

CNN

(Convolutional Neural Network)

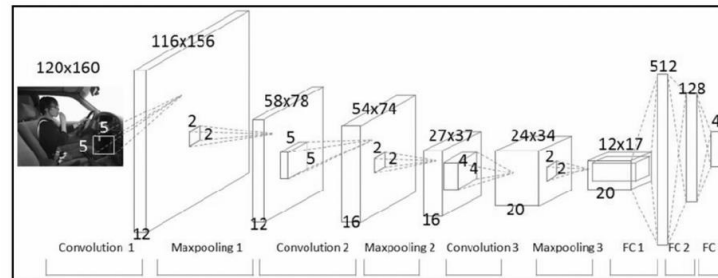


그림 8: 전형적인 CNN, 출처: https://www.researchgate.net/figure/Architecture-of-our-unsupervised-CNN-Network-contains-three-stages-each-of-which_283433254

CNN

합성곱 층(Convolution layer) 과 완전 연결 층(Dense layer)의 차이

❖ 완전 연결 층은 입력 이미지 전역 패턴 학습하지만, 합성곱 층은 입력 이미지 지역 패턴 학습한다.

특징

- 평행이동 불변성: 평행이동 된 지역 패턴은 같은 패턴으로 인식한다. 이는 합성곱 신경망이 효율적으로 지역 패턴 학습하게 한다. 또, 인간 두뇌가 객체 인식하는 방법과 같다.
- 객체를 계층적 인식: 컨브넷은 객체를 계층적으로 인식한다. 각 위치 에지, 질감에서 시작해서 귀, 코, 눈 등 더 상위 개념을 순차.계층적으로 인식해간다. 두뇌가 객체 인식하는 방법과 같다.

CNN

합성곱 연산

- 합성곱 층은 입력 이미지 받아 합성곱 연산 수행하고, 결과 출력한다.
- 합성곱 연산은 입력 이미지 모든 위치에서 지역 패턴들 추출한다.
- 지역패턴에는 에지(Edge), 질감(Texture) 등 포함된다.

CNN

합성곱 연산 과정

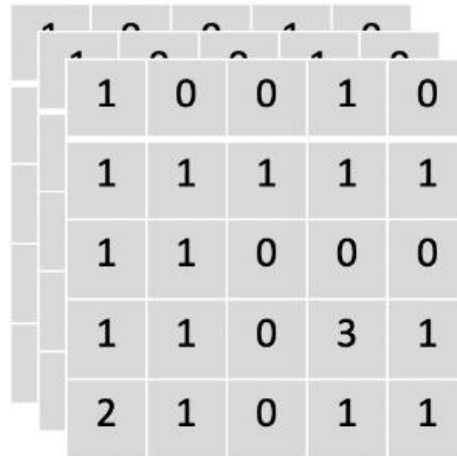
1. 입력 이미지 위를 3×3 또는 5×5 크기 윈도우가 슬라이딩(Sliding) 하며 3×3 또는 5×5 크기 패치(Patch:작은 조각) 추출한다. (윈도우 사이즈 크기 패치 추출한다)
 2. 각 패치와. 에지, 질감 등 지역 특징 담고있는 필터(또는 커널)를 요소별 텐서 곱 연산 한다.
 3. 텐서 곱 연산 결과 모두 합한다. 곧, 2 과정은 패치 각 요소를 가중합 한 것과 같다.
 4. <3> 결과를 특성 맵(Feature Map) 또는 응답 맵(Response Map) 이라고 한다. 입력 이미지 각 위치에 그 필터 패턴이 나타나 있었는지 확인. 응답한 결과다.
 5. 입력 이미지 각 채널(예: RGB 3개) 별로 각각 다른 필터 적용된다.
- 1개 합성곱 층에서 입력 이미지에 대해 여러 개 필터 적용한다(예: 32개, 64개, 128개...).
 - 보통 1개 합성곱 층 필터 개수는 하위 층에서 상위 층 갈 수록 2 제곱 수로 점차 증가시켜 간다.
 - 일반 경우, 합성곱 하위 층에서 상위 층(더 깊은 층) 갈 수록, 각 층의 출력 특성 맵 크기는 줄어든다.

CNN

위 과정 그림으로 그리면 아래와 같다.

