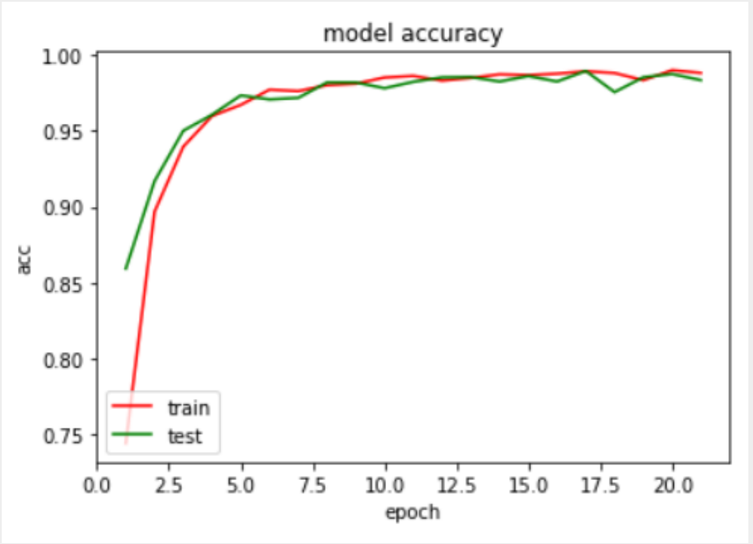
LSTM 하이퍼 파라미터 튜닝

Hidden Layer 30개 → 128개

- Hidden Layer : 128
- Batch Size : 128
- Loss Function
: Categorical Crossentropy

```
Epoch 21/100
769/769 [==============================] - ETA: 0s - loss: 0.0362 - acc: 0.9882
Epoch 21: val_acc did not improve from 0.98949
769/769 [==============================] - 252s 327ms/step - loss: 0.0362 - acc: 0.9882 - val_loss: 0.0483 - val_acc: 0.9834
Epoch 21: early stopping
```





Epoch 21

21 epoch에서
Early Stopping진행

0.96 0.9882

Accuracy 상승

0.11 0.0362

Loss 감소

Long Short-Term Memory

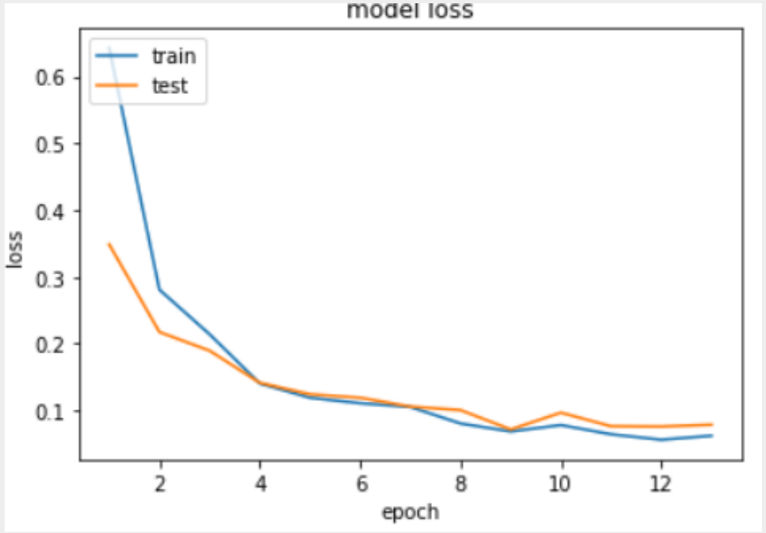
05.

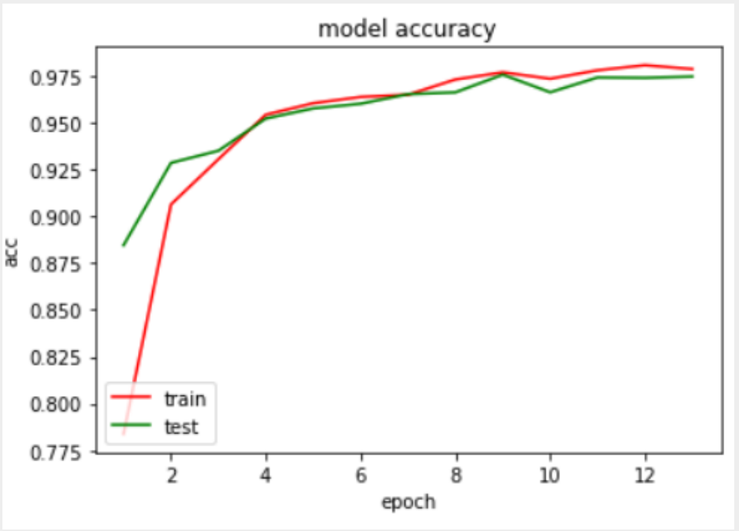
LSTM 하이퍼 파라미터 튜닝

Batch Size 128 → 64

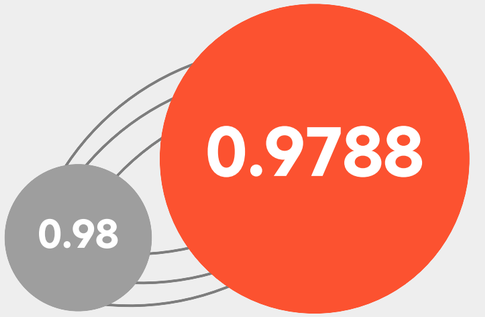
- Hidden Layer : 128
- Batch Size : 64
- Loss Function
: Categorical Crossentropy

```
Epoch 13/100
1537/1537 [==============================] - ETA: 0s - loss: 0.0623 - acc: 0.9788
Epoch 13: val_acc did not improve from 0.97579
1537/1537 [==============================] - 343s 223ms/step - loss: 0.0623 - acc: 0.9788 - val_loss: 0.0788 - val_acc: 0.9748
Epoch 13: early stopping
```





