AI Hub Easy Builder 사용

사용 의의

- 이지 빌더의 데이터 셋을 이용해 Epoch, layer 수들을 변경해 가며 정확도가 높은 학습 모델을 만든다.

- 그 후 모델을 추출하여 다른 프로젝트에 사용한다.

1. 데이터 셋 선택

- 이지 빌더에서 제공되는 데이터 셋은 아래와 같다.
- 한국형 사물이미지는 아예 다른 이미지들을 학습시키는 것에
- 위해 물품은 가방 을 공항 검색대에서 스캔한 것으로, 전체 이미지 중 사이즈가 작은 물품들을 학습 시키는 것에
- 농업(토마토)는 같은 대상의 상태에 대해 학습 시키는 것에 적합한 것으로 보인다.

한국형 사물이미지

국내 특성에 맞는
궁궐, 가옥, 탑, 무덤, 사찰 이미지

위해물품 이미지

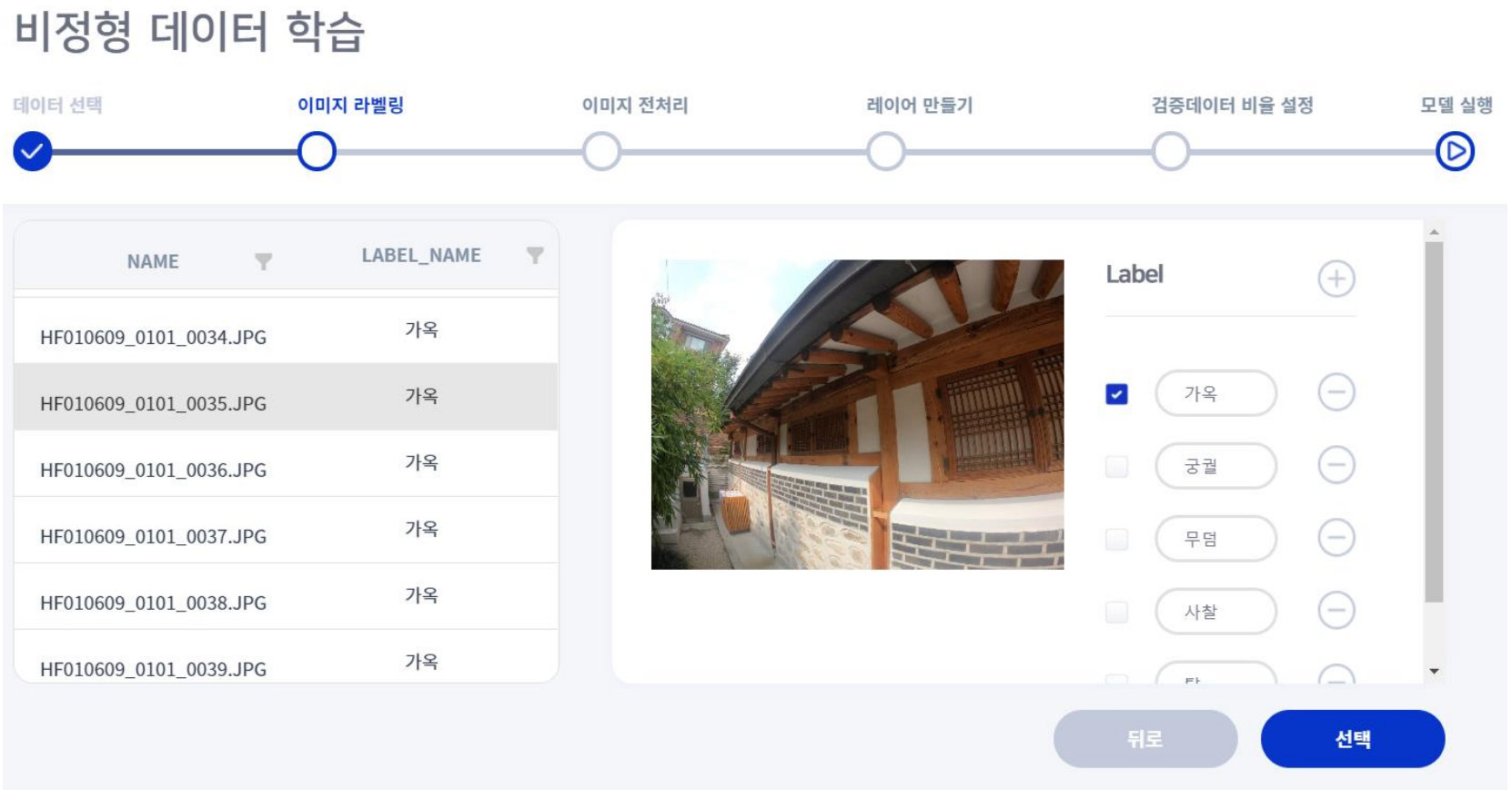
국내 공항 항만에 사용중인
엑스레이 스캐너 3종 장비를
이용한 이미지

농업 (토마토)

토마토 11종, 촬영 가이드에 따라
온습도, 촬영각도, 장소 등의 메타
정보를 포함하여 데이터 셋 구축

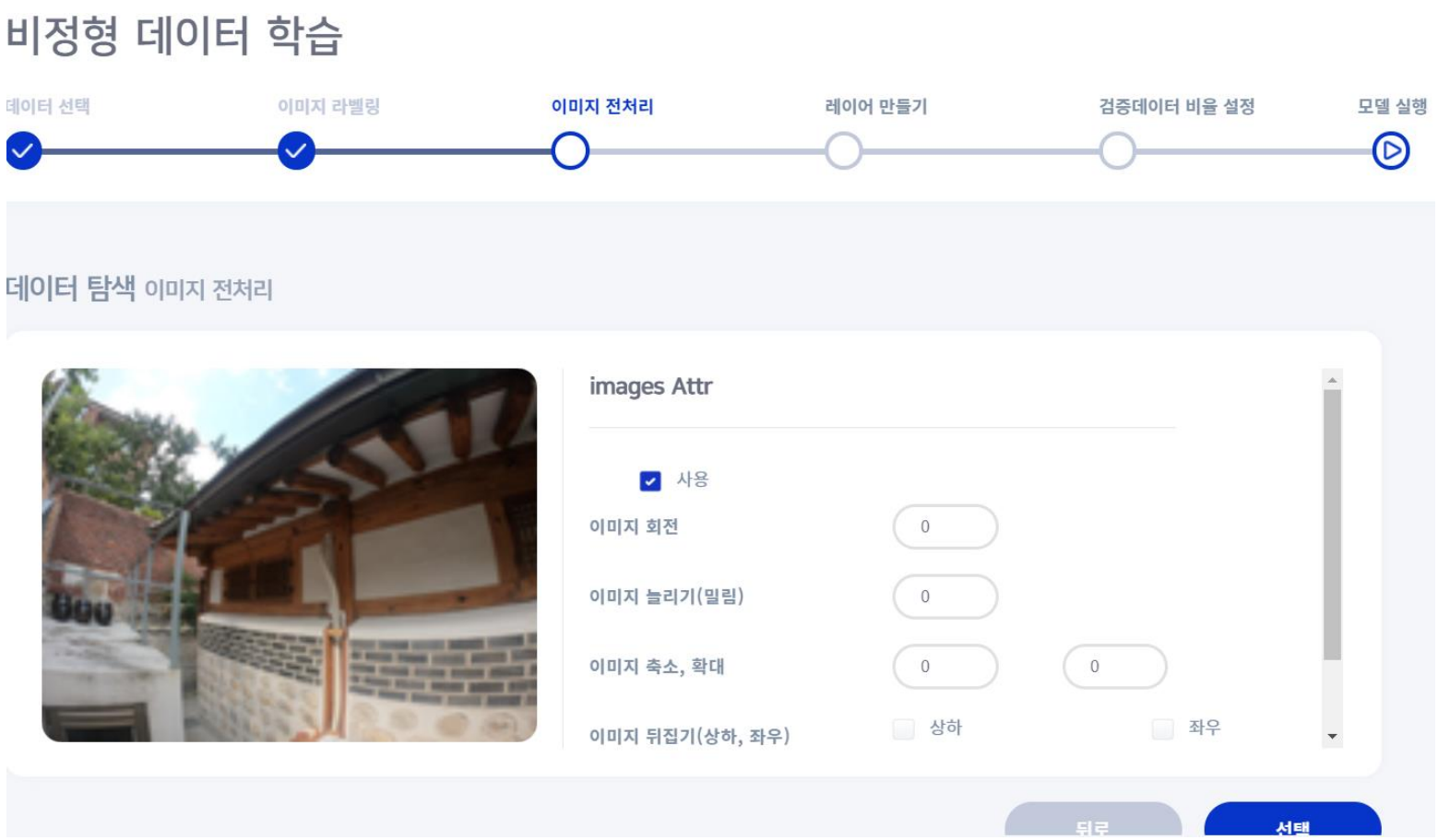
2. 이미지 라벨링

- 테스트 셋들의 이미지 라벨들을 변경해 줄 수 있습니다.
- 이미 주어진 데이터 셋에는 올바른 라벨들이 붙어있으니 이 단계는 스킵하겠습니다.



3. 이미지 전처리

- 인풋으로 사용할 이미지를 전처리합니다.