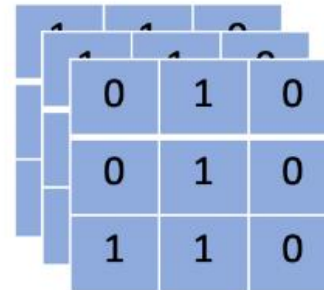
입력 이미지와 필터

RGB 3개 채널로 이루어진 입력 이미지



1	0	0	1	0
1	1	1	1	1
1	1	0	0	0
1	1	0	3	1
2	1	0	1	1

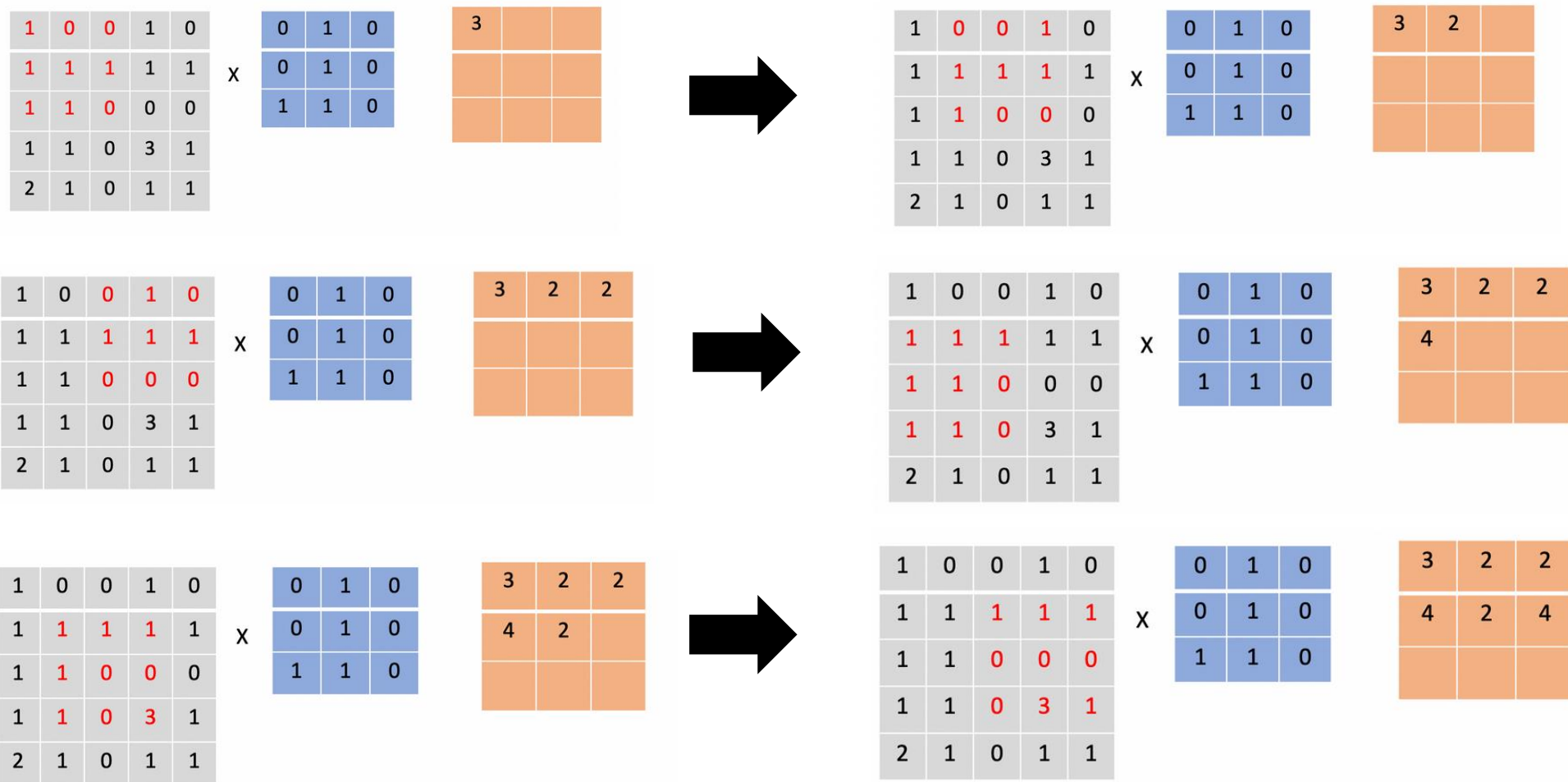
필터 (커널)



0	1	0
0	1	0
1	1	0

CNN

입력 이미지 위를 윈도우가 슬라이딩 하면서 패치 추출, 패치와 필터 합성곱



CNN

1	0	0	1	0		0	1	0	3	2	2
1	1	1	1	1	x	0	1	0	4	2	4
1	1	0	0	0		1	1	0	5		
1	1	0	3	1							
2	1	0	1	1							

1	0	0	1	0		0	1	0	3	2	2
1	1	1	1	1	x	0	1	0	4	2	4
1	1	0	0	0		1	1	0	5	1	
1	1	0	3	1							
2	1	0	1	1							

1	0	0	1	0		0	1	0	3	2	2
1	1	1	1	1	x	0	1	0	4	2	4
1	1	0	0	0		1	1	0	5	1	4
1	1	0	3	1							
2	1	0	1	1							

출력 특성 맵(응답 맵)

3개 채널 합성곱 결과 행렬을 요소별로 합 한 게 필터 1개에 대한 최종 출력 특성 맵 이다.
이게 필터 수 만큼의 차원을 이룬다.

CNN

패딩(Padding)

Zero Padding

- 입력 특성 맵과 출력 특성 맵 크기를 같게 하고 싶으면 입력 특성 맵에 패딩(Padding) 추가하면 된다.
- 입력 특성 맵 가장자리에 적절한 개수 행과 열 추가하는 걸 패딩이라 한다.
- 패딩 자리에 보통 0 넣기 때문에, 제로 패딩 이라고도 한다.

	1	0	0	1	0	
	1	1	1	1	1	
	1	1	0	0	0	
	1	1	0	3	1	
	2	1	0	1	1	

- 빈 부분이 패딩 자리다.
- 위 행렬에 대해 합성곱 하면 출력 특성 맵 크기가 입력 특성 맵과 같아진다 (5*5)

CNN

케라스에서 패딩 사용하기

- Conv2D 층에서 padding 파라미터 설정하면 된다. 'valid' 는 패딩 사용 안함, 'same'은 패딩 사용함 이다.
- 기본 파라미터는 valid (패딩 사용 안함) 이다.

CNN

스트라이드(Stride): 보폭.

정의

연속한 두 윈도우 사이 거리를 스트라이드 라고 한다.

- 스트라이드 값 기본은 1이다.

예) 스트라이드 = 2 인 경우

1	0	0	1	0
1	1	1	1	1
1	1	0	0	0
1	1	0	3	1
2	1	0	1	1

1	0	0
1	1	1
1	1	0

1	0	0	1	0
1	1	1	1	1
1	1	0	0	0
1	1	0	3	1
2	1	0	1	1

1	0	0	0	1	0
1	1	1	1	1	1
1	1	0	0	0	0

CNN

스트라이드(Stride): 보폭.