13 epoch에서
Early Stopping진행



Accuracy 감소

Loss 증가

Long Short-Term Memory

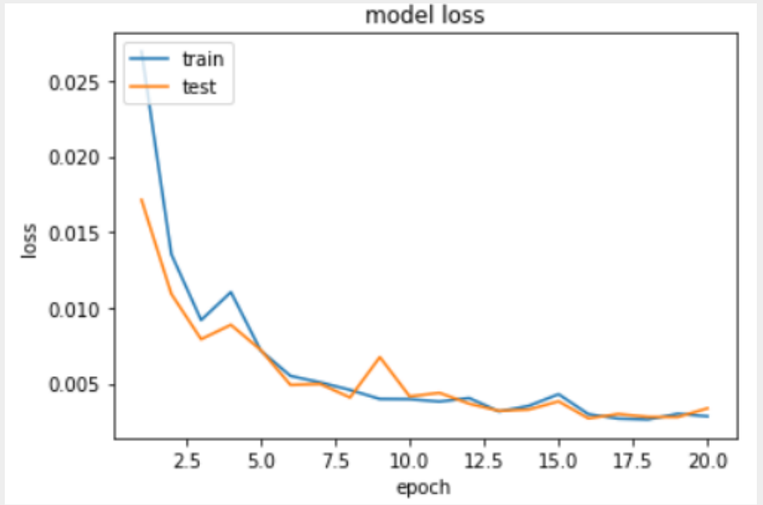
05.

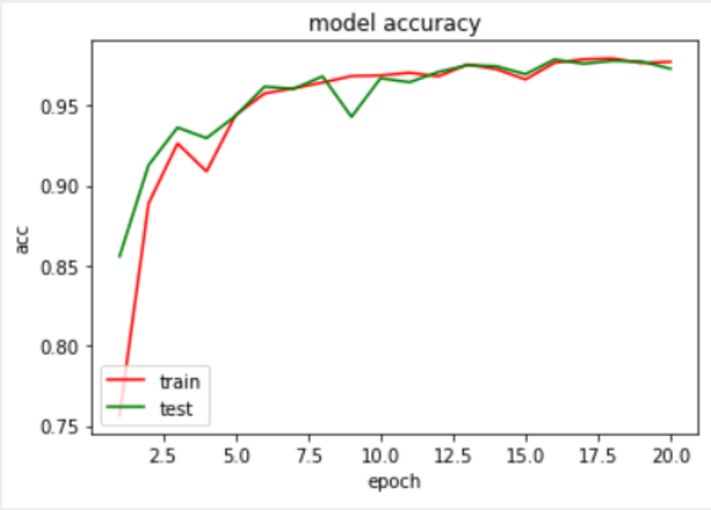
LSTM 하이퍼 파라미터 튜닝

Loss Function을 MSE로 변경

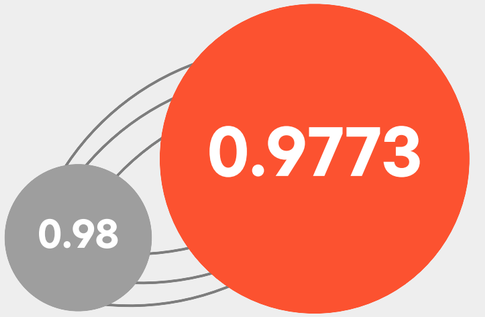
- Hidden Layer : 128
- Batch Size : 128
- Loss Function
: Mean Squared Error

```
Epoch 20/100
769/769 [==============================] - ETA: 0s - loss: 0.0028 - acc: 0.9773
Epoch 20: val_acc did not improve from 0.97880
769/769 [==============================] - 256s 333ms/step - loss: 0.0028 - acc: 0.9773 - val_loss: 0.0034 - val_acc: 0.9730
Epoch 20: early stopping
```





20 epoch에서
Early Stopping진행



Accuracy 감소

Loss 감소

Dog Behavior Analysis Dataset

결론

conclusion

인공지능공학

conclusion

결론

1. 4종류의 모델중 LSTM이 가장 높은 정확도를 가지고 있었음

   SVM - N/A KNN - 0.51 RF - 0.79 LSTM - 0.988

2. 1000만개가 넘는 데이터를 돌리기에
   하드웨어 리소스 부족으로 데이터를 축소해 학습

   DogID = 16, TestNum = 1

   데이터 손실 없이 데이터를 축소시키는 방법을 연구
   or 더 괜찮은 성능을 가진 하드웨어를 이용해 학습

감사합니다
감사합니다

Artificial intelligence engineering

2022-2 인공지능공학

6팀 기말 팀프로젝트

Dog Behavior Analysis Dataset