3. 이미지 전처리

images Attr

사용

이미지 회전 20

이미지 늘리기(밀림) 0

이미지 축소, 확대 0 0

이미지 뒤집기(상하, 좌우) 상하 좌우

images Attr

사용

이미지 회전 0

이미지 늘리기(밀림) 20

이미지 축소, 확대 0 0

이미지 뒤집기(상하, 좌우) 상하 좌우

3. 이미지 전처리

사용

이미지 회전 0

이미지 늘리기(밀림) 0

이미지 축소, 확대 100 30

이미지 뒤집기(상하, 좌우) 상하 좌우

이미지 리사이즈(넓이, 높이) 150 150

축소, 확대는 가시적 확인 불가

사용

이미지 회전 0

이미지 늘리기(밀림) 0

이미지 축소, 확대 0 0

이미지 뒤집기(상하, 좌우) 상하

이미지 리사이즈(넓이, 높이) 150 150

사용

이미지 회전 0

이미지 늘리기(밀림) 0

이미지 축소, 확대 0 0

이미지 뒤집기(상하, 좌우) 상하 좌우

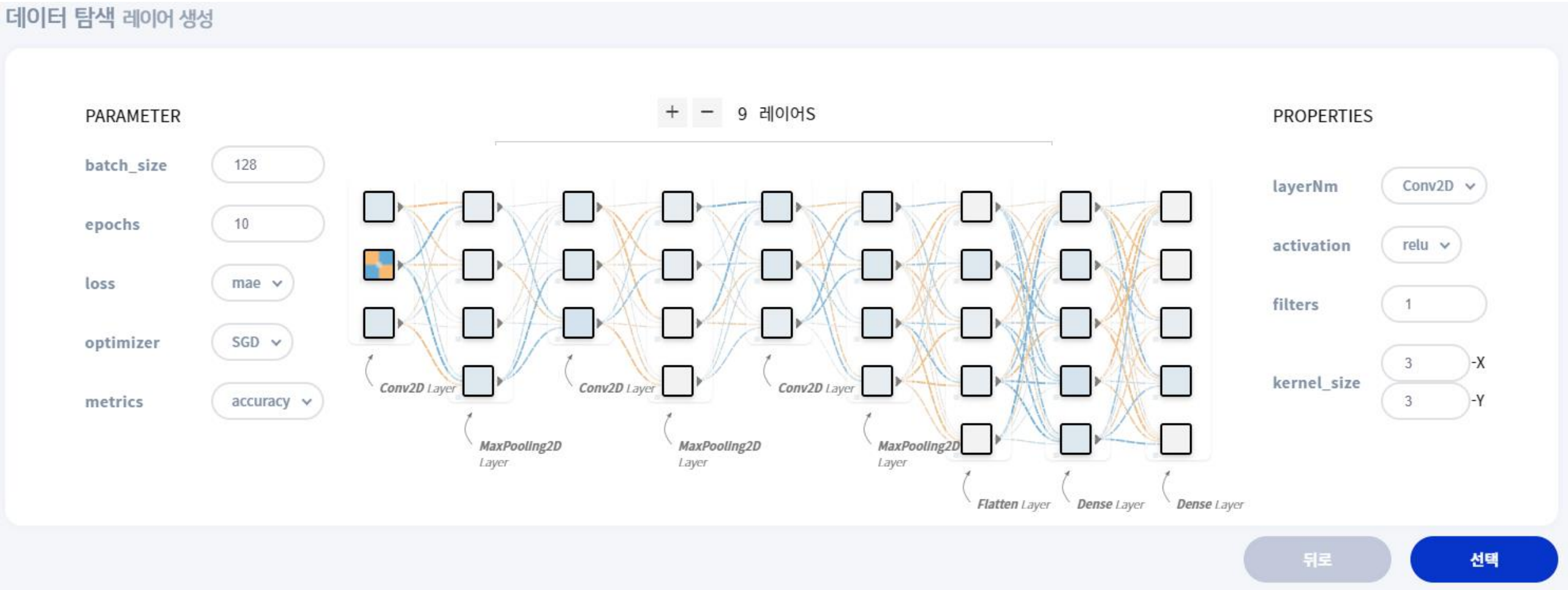
이미지 리사이즈(넓이, 높이) 150 150

3. 이미지 전처리

사진 크기는 동일해 보이나, 해상도 차이가 있음

이제 전처리를 완료했다면 레이어를 편집 해야함.

4. 레이어(모델) 만들기


데이터 탐색 레이어 생성
PARAMETER
batch_size
128
epochs
10
loss
mae
optimizer
SGD
metrics
accuracy
+
−
9 레이어S
Conv2D Layer
MaxPooling2D Layer
Conv2D Layer
MaxPooling2D Layer
Conv2D Layer
MaxPooling2D Layer
Flatten Layer
Dense Layer
Dense Layer
PROPERTIES
layerNm
Conv2D
activation
relu
filters
1
kernel_size
3
-X
3
-Y
뒤로
선택

4. 레이어(모델) 만들기

batch size : 한번에 데이터를 몇 개씩 넘겨주는가?

epochs : 몇 회 반복할 것인가?

loss(손실함수) : mae / mse / mape 가능

MAE와 MSE의 차이는 무엇인가?
-MSE는 제곱을 해주고 / MAE는 제곱은 하지 않고 절대값을 구한다.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

MAE와 MSE의 공통점은 무엇인가?
- 실제값과 측정값을 빼주는 것
- 평균을 내주는 것 (M - mean)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

→ 크기 의존적 에러 발생 가능성 존재

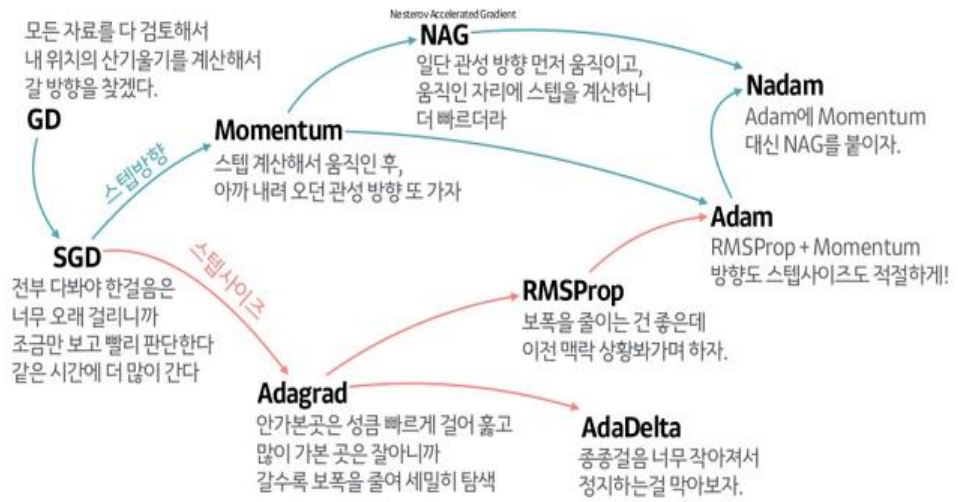
MAPE는 크기 의존적 에러의 단점을 커버하기 위한 모델입니다. MAPE의 공식은 아래와 같습니다.

$$\mathrm{M} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|,$$

At 는 실제 값이고, Ft 는 예측 값입니다.

optimizer(최적함수) : SGD / RMSProp / Adagrad / Adadelta / Adam / Adamax / Nadam 가능

<Optimizer의 종류>



출처 : https://www.slideshare.net/yongho/ss-79607172

4. 레이어(모델) 만들기

척도(Metrics) : Accuracy / BinaryAccuracy / CategoricalAccuracy 가 있다.

Accuracy : 예측 값이 실제 라벨과 얼마나 맞는가 → 전체 오차율이 낮아지는 방향으로 (but, 정답일 확률이 올라가는 것을 의미하지 않음

Binary Accuracy : binary label과 비교 [0, 1, 1, 0, 1] 이런 것이 바이너리 라벨

Categorical Accuracy : One-Hot 라벨과 비교 (원 핫은 실제 값을 밑에 t 처럼 정답만 1, 나머지는 0으로 한 것)

추가적으로 공부
교차 엔트로피 오차로 계산 → 정답 제외 나머지를 제거, 정답에 가깝도록 만듦. 이렇게 설정해 두면, [2]를 제외한 나머지 예측값들은 * 0 되어서 신경 X.
그저 [2]의 정답이 높아지는 쪽으로만

교차엔트로피 오차 CEE (Cross Entropy Error)

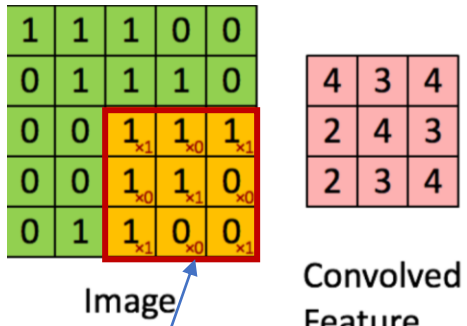
$$E = -\sum t_i \cdot log(y_i)$$

t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

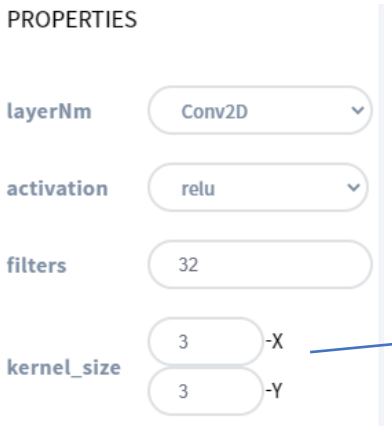
4. 레이어(모델) 만들기

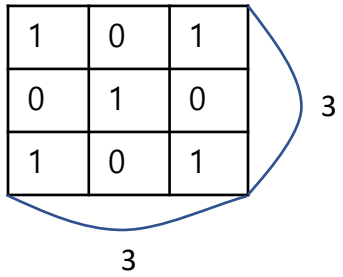
보통 { Conv2D + Maxpolling2D } X a + Flatten X b + Dense X c 로 구성

Conv2D : Filter로 특징을 뽑아주는 역할을 함.
(압축 비슷한 느낌)



Convolved
Feature



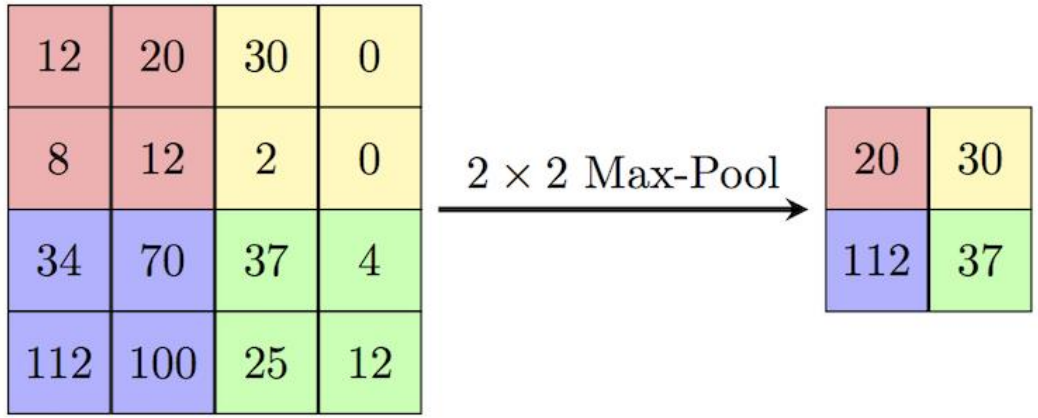


이러한 filter (안에 값은 다 조금씩 다름)를 32개로 해서,
한 이미지에 대해 32번 반복

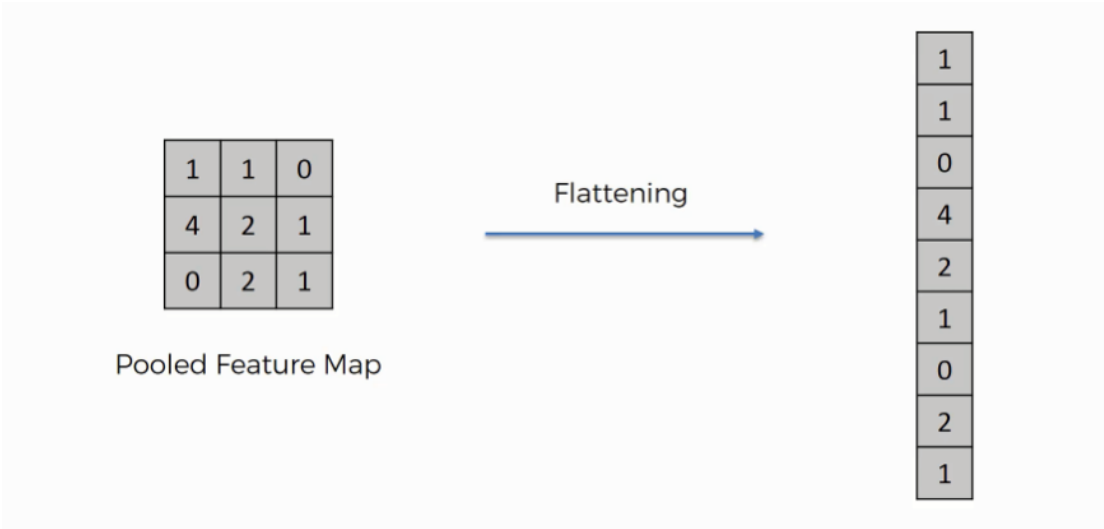
(1, 1),(2, 2), (3, 3)처럼 (x, x) 만 가능함.