1	0	0	1	0
1	1	1	1	1
1	1	0	0	0
1	1	0	3	1
2	1	0	1	1

1	0	0
1	1	1
1	1	0
1	1	0
1	1	0
2	1	0

0	1	0
1	1	1
0	0	0

1	0	0	1	0
1	1	1	1	1
1	1	0	0	0
1	1	0	3	1
2	1	0	1	1

1	0	0
1	1	1
1	1	0
1	1	0
1	1	0
2	1	0
0	1	0
1	1	1
0	0	0
0	0	0
0	3	1
0	1	1

CNN

최대 풀링 연산

- 합성곱 층 출력 특성맵 받아 다운샘플링(행렬 크기 줄이기) 하는 연산이다.
- 2×2 윈도우와 스트라이드 2 사용해서, 패치 별로 최대값만 추출한다.

최대 풀링 연산 목적

- 입력을 다운샘플링 해서, 특성 맵 가중치 개수를 줄인다.
- 모델이 공간적 계층 구조 학습하는 걸 돕는다.

CNN

최대 풀링 연산 예시

3	2	2
4	2	4
5	1	4

4	1
1	1

3	2	2
4	2	4
5	1	4

4	4
1	1

3	2	2
4	2	4
5	1	4

4	4
5	1

3	2	2
4	2	4
5	1	4

4	4
5	4

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

MNIST 이미지 데이터 셋



0에서 9까지 10가지로 분류될 수 있는 손 글씨 숫자 이미지 70,000개로 구성된 데이터 셋.
각 손 글씨 숫자 이미지는 28*28 픽셀로 구성되고, 각 픽셀은 다음과 같이 0~255 사이의 숫자 행렬로 표현된다.

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

행렬로 나타낸 MNIST Single data 예시

```
[[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 3 18 18 18 126 136 175 26 166 255 247 127 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 253 253 225 172 253 242 195 64 0 0 0 0]
 [ 0 0 0 0 0 0 0 49 238 253 253 253 253 253 253 253 251 93 82 82 56 39 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 18 219 253 253 253 253 253 198 182 247 241 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 14 1154 253 90 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 11 190 253 70 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 35 241 225 160 108 1 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 45 186 253 253 150 27 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 249 253 249 64 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 46 130 183 253 253 207 2 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 24 114 221 253 253 253 253 201 78 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 18 171 219 253 253 253 253 195 80 9 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 136 253 253 253 212 135 132 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

총 70,000 개 손 글씨 데이터 중

- 60,000 개 이미지를 CNN 훈련(Training) 에 사용하고,
- 10,000 개 이미지를 일반화 성능 테스트(Test) 위해 사용한다.

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

```
1 from keras import layers
2 from keras import models
3
4 # 간단한 컨브넷 작성
5 # 이미지 특징 추출 층(합성곱 기반 층)
6 model = models.Sequential()
7 # 필터 수, 패치 사이즈(3x3), 요소별 적용할 활성화 함수, 입력 특성 맵 사이즈
8 model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))
9 model.add(layers.MaxPooling2D((2,2)))
10 # activation='relu': 음수, 0은 모두 0으로 만들고, 양수 값만 남긴다.
11 model.add(layers.Conv2D(64, (3,3), activation='relu'))
12 model.add(layers.MaxPooling2D((2,2)))
13 model.add(layers.Conv2D(64, (3,3), activation='relu'))
14 model.add(layers.Flatten()) # 특성공학 결과물 1차원 텐서(벡터)로 변환하는 층
15
16 # 완전 연결 분류기
17 model.add(layers.Dense(64, activation='relu'))
18 model.add(layers.Dense(10, activation='softmax')) # 출력층: 최상위층, 분류 결과물 확률 꼴로 변환.
19
20 # 모델 설계 결과 요약
21 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

- 합성곱 층과 최대 풀링 층 교차해서 쌓은. 전형적인 컨브넷 구조다.
- 합성곱 층 파라미터 별 의미: 입력 특성 맵에 적용할 필터 수(32), 윈도우 크기(3×3), 활성화 함수(렐루함수), 입력 특성 맵 크기.
- 최대 풀링 층 파라미터 의미: 윈도우 크기(2×2)
- Flatten() 층: 특성공학 결과를 1차원 텐서(벡터)로 변환하는 층. Dense 층에 넣기 위함이다.

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

MNIST 데이터 로드 및 전처리

```
1 # MNIST 숫자 이미지 합성곱 신경망으로 분류
2 from keras.datasets import mnist
3 from tensorflow.keras.utils import to_categorical
4
5 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
6
7 train_images = train_images.reshape((60000, 28, 28, 1)) # 6만개 배치, 높이 28, 너비 28, 채널 1 사이즈로 크기 조정
8 train_images = train_images.astype('float32') / 255
9
10 test_images = test_images.reshape((10000, 28, 28, 1)) # 1만개 배치, 높이 28, 너비 28, 채널 1 사이즈로 크기 조정
11 test_images = test_images.astype('float32') / 255 # 전부 부동소수점 실수로 변환 + 1/255 로 스케일 조정
12
13 train_labels = to_categorical(train_labels) # train_label 들을 모두 원핫 인코딩 벡터로 변환 # 분류 결과와 크로스엔트로피 비교하기 위함
14 test_labels = to_categorical(test_labels)
```

CNN

간단한 컨브넷 만들기 - MNIST 이미지 분류하는 간단한 컨브넷 만들기

모델 컴파일(학습 준비시키기)

```
16 # 모델 컴파일
17 model.compile(
18     optimizer = 'rmsprop',
19     loss = 'categorical_crossentropy',
20     metrics = ['accuracy']
21 )
22
```