4. 레이어(모델) 만들기

Maxpooling

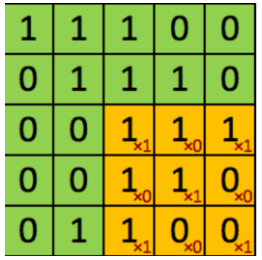


Flatten



4. 레이어(모델) 만들기

Dense Layer



Dot 연산

5. Test1

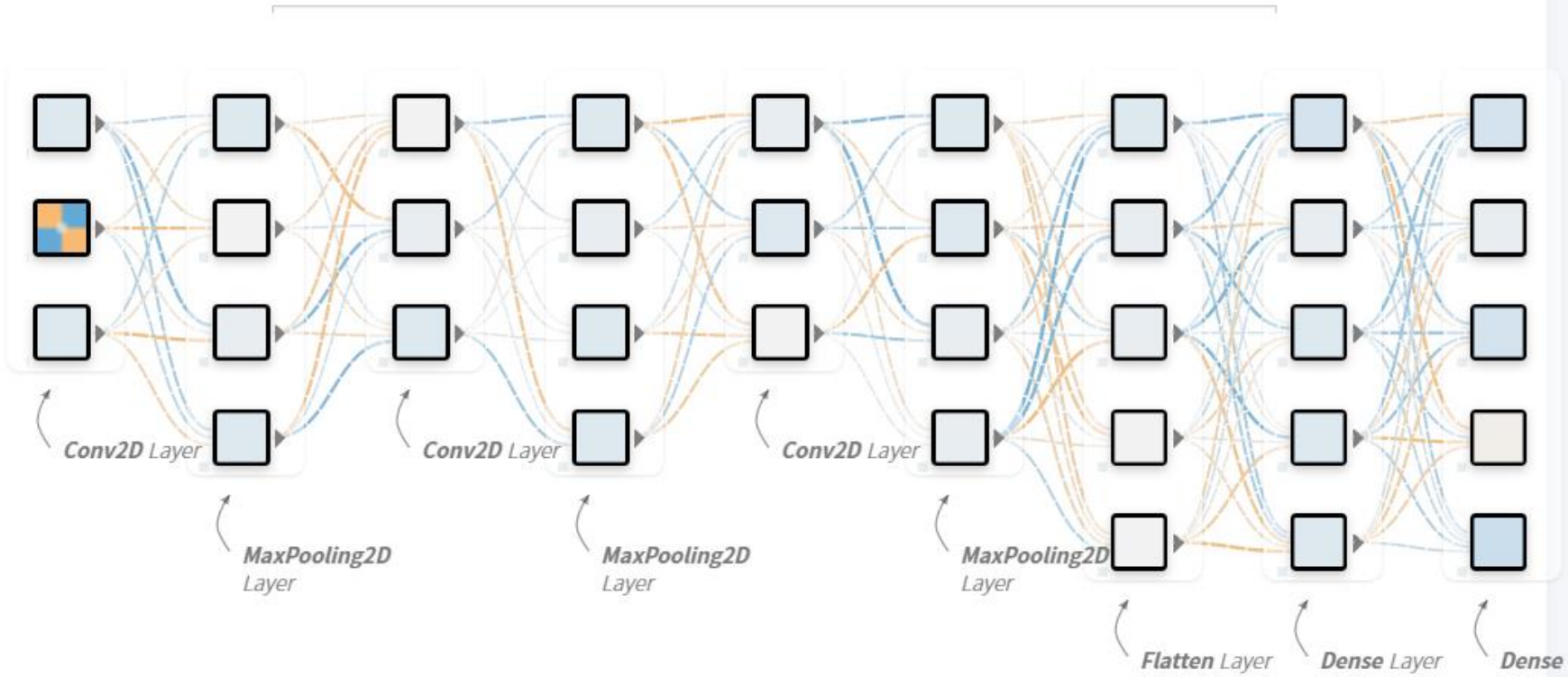
128

5

mae

SGD

accuracy



layerNm Conv2D

activation relu

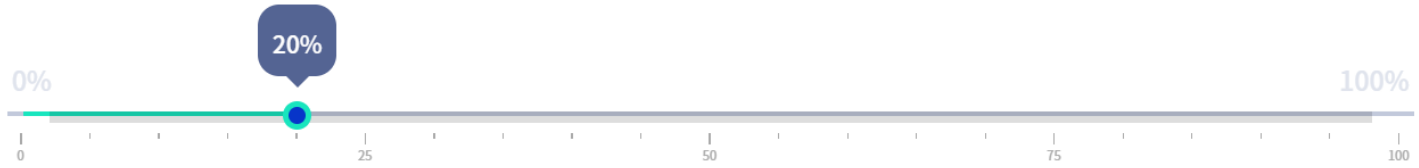
filters 32

3

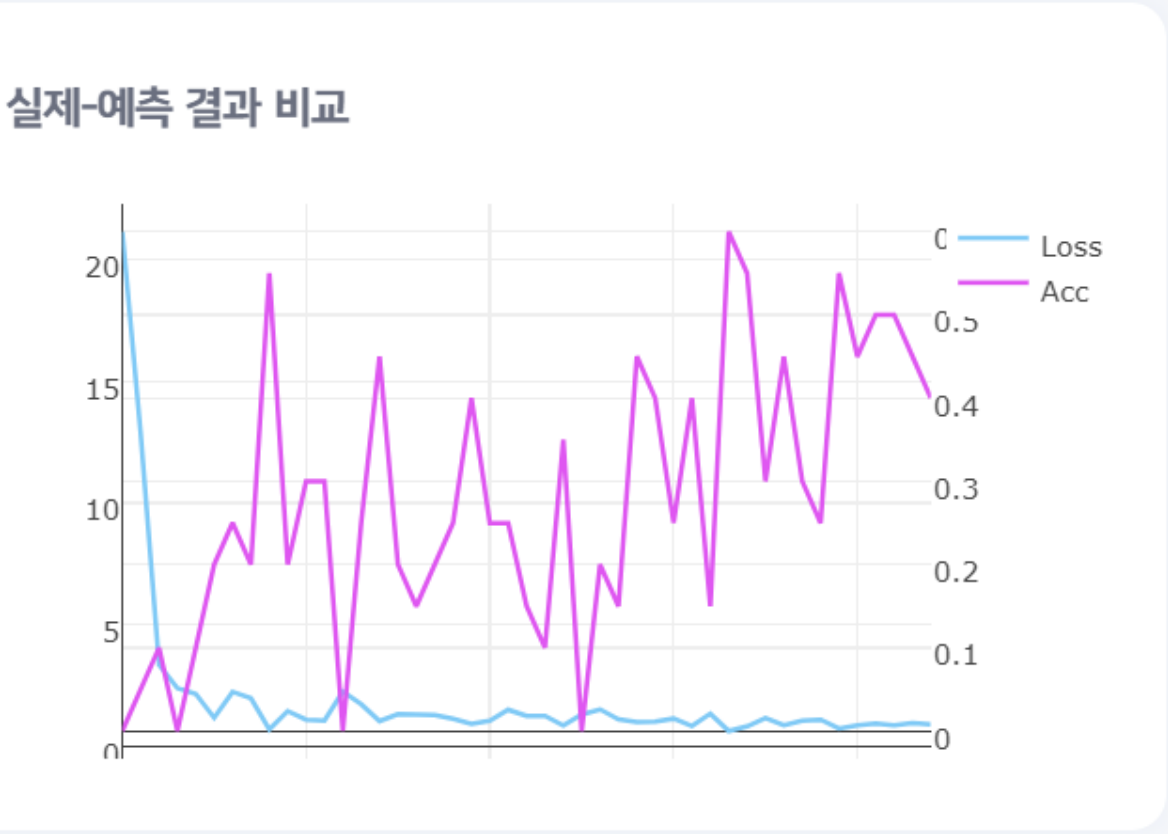
kernel_size -X 3 -Y

평가 데이터 분할 비율

평가 데이터의 비율은 2~98% 사이여야 합니다.



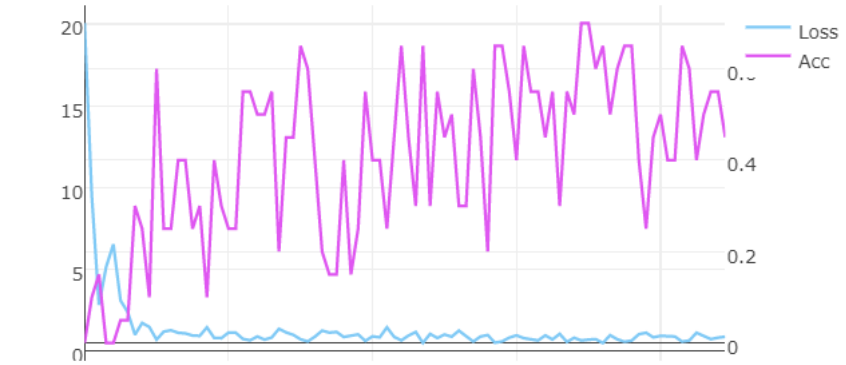
5. Test1 결과



Epochs	Status	Loss	Acc	Val loss	Val acc
1	실행완료	4.97	0.17	1.23	0.15
2	실행완료	1.36	0.23	1.4	0.16
3	실행완료	1.21	0.22	1.19	0.19
4	실행완료	1.01	0.34	0.93	0.44
5	실행완료	0.9	0.43	0.97	0.48

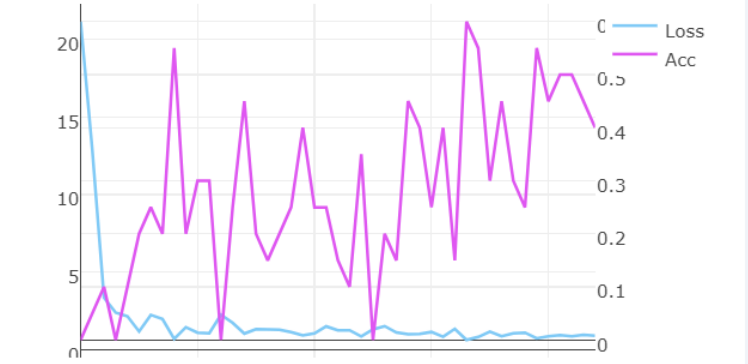
5. Test2 – 추후 비교는 Test2를 기준으로 하겠습니다.

5. Test2 결과



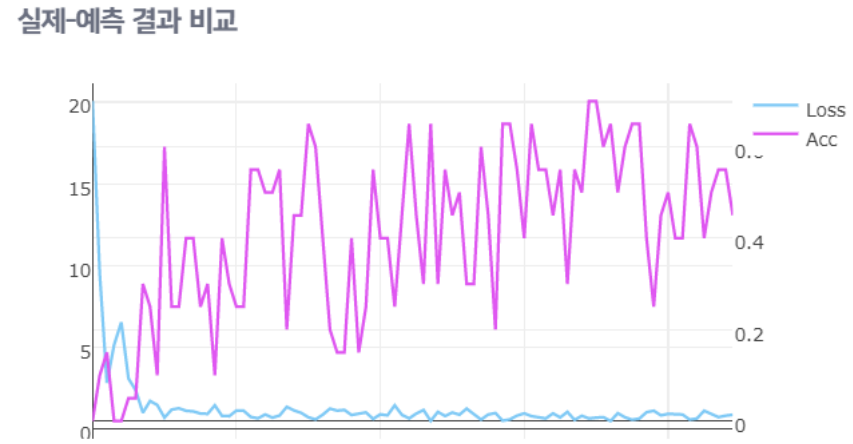
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

5. Test1 – Test2 Epoch 를 조금 늘릴 경우



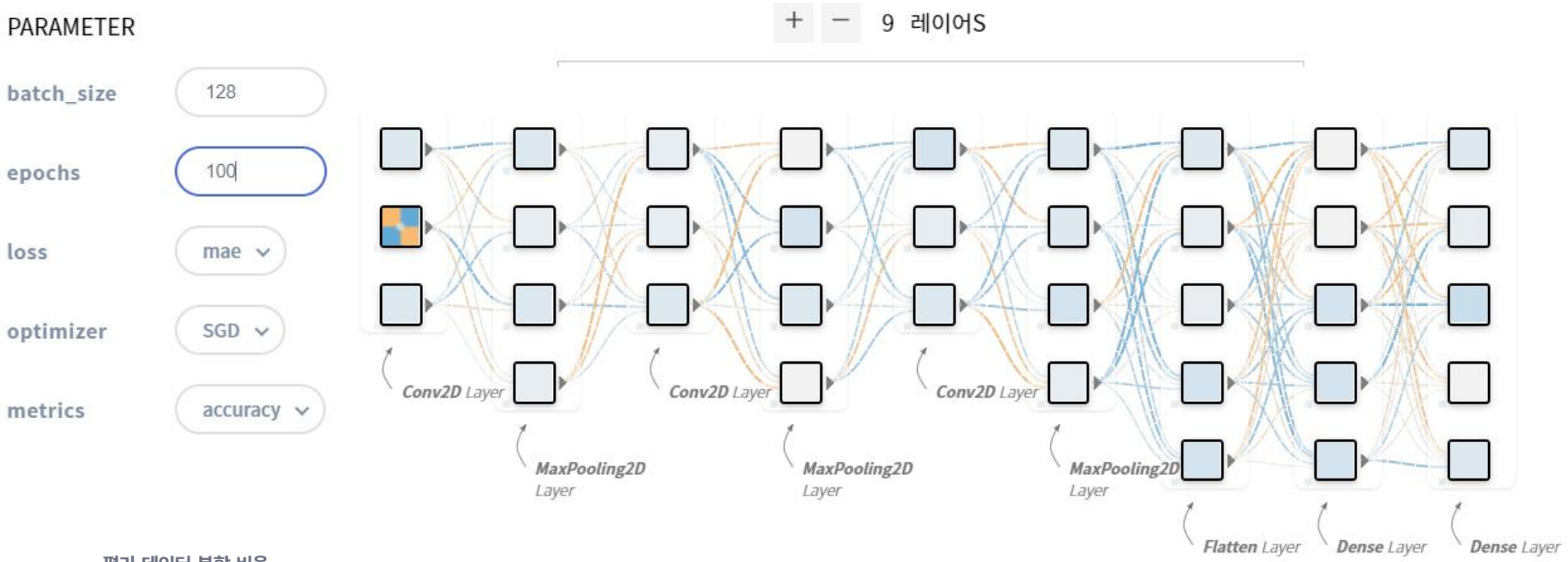
Epochs	Status	Loss	Acc	Val loss	Val acc
1	실행완료	4.97	0.17	1.23	0.15
2	실행완료	1.36	0.23	1.4	0.16
3	실행완료	1.21	0.22	1.19	0.19
4	실행완료	1.01	0.34	0.93	0.44
5	실행완료	0.9	0.43	0.97	0.48

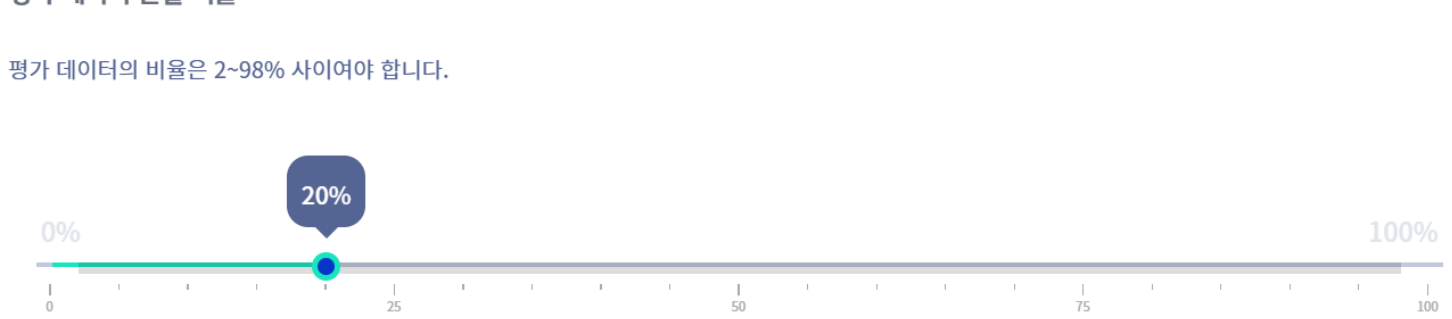
정확도가 조금 더 높아졌음을 알 수 있습니다.



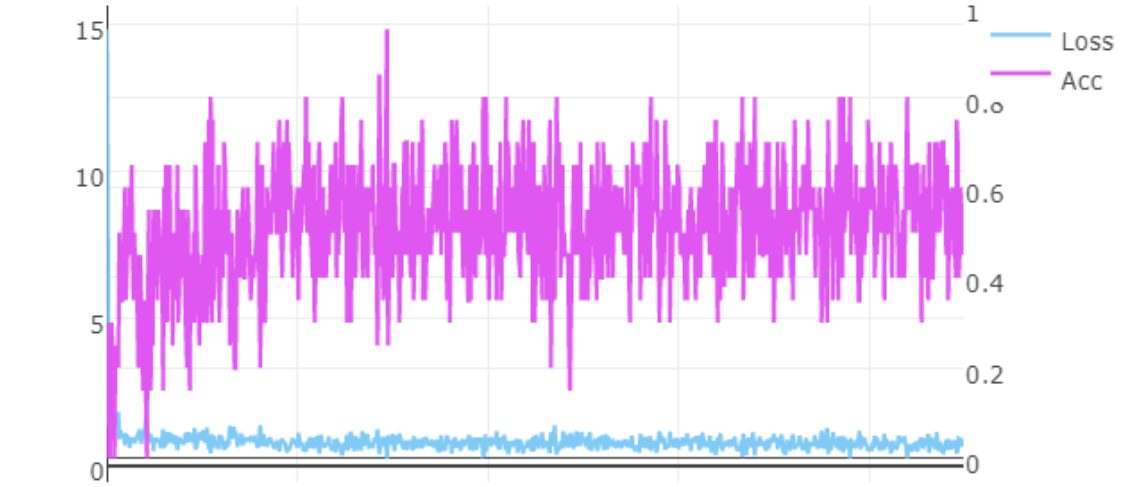
Epochs	Status	Loss	Acc	Val loss	Val acc
5	실행완료	0.9	0.38	1.03	0.4
6	실행완료	0.93	0.43	0.86	0.53
7	실행완료	0.74	0.54	0.74	0.63
8	실행완료	0.75	0.55	0.78	0.55
9	실행완료	0.79	0.52	0.79	0.49
10	실행완료	0.81	0.52	0.74	0.58

5. Test2-1. Epoch를 100으로 확 늘릴경우





5. Test2-1 결과

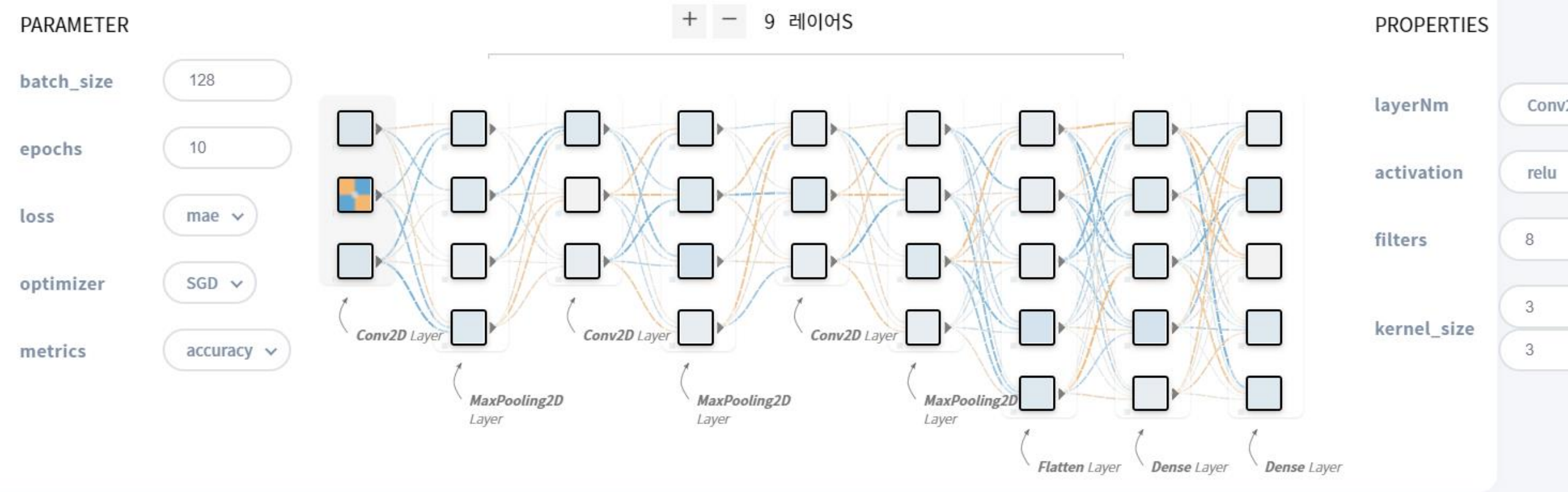


	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	93	실행완료	0.82	0.51	0.8	0.49
▸	94	실행완료	0.66	0.6	0.74	0.55
▸	95	실행완료	0.74	0.56	0.88	0.47
▸	96	실행완료	0.78	0.5	0.81	0.52
▸	97	실행완료	0.78	0.54	0.79	0.56