Optimizer, loss function, 성능 측정 metric 지정

모델 학습시키기

```
23 # 모델 학습
24 model.fit(
25     train_images,
26     train_labels,
27     epochs = 5,
28     batch_size=64,
29     validation_data = (test_images, test_labels)
30 )
31
32 test_loss, test_acc = model.evaluate(test_images, test_labels) ; print(test_acc)
33 # 98% 정도 정확도 기록함.
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 [=====] - 0s 0us/step

Epoch 1/5

938/938 [=====] - 62s 65ms/step - loss: 0.1736 - accuracy: 0.9457 - val_loss: 0.1001 - val_accuracy: 0.9663

Epoch 2/5

938/938 [=====] - 60s 63ms/step - loss: 0.0484 - accuracy: 0.9849 - val_loss: 0.0332 - val_accuracy: 0.9888

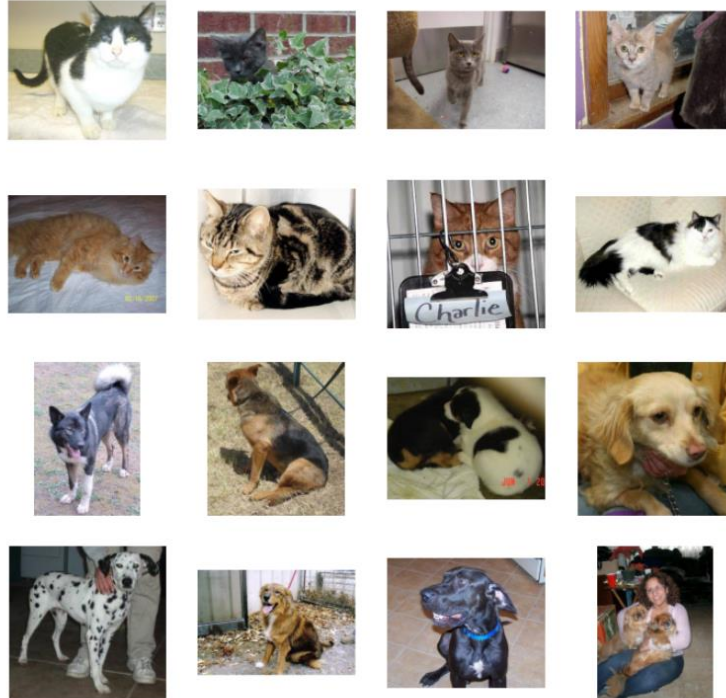
Epoch 3/5

59/938 [>.....] - ETA: 1:00 - loss: 0.0358 - accuracy: 0.9862

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

Kaggle 개 vs 고양이 데이터 소개



- Total training cat image: 1000 개
- Total training dog image: 1000 개
- Total validation cat image: 500 개
- Total validation dog image: 500 개

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

Kaggle 개 vs 고양이 데이터 내려받기

- PLATO 'dog_and_cat' 데이터셋 다운로드 받기
- 자신의 컴퓨터 바탕화면에 압축 풀기

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

개 vs 고양이 분류 컨브넷 정의

```
1 # 네트워크 구성
2
3 from keras import layers
4 from keras import models
5
6 model = models.Sequential()
7 # 입력 특성 맵에 적용 할 필터 수: 32, 윈도우 사이즈, 활성화함수, 입력 데이터 규격: 150*150, RGB 3 채널
8 model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)))
9 model.add(layers.MaxPooling2D((2,2))) # 최대 풀링 연산 적용할 윈도우 사이즈 - 다운샘플링(크기 축소)
10 model.add(layers.Conv2D(64, (3,3), activation='relu')) # 입력 특성 맵에 적용할 필터 수:64, 윈도우 사이즈, 활성화 함수
11 model.add(layers.MaxPooling2D((2,2))) # 윈도우 사이즈
12 model.add(layers.Conv2D(128, (3,3), activation='relu')) # 필터 수: 128개, 윈도우 사이즈
13 model.add(layers.MaxPooling2D((2,2))) # 윈도우 사이즈
14 model.add(layers.Conv2D(128, (3,3), activation='relu')) # 필터 수: 128개, 윈도우 사이즈
15 model.add(layers.MaxPooling2D(2,2)) # 윈도우 사이즈
16 # 여기까지 합성곱 기반 층(지역 패턴 추출 층)
17
18 # 여기서부터 완전 연결 층(전역 패턴 추출, 분류기)
19 model.add(layers.Flatten()) # 1차원 텐서(벡터)로 변환
20 model.add(layers.Dense(512, activation='relu')) # 512차원 벡터공간에 투영
21 model.add(layers.Dense(1, activation='sigmoid'))
22
23 model.summary()
24 # 1. 150*150 입력 이미지에서 3*3 윈도우 슬라이딩하면서 3*3 패치 추출 -> 32개 필터에 대해 합성곱 -> 148*148*32
25 # 2. 2*2 윈도우 1의 출력 특성 맵에 적용해서 패치 구역 별 최대값만 추출 -> 출력 특성 맵 크기 절반으로 줄어든다 -> 74*74*32
26 # 3. 2의 출력 특성 맵에서 다시 3*3 패치 추출 -> 64개 필터에 대해 합성곱 -> 72*72*64
27 # 4. 2 처럼 최대 풀링 연산 3 출력에 적용 -> 출력 특성 맵 크기 절반으로 줄어든다 -> 36*36*64
28 # 5. 3*3 패치, 128개 필터에 대해 합성곱 -> 34*34*128
29 # 6. 최대 풀링 연산 적용 -> 17*17*128
30 # 7. 3*3 패치, 128개 필터에 대해 합성곱 -> 15*15*128
31 # 8. 최대 풀링 연산 적용 -> 7*7*128
32 # 9. 완전 연결 분류기 주입 위해 1차원 텐서(벡터)로 변환하는 층
33 # 10. 512차원 벡터공간에 투영
34 # 11. 1차원 벡터공간으로 차원축소 후 시그모이드 함수 적용
35
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

```
1 # 네트워크 구성
2
3 from keras import layers
4 from keras import models
5
6 model = models.Sequential()
7 # 입력 특성 맵에 적용 할 필터 수: 32, 윈도우 사이즈, 활성화함수, 입력 데이터 규격: 150*150, RGB 3 채널
8 model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)))
9 model.add(layers.MaxPooling2D((2,2))) # 최대 풀링 연산 적용할 윈도우 사이즈 - 다운샘플링(크기 축소)
10 model.add(layers.Conv2D(64, (3,3), activation='relu')) # 입력 특성 맵에 적용할 필터 수:64, 윈도우 사이즈, 활성화 함수
11 model.add(layers.MaxPooling2D((2,2))) # 윈도우 사이즈
12 model.add(layers.Conv2D(128, (3,3), activation='relu')) # 필터 수: 128개, 윈도우 사이즈
13 model.add(layers.MaxPooling2D((2,2))) # 윈도우 사이즈
14 model.add(layers.Conv2D(128, (3,3), activation='relu')) # 필터 수: 128개, 윈도우 사이즈
15 model.add(layers.MaxPooling2D(2,2)) # 윈도우 사이즈
16 # 여기까지 합성곱 기반 층(지역 패턴 추출 층)
```


CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

```
17
18 # 여기서부터 완전 연결 층(전역 패턴 추출, 분류기)
19 model.add(layers.Flatten()) # 1차원 텐서(벡터)로 변환
20 model.add(layers.Dense(512, activation='relu')) # 512차원 벡터공간에 투영
21 model.add(layers.Dense(1, activation='sigmoid'))
22
23 model.summary()
24 # 1. 150*150 입력 이미지에서 3*3 윈도우 슬라이딩하면서 3*3 패치 추출 -> 32개 필터에 대해 합성곱 -> 148*148*32
25 # 2. 2*2 윈도우 1의 출력 특성 맵에 적용해서 패치 구역 별 최댓값만 추출 -> 출력 특성 맵 크기 절반으로 줄어든다 -> 74*74*32
26 # 3. 2의 출력 특성 맵에서 다시 3*3 패치 추출 -> 64개 필터에 대해 합성곱 -> 72*72*64
27 # 4. 2 처럼 최대 풀링 연산 3 출력에 적용 -> 출력 특성 맵 크기 절반으로 줄어든다 -> 36*36*64
28 # 5. 3*3 패치, 128개 필터에 대해 합성곱 -> 34*34*128
29 # 6. 최대 풀링 연산 적용 -> 17*17*128
30 # 7. 3*3 패치, 128개 필터에 대해 합성곱 -> 15*15*128
31 # 8. 최대 풀링 연산 적용 -> 7*7*128
32 # 9. 완전 연결 분류기 주입 위해 1차원 텐서(벡터)로 변환하는 층
33 # 10. 512차원 벡터공간에 투영
34 # 11. 1차원 벡터공간으로 차원축소 후 시그모이드 함수 적용
35
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_8 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_9 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_8 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_10 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_9 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_2 (Flatten)	(None, 6272)	0
dense_4 (Dense)	(None, 512)	3211776
dense_5 (Dense)	(None, 1)	513

=====
Total params: 3,453,121

Trainable params: 3,453,121

Non-trainable params: 0

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

모델 컴파일

```
1  # 모델 컴파일
2
3  from keras import optimizers
4
5  model.compile(
6      loss = 'binary_crossentropy',
7      optimizer = optimizers.adam_v2.Adam(learning_rate=0.001),
8      metrics=['acc']
9  )
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

데이터 전처리

```
1  # 데이터 전처리
2
3  from keras.preprocessing.image import ImageDataGenerator
4
5  train_datagen = ImageDataGenerator(rescale=1./255) # 스케일 1/255 로 조정 , 부동소수점 형태로 변환
6  test_datagen = ImageDataGenerator(rescale=1./255) # 스케일 조정
7
8  train_generator = train_datagen.flow_from_directory(
9      '/Users/kibeomkim/Desktop/cats_and_dogs_small/train',
10     target_size=(150, 150), # 네트워크 입력 규격에 맞게 크기 변환
11     batch_size=20, # 1에폭 동안 투입 할 데이터 묶음
12     class_mode = 'binary' # 데이터가 이진 레이블임.
13 )
14
15 valid_generator = test_datagen.flow_from_directory(
16     '/Users/kibeomkim/Desktop/cats_and_dogs_small/test',
17     target_size=(150,150),
18     batch_size=20,
19     class_mode='binary'
20 )
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

모델 훈련

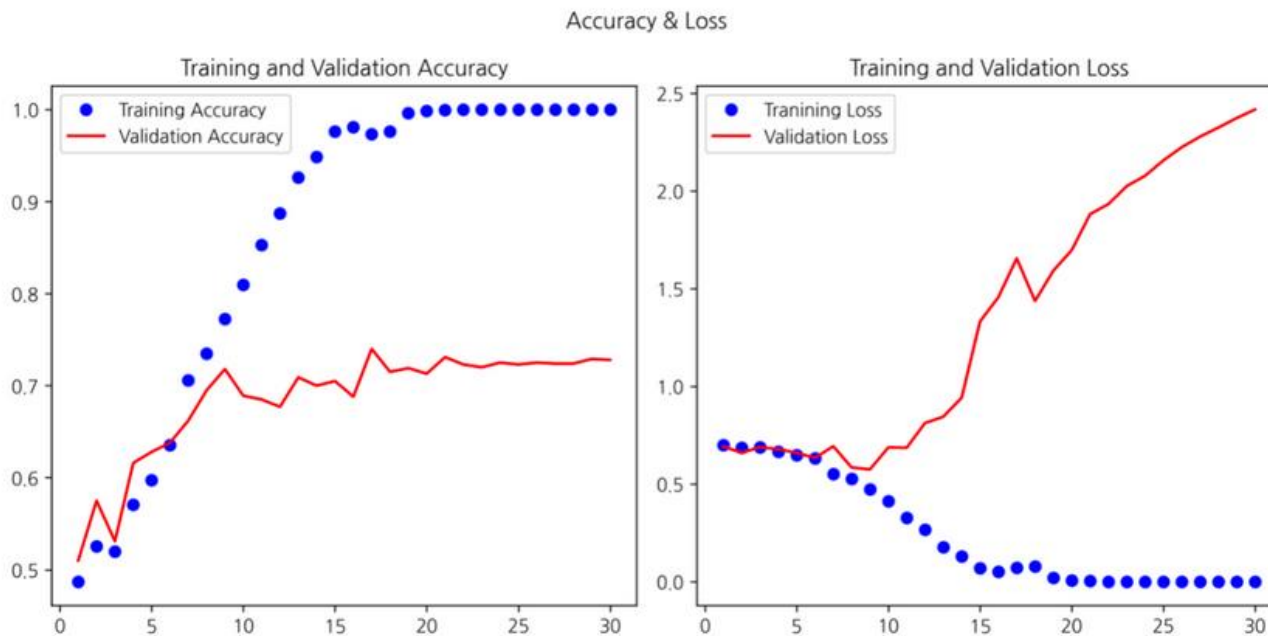
```
22  # 모델 훈련
23
24  history = model.fit_generator(
25      train_generator,
26      steps_per_epoch= 100, # 20*100 = 총 훈련 데이터 갯수
27      epochs = 30 ,
28      validation_data = valid_generator,
29      validation_steps = 50
30  )
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