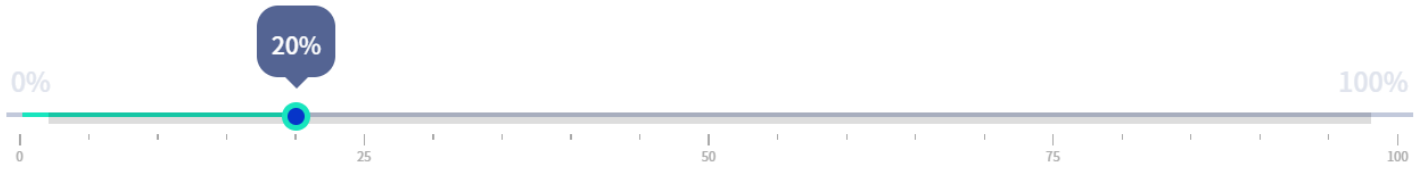
1 2 3 4 5 6 7 8 9 10

전반적으로 정확도가 0.5~0.6사이로 조금 높아졌지만, 크게 차이가 없습니다

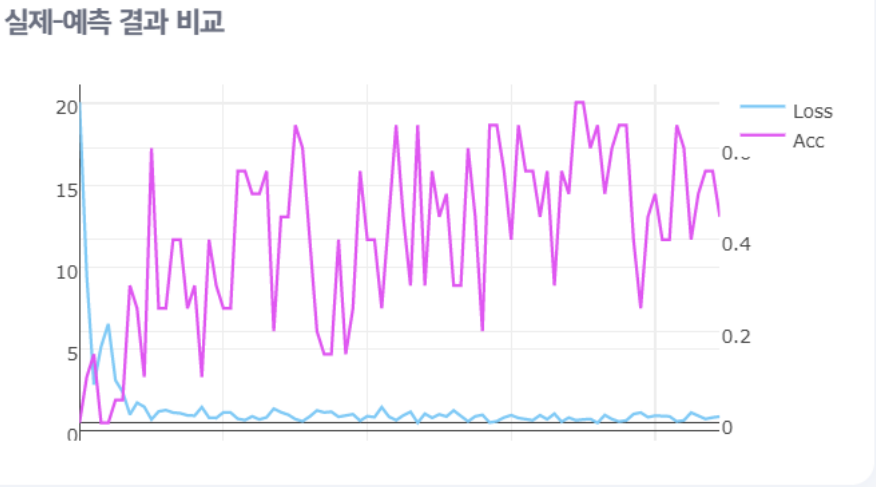
5. Test3 제일 처음 conv2D의 filter수를 8로



평가 데이터의 비율은 2~98% 사이여야 합니다.

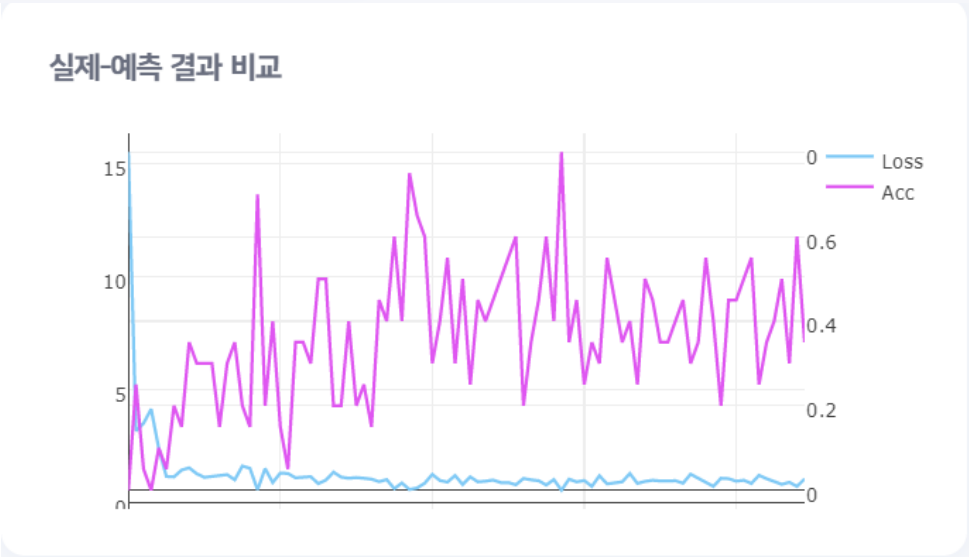


5. Test2-Test3 비교 (epoch는 10으로 동일)



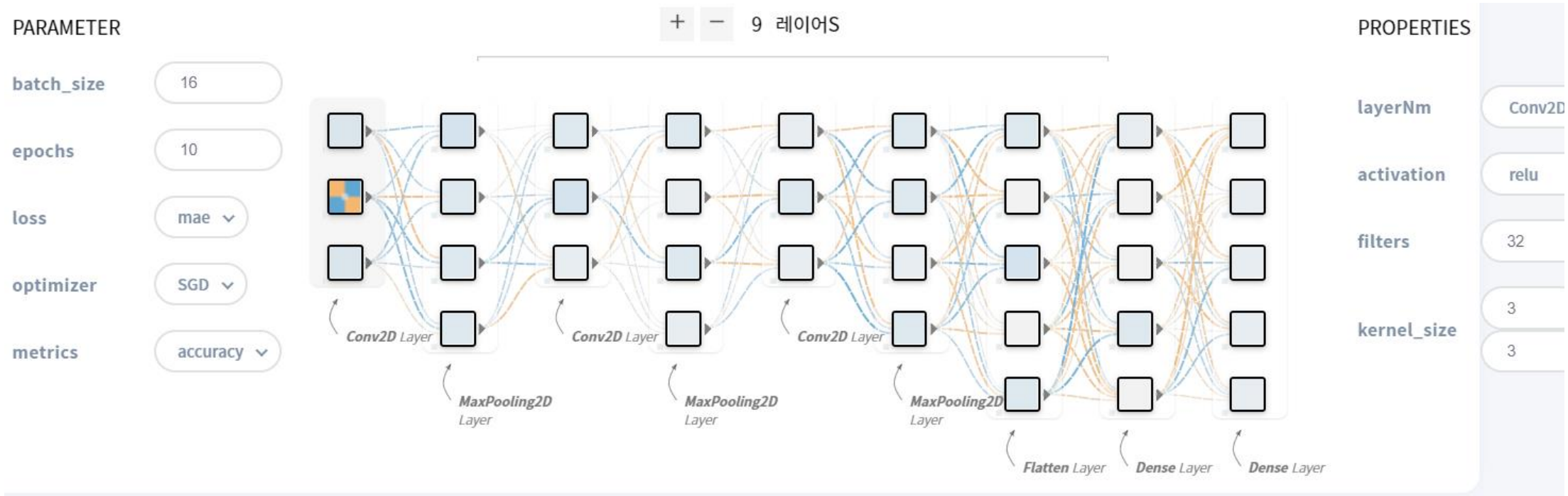
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

좌 – test2 – filter 32개
우 – test3 – filter 8개
정확도가 조금
낮아졌습니다.



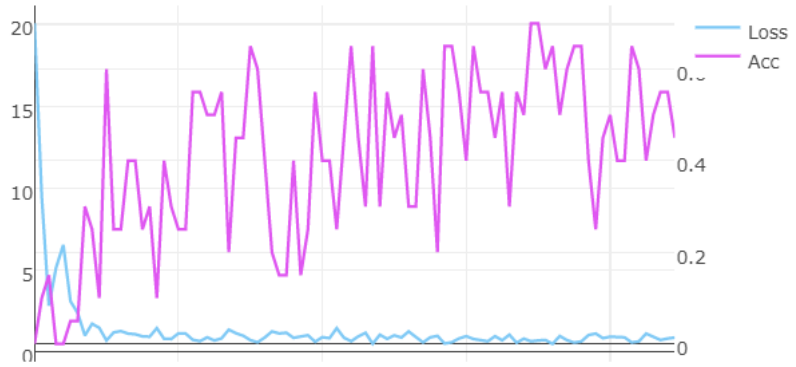
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.88	0.51	0.96	0.48
▸	6	실행완료	0.93	0.44	0.86	0.51
▸	7	실행완료	0.92	0.44	1	0.37
▸	8	실행완료	0.95	0.41	1.04	0.39
▸	9	실행완료	0.99	0.39	1.01	0.47
▸	10	실행완료	0.91	0.44	0.8	0.41

5. Test4 batch size를 16으로 줄임



5. Test2-Test4 비교

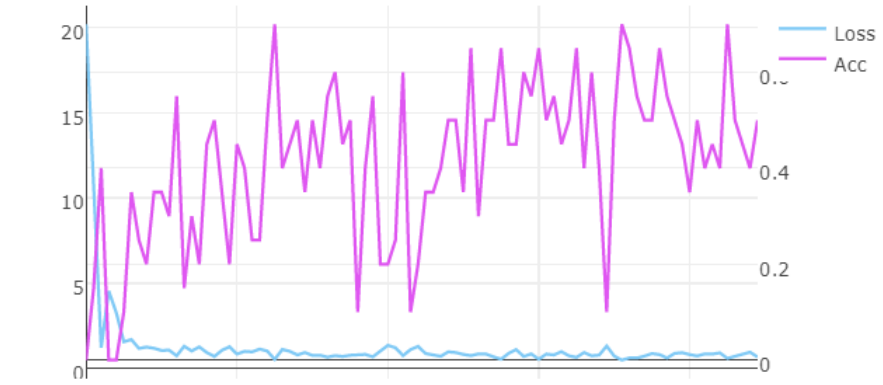
실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

별 차이 없음

실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	1.03	0.32	0.99	0.17
▸	6	실행완료	0.88	0.41	0.88	0.47
▸	7	실행완료	0.8	0.53	0.83	0.5
▸	8	실행완료	0.84	0.48	0.81	0.53
▸	9	실행완료	0.78	0.52	0.86	0.47
▸	10	실행완료	0.8	0.49	0.85	0.46

5. Test 5 Dense Layer 2개를 추가

PARAMETER

batch_size 128

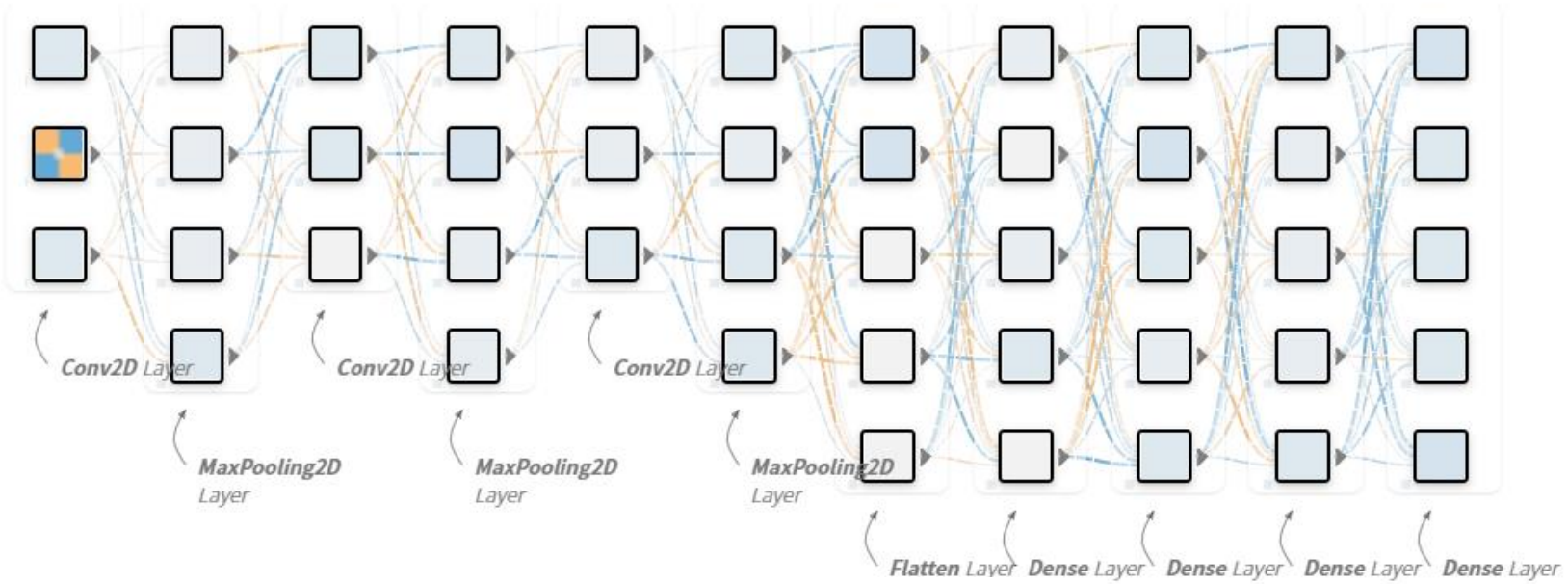
epochs 10

loss mae

optimizer SGD

metrics accuracy

+ − 11 레이어S



PROPERTIES

layerNm Conv2D

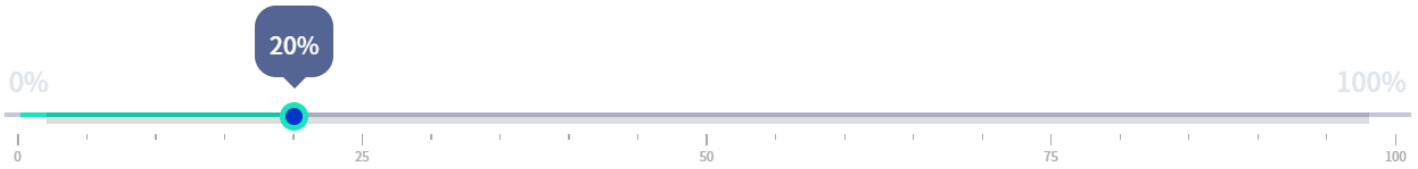
activation relu

filters 1

kernel_size 3 -X 3 -Y

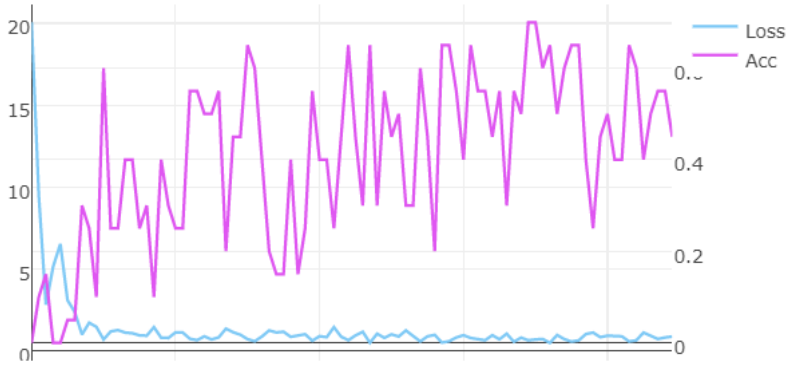
평가 데이터 분할 비율

평가 데이터의 비율은 2~98% 사이여야 합니다.



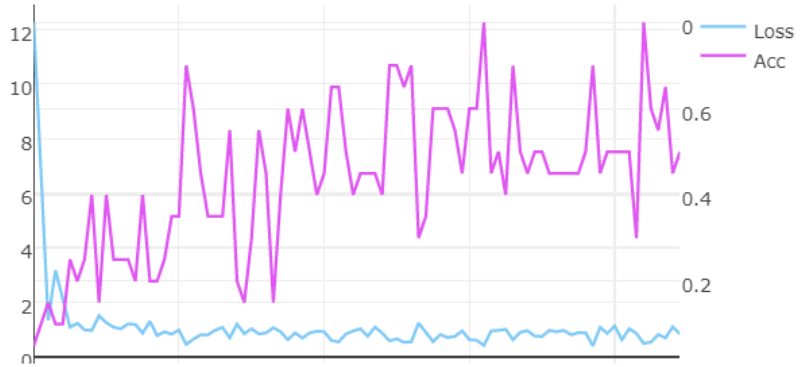
5. Test2-Test5 비교

실제-예측 결과 비교



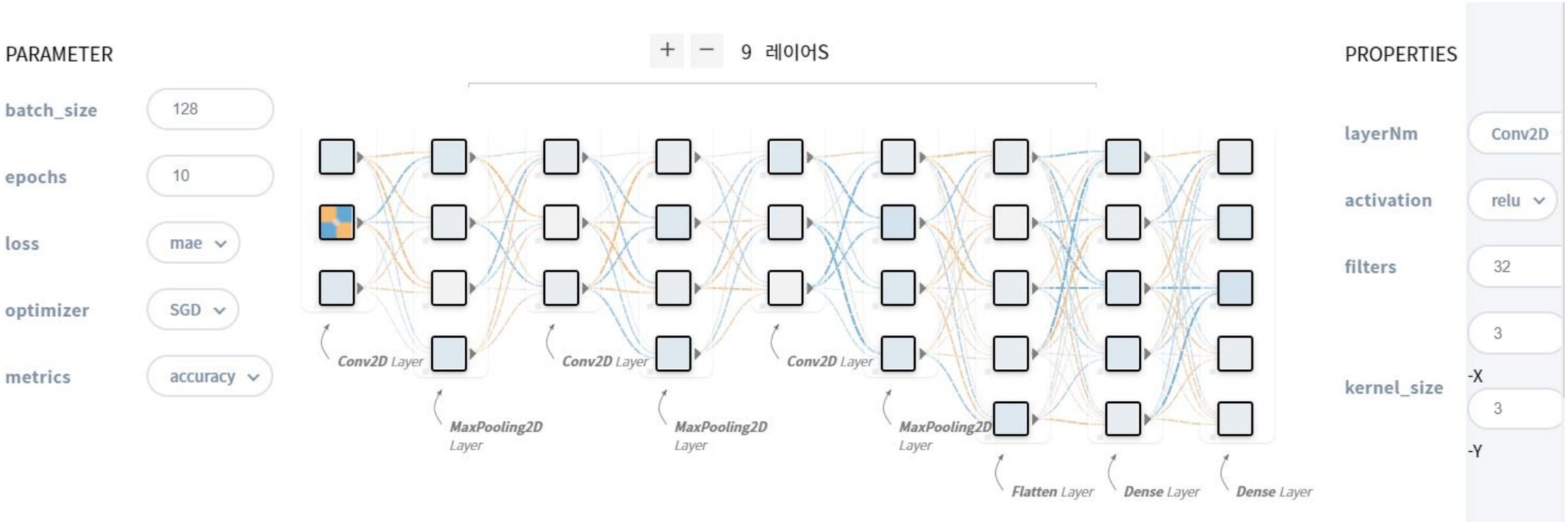
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

실제-예측 결과 비교



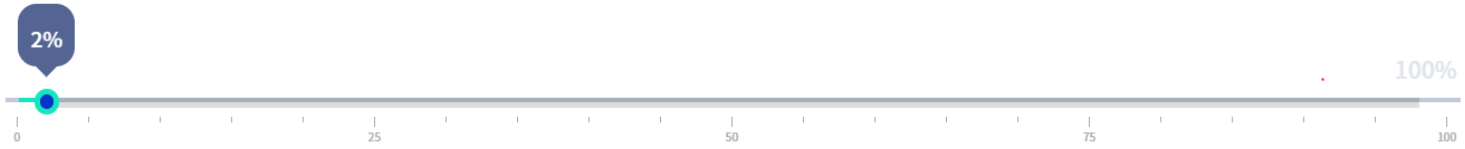
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.8	0.51	1.09	0.3
▸	6	실행완료	0.81	0.5	0.88	0.3
▸	7	실행완료	0.69	0.56	0.79	0.44
▸	8	실행완료	0.88	0.48	0.86	0.52
▸	9	실행완료	0.88	0.49	0.61	0.59
▸	10	실행완료	0.78	0.53	0.78	0.49

5. Test6 평가 데이터 2%



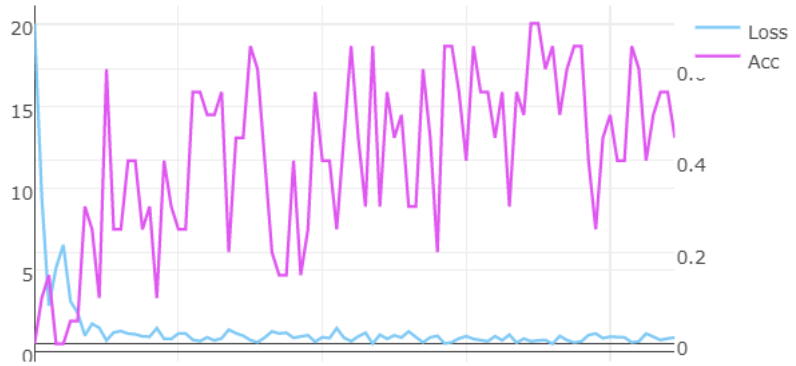
평가 데이터 분할 비율

평가 데이터의 비율은 2~98% 사이여야 합니다.