훈련 및 검증 정확도, 훈련 및 검증 손실 시각화

```
1  acc = history.history['acc']
2  val_acc = history.history['val_acc']
3  loss = history.history['loss']
4  val_loss = history.history['val_loss']
5
6  epochs = range(1, len(acc) + 1)
7
8  plt.figure(figsize=(10,5))
9
10 plt.subplot(1,2,1)
11 plt.plot(epochs, acc, 'bo', label='Training Accuracy')
12 plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
13 plt.title('Training and Validation Accuracy')
14 plt.legend()
15
16 plt.subplot(1,2,2)
17 plt.plot(epochs, loss, 'bo', label='Tranining Loss')
18 plt.plot(epochs, val_loss, 'r', label='Validation Loss')
19 plt.title('Training and Validation Loss')
20 plt.legend()
21
22 plt.suptitle('Accuracy & Loss')
23 plt.tight_layout()
24
25 plt.show()
```



- 과대적합 억제하는 가장 좋은 방법 = 훈련 데이터 수 늘리는 거다.
- 데이터 증식 사용해 과대적합 억제해보자.

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

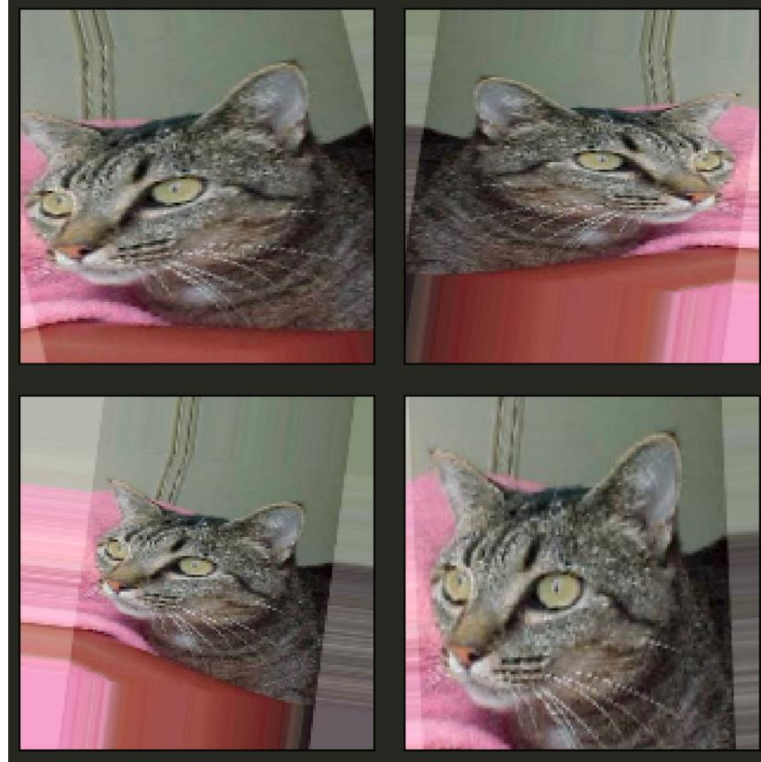
데이터 증식 - 데이터 수 인위적으로 부풀려서 과대적합(overfitting) 억제

```
1 # 데이터 증식
2
3 datagen = ImageDataGenerator(
4     rotation_range=30, # 회전을 몇 도 시킬 건가
5     width_shift_range=0.1, # 수평으로 평행이동 정도
6     height_shift_range=0.1, # 수직으로 평행이동 정도
7     shear_range=0.2, # y축 방향으로 각도 증가
8     zoom_range=0.5, # 확대/축소 범위
9     horizontal_flip=True, # 좌우 대칭시킨다
10    fill_mode='nearest'
11 )
12
13 # 데이터 증식 결과 시각화해서 살펴보기
14 from keras.preprocessing import image
15
16 fnames = sorted([os.path.join('/Users/kibeomkim/Desktop/cats_and_dogs_small/train/cats', fname) for fname in os.listdir('/Users/kibeomkim/Desktop/cats_and_dogs_small/train/cats')])
17 img_path = fnames[7]
18
19 img = image.load_img(img_path, target_size = (150,150)) # 이미지 읽어오기, 크기 150*150으로 변환
20
21 x = image.img_to_array(img) # (150,150,3) 크기 넘파이 배열(텐서)로 변환
22 x = x.reshape((1,)+x.shape) # (1,150,150,3) 으로 변환 (배치 차원 추가)
23
24 plt.figure(figsize=(5,5))
25 i = 1
26 for batch in datagen.flow(x, batch_size=1) :
27     plt.subplot(2,2,i) # i번째 이미지
28     imgplot = plt.imshow(image.array_to_img(batch[0]))
29     plt.xticks([])
30     plt.yticks([])
31     i += 1
32     if i == 5 : break
33 plt.tight_layout()
34 plt.show()
35
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

데이터 증식 결과 관찰하기



- 데이터 증식 적용하더라도, 애초에 훈련 데이터 수가 적기 때문에 과대적합 억제하는 데 충분치 않을 수 있다.
- 모델에 드롭아웃 추가해서 과대적합을 좀 더 억제해 보자.

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

컨브넷에 드롭아웃 층 추가해서 과대적합 억제하기

```
1 # 드롭아웃 포함한 새로운 컨브넷 정의
2
3 from keras import models
4 from keras import layers
5 from keras import optimizers
6
7 model = models.Sequential()
8 # 합성곱 기반 층
9 model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)))
10 model.add(layers.MaxPooling2D((2,2)))
11 model.add(layers.Conv2D(64, (3,3), activation='relu'))
12 model.add(layers.MaxPooling2D((2,2)))
13 model.add(layers.Conv2D(128, (3,3), activation='relu'))
14 model.add(layers.MaxPooling2D((2,2)))
15 model.add(layers.Conv2D(128, (3,3), activation='relu'))
16 model.add(layers.MaxPooling2D((2,2)))
17 model.add(layers.Flatten())
18 model.add(layers.Dropout(0.5))    랜덤하게 절반만큼 특성 값 0으로 만든다(드롭아웃)
19
20 # 완전 연결 분류기
21 model.add(layers.Dense(512, activation='relu'))
22 model.add(layers.Dense(1, activation='sigmoid'))
23
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

컨브넷에 드롭아웃 층 추가해서 과대적합 억제하기 - 모델 컴파일 및 데이터 준비하기

```
--
24 # 모델 컴파일
25 model.compile(
26     loss = 'binary_crossentropy',
27     metrics = ['acc'],
28     optimizer = optimizers.adam_v2.Adam(lr = 0.001)
29 )
30
31 # 데이터 증식 & 전처리
32
33 train_datagen = ImageDataGenerator(
34     rescale = 1./255,
35     rotation_range = 40,
36     width_shift_range= 0.1,
37     height_shift_range=0.1,
38     shear_range = 0.4,
39     zoom_range= 0.5,
40     horizontal_flip=True
41 )
42
43 test_datagen = ImageDataGenerator(
44     rescale=1./255
45 )
46
47 train_generator = train_datagen.flow_from_directory(
48     '/Users/kibeomkim/Desktop/cats_and_dogs_small/train',
49     target_size=(150,150),
50     batch_size= 20,
51     class_mode='binary'
52 )
53
54 valid_generator = test_datagen.flow_from_directory(
55     '/Users/kibeomkim/Desktop/cats_and_dogs_small/test',
56     target_size = (150,150),
57     batch_size=20,
58     class_mode = 'binary'
59 )
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

모델 훈련시키고 저장하기

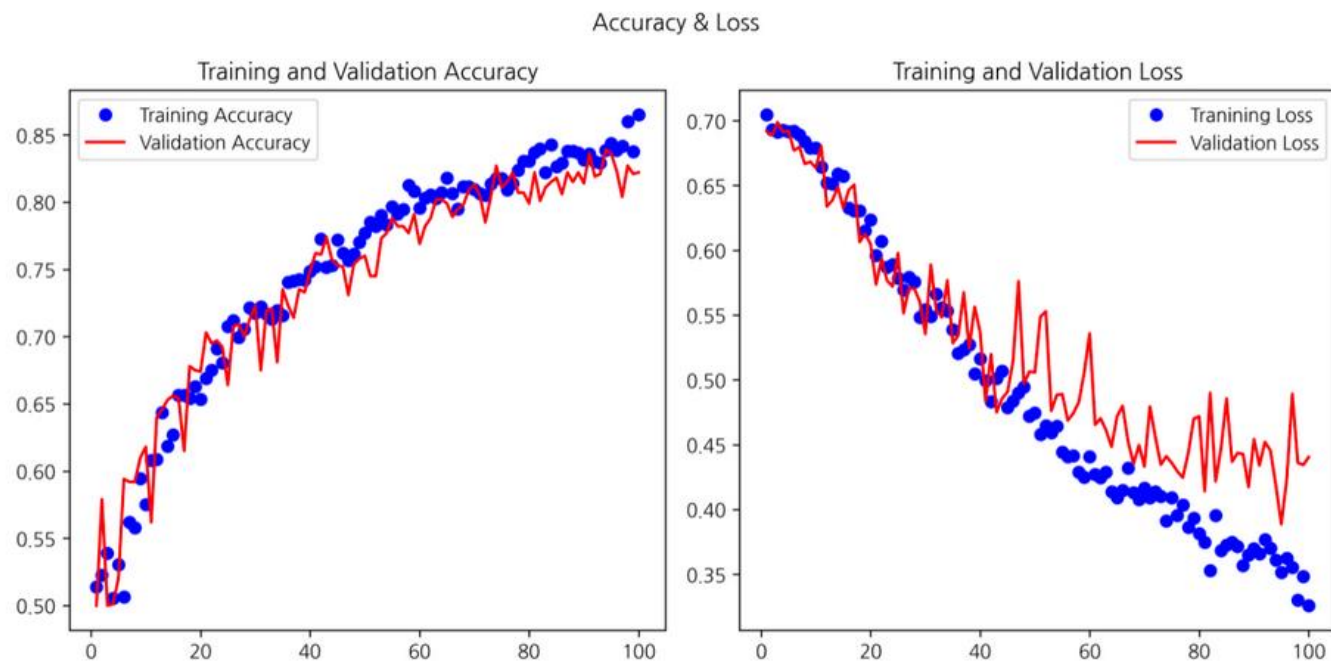
```
60
61 # 모델 훈련
62 history = model.fit_generator(
63     train_generator,
64     steps_per_epoch=100,
65     epochs = 100,
66     validation_data = valid_generator,
67     validation_steps = 50
68 )
69
70 model.save(' /Users/kibeomkim/Desktop/models_saved/dog_and_cant.h5 ')
71
```

CNN

간단한 컨브넷 밑바닥 부터 훈련시키기

훈련 결과 되짚어 보기 - 정확도 및 손실 시각화