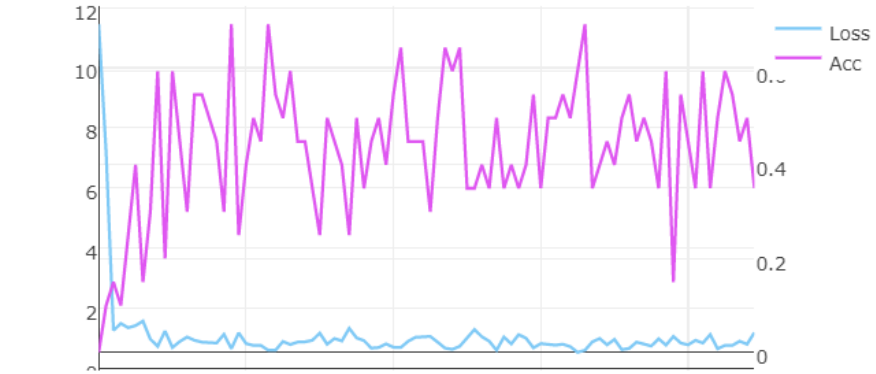
5. Test2 - Test6 결과

실제-예측 결과 비교



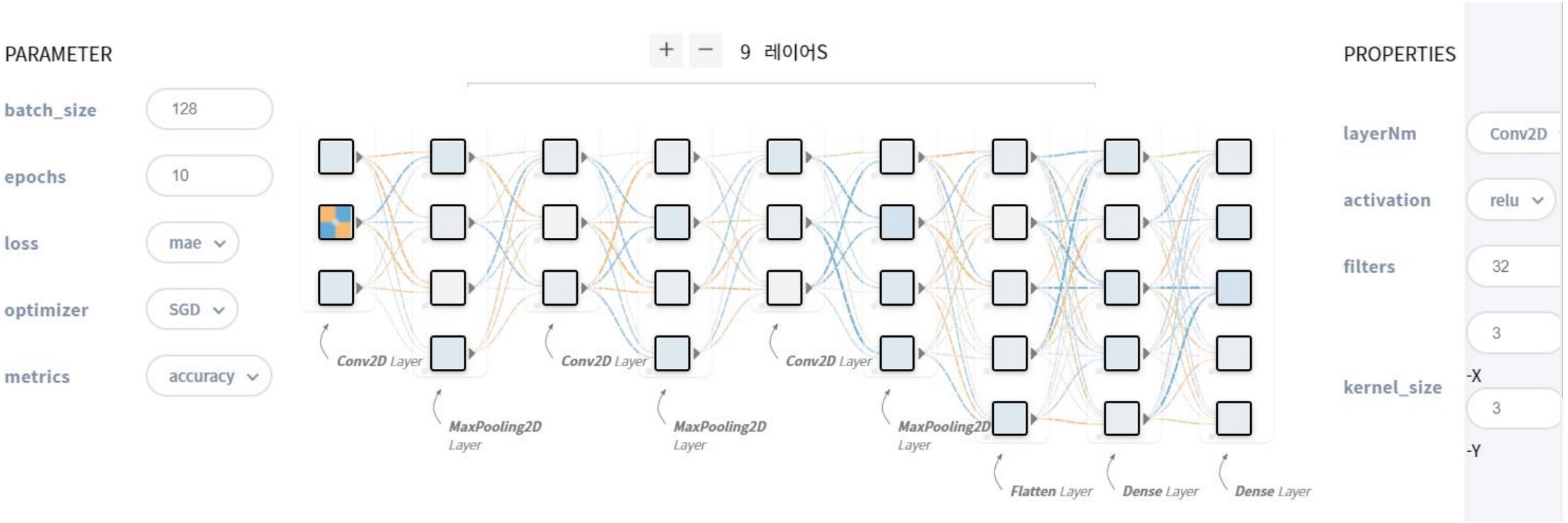
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

실제-예측 결과 비교



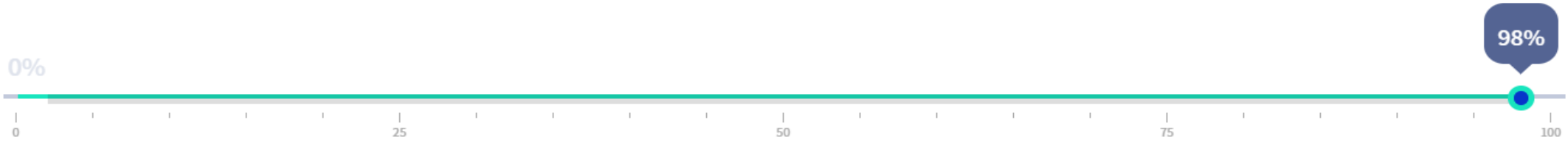
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.81	0.47	0.9	0.42
▸	6	실행완료	0.92	0.45	0.98	0.31
▸	7	실행완료	0.88	0.41	0.78	0.52
▸	8	실행완료	0.77	0.5	0.89	0.44
▸	9	실행완료	0.83	0.44	0.82	0.5
▸	10	실행완료	0.88	0.48	0.82	0.44

5. Test7 평가 데이터 98%



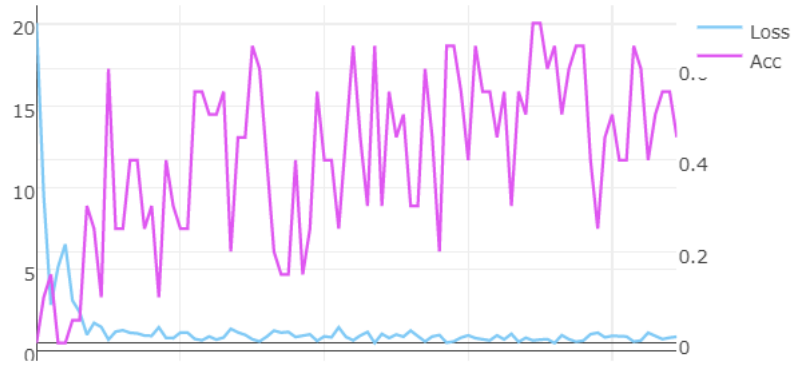
평가 데이터 분할 비율

평가 데이터의 비율은 2~98% 사이여야 합니다.



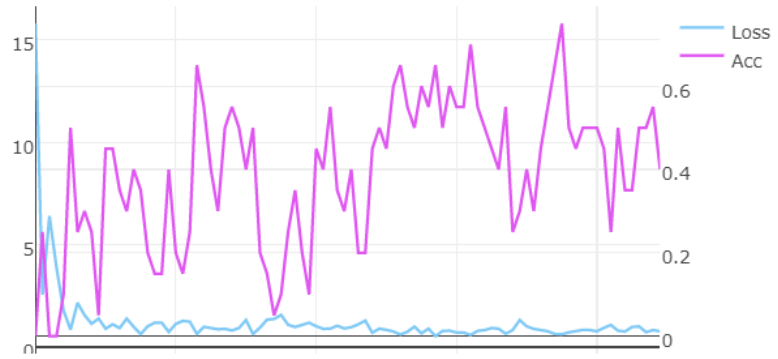
5. Test2 - Test7 결과

실제-예측 결과 비교



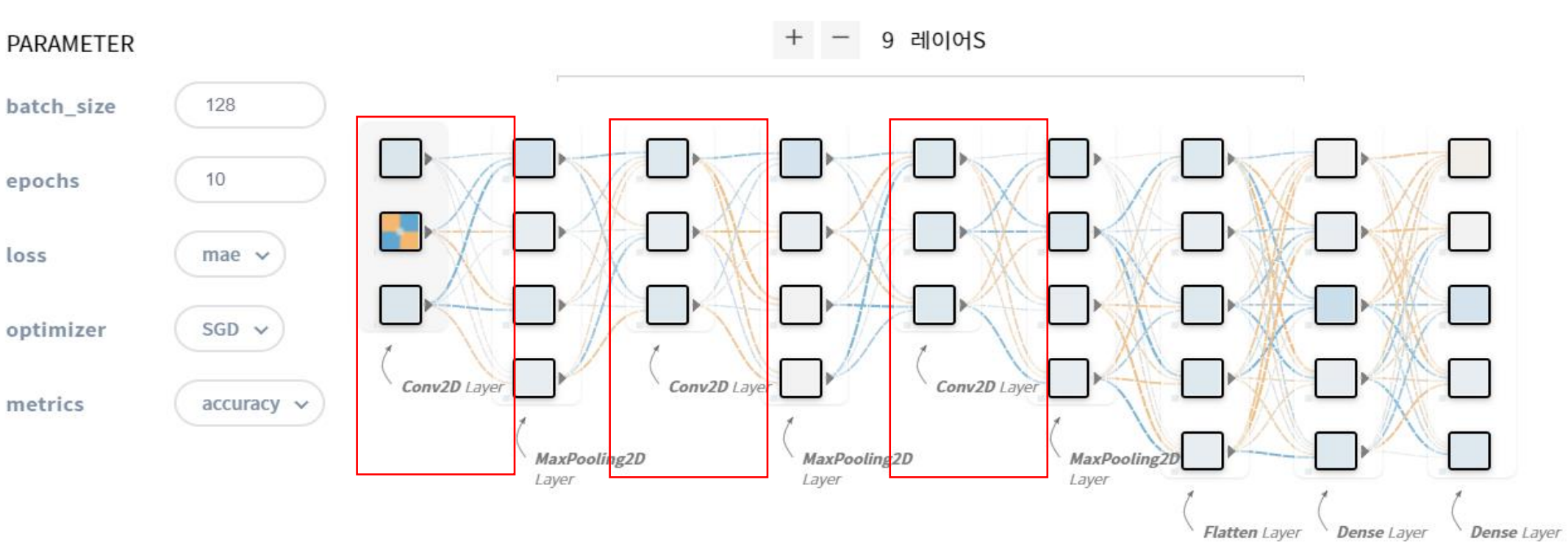
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

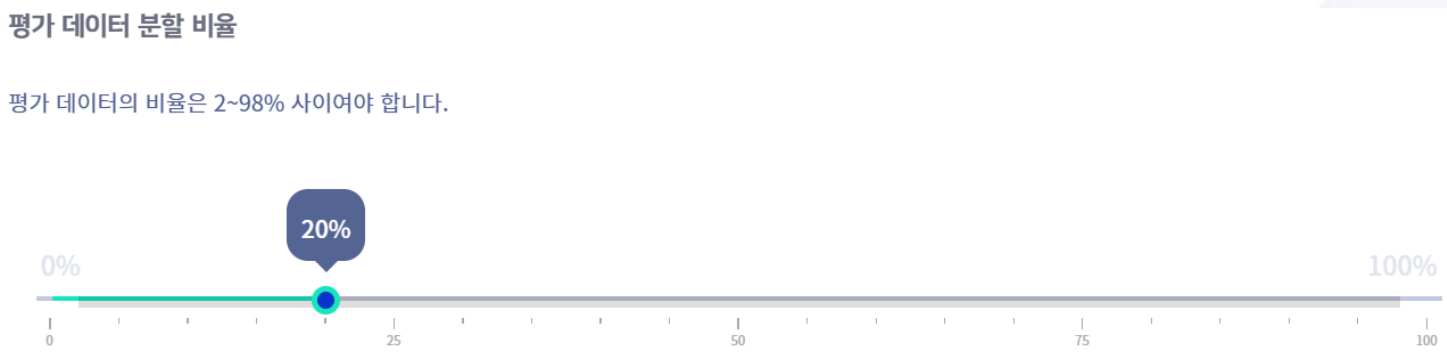
실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.99	0.34	1	0.38
▸	6	실행완료	0.91	0.44	0.82	0.54
▸	7	실행완료	0.75	0.57	0.67	0.59
▸	8	실행완료	0.91	0.41	1.06	0.27
▸	9	실행완료	0.79	0.51	0.8	0.54
▸	10	실행완료	0.88	0.43	0.89	0.49

5. Test8 – Sigmoid 사용

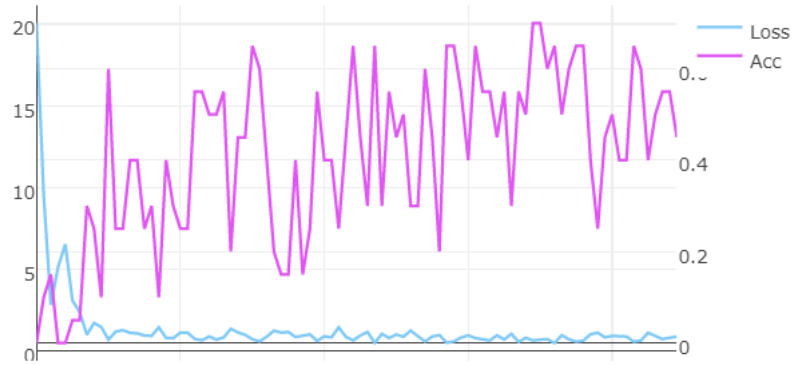




- Conv2D의 Activation 함수를 sigmoid로 변경,
- 평가 데이터는 20%로 동일

5. Test2 – Test8 결과

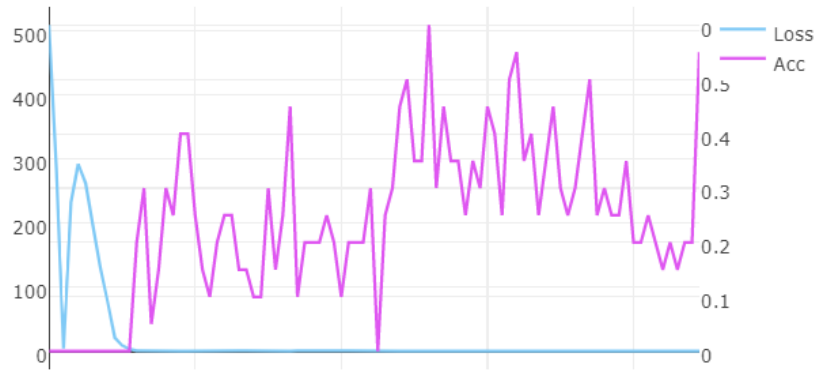
실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

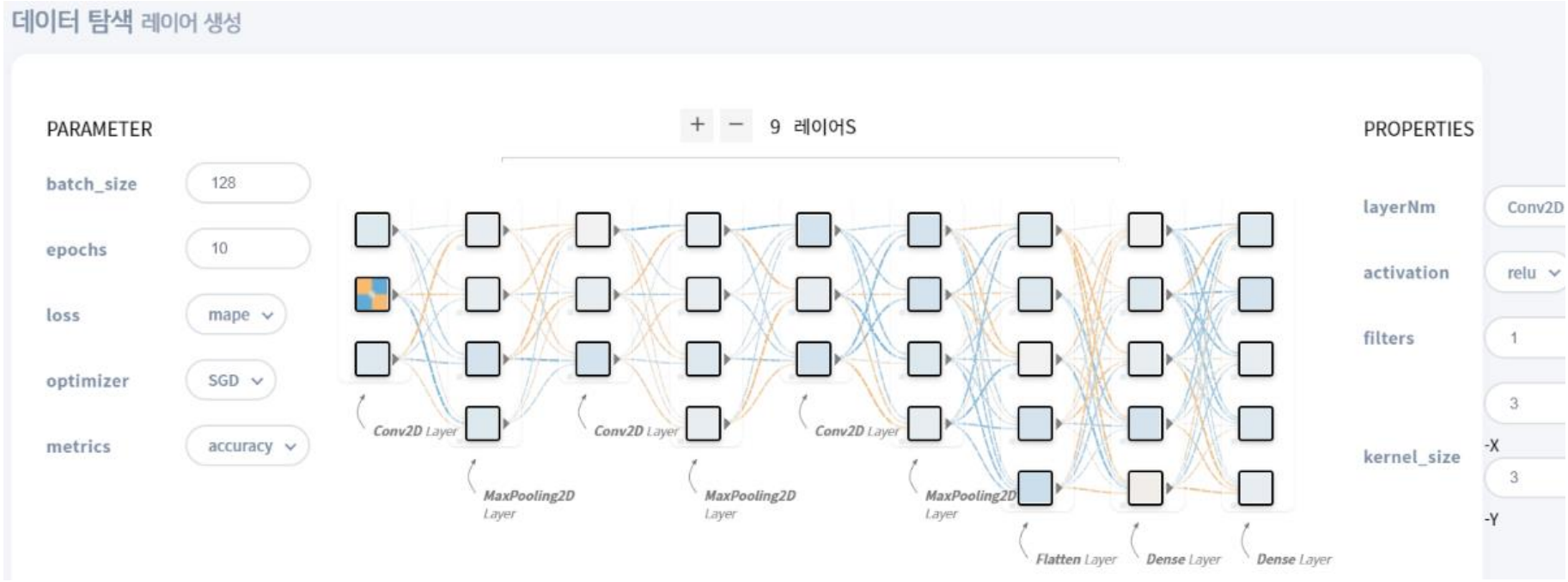
- 정확도가 더 낮게 나온다.

실제-예측 결과 비교



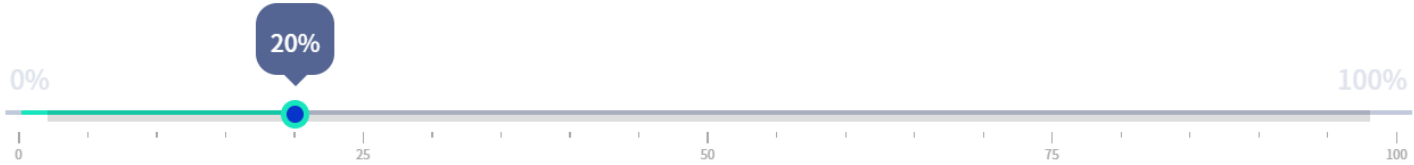
	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	1.15	0.2	1.15	0.23
▸	6	실행완료	1.05	0.31	1	0.38
▸	7	실행완료	1.05	0.36	1.01	0.37
▸	8	실행완료	0.96	0.36	0.95	0.37
▸	9	실행완료	1.03	0.32	1.07	0.09
▸	10	실행완료	1.06	0.22	0.95	0.34

5. Test9 – loss : mape 사용



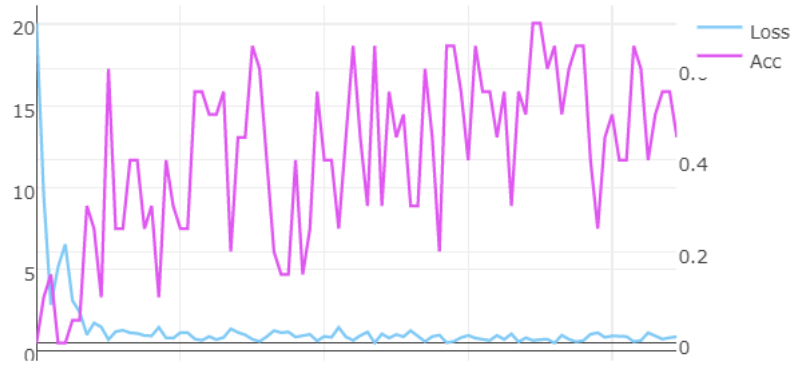
평가 데이터 분할 비율

평가 데이터의 비율은 2~98% 사이여야 합니다.



5. Test2 – Test9 결과

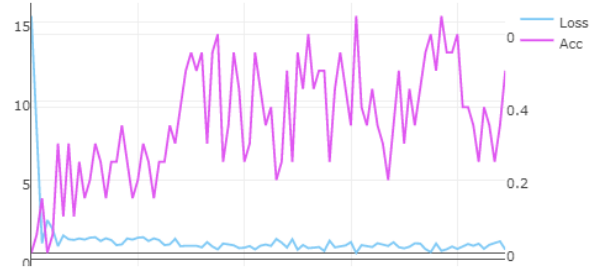
실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.9	0.38	1.03	0.4
▸	6	실행완료	0.93	0.43	0.86	0.53
▸	7	실행완료	0.74	0.54	0.74	0.63
▸	8	실행완료	0.75	0.55	0.78	0.55
▸	9	실행완료	0.79	0.52	0.79	0.49
▸	10	실행완료	0.81	0.52	0.74	0.58

- 정확도가 더 낮게 나온다.

실제-예측 결과 비교



	Epochs	Status	Loss	Acc	Val loss	Val acc
▸	5	실행완료	0.84	0.4	0.93	0.43
▸	6	실행완료	0.91	0.43	0.8	0.51
▸	7	실행완료	0.86	0.44	0.69	0.6
▸	8	실행완료	0.88	0.36	0.77	0.49
▸	9	실행완료	0.77	0.54	0.81	0.48
▸	10	실행완료	0.91	0.38	0.88	0.5

뒤로 결과 다운로드 모델 다운로드