```
1  acc = history.history['acc']
2  val_acc = history.history['val_acc']
3  loss = history.history['loss']
4  val_loss = history.history['val_loss']
5
6  epochs = range(1, len(acc) + 1)
7
8  plt.figure(figsize=(10,5))
9
10 plt.subplot(1,2,1)
11 plt.plot(epochs, acc, 'bo', label='Training Accuracy')
12 plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
13 plt.title('Training and Validation Accuracy')
14 plt.legend()
15
16 plt.subplot(1,2,2)
17 plt.plot(epochs, loss, 'bo', label='Tranining Loss')
18 plt.plot(epochs, val_loss, 'r', label='Validation Loss')
19 plt.title('Training and Validation Loss')
20 plt.legend()
21
22 plt.suptitle('Accuracy & Loss')
23 plt.tight_layout()
24
25 plt.show()
```



CNN

컨브넷 학습 시각화

활성화 시각화

- 합성곱 층 출력에 요소 별로 활성화 함수 적용한 결과를 ‘활성화’ 라고 한다.
- 이 활성화를 시각화 해서, 각 층의 의미(역할)를 직접 눈으로 확인할 수 있다.

앞에서 학습 시킨 모델 로드하기

```
1 from keras.models import load_model
2
3 # 저장한 작은 컨브넷 로드
4 model2 = load_model('/Users/kibeomkim/Desktop/models_saved/dog_and_cant.h5')
5 model2.summary()
```

```
1 # 합성곱 층, 풀링 층 출력 시각화
2 # '활성화' 시각화
3 img_path = '/Users/kibeomkim/Desktop/cc.png'
4
5 from keras.preprocessing import image
6
7 img = image.load_img(img_path, target_size=(150,150))
8 img_tensor = image.img_to_array(img) # 텐서로 변환
9 img_tensor = np.expand_dims(img_tensor, axis=0) # 배치 축 추가
10
11 img_tensor /= 255. # 1/255로 스케일 조정
12
13 print(img_tensor.shape)
```

(1, 150, 150, 3)

CNN

컨브넷 학습 시각화

활성화 시각화

```
1  # 원본 이미지 출력
2  plt.figure(figsize=(2,2))
3  plt.imshow(img_tensor[0])
4  plt.xticks([])
5  plt.yticks([])
6  plt.show()
```



[고양이 이미지 출처: <https://www.rd.com/list/black-cat-breeds/>]

CNN

컨브넷 학습 시각화

활성화 시각화

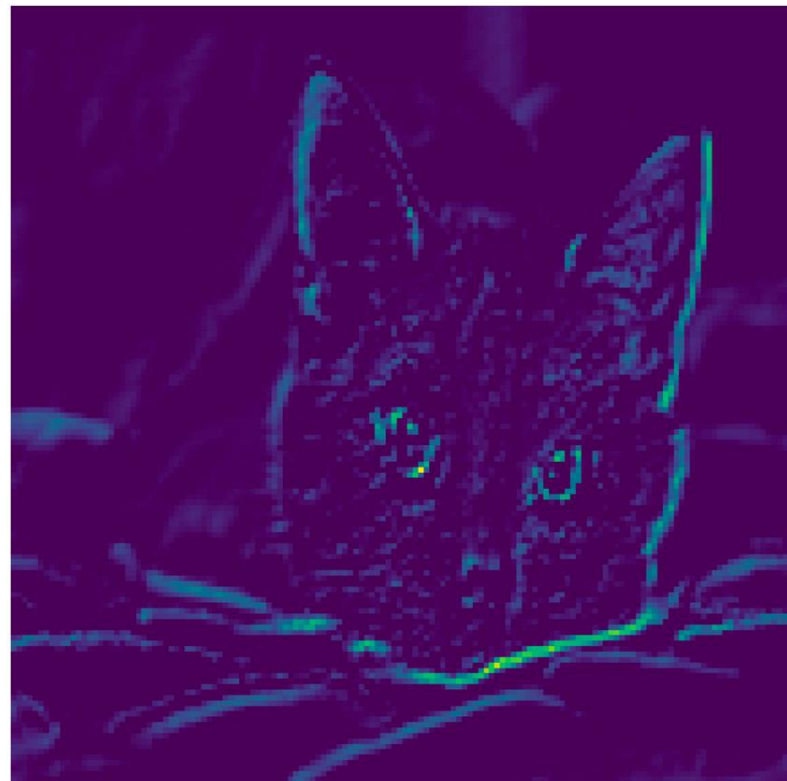
```
1 # 상위 8개 레이어 출력만 추출
2 from keras import models
3
4 layer_outputs = [layer.output for layer in model2.layers[:8]] # 상위 8개 레이어 출력 추출
5
6 # 특정 입력에 대한 출력 매핑하는 모형
7 activation_model = models.Model(inputs=model2.input, outputs=layer_outputs)
8 # 하나 입력에 대해: 8개 출력 대응된다 (총 8개 출력 결과)
9
10 # 예측모드로 모델 실행하기
11 activations = activation_model.predict(img_tensor) # img_tensor 1개 입력에 대해: 8개 층 각각에 통과시켜서 그 출력 반환
12
```

CNN

컨브넷 학습 시각화

활성화 시각화 - 첫번째 합성곱 층 활성화 맵 시각화

```
1 # 첫번째 합성곱 층 활성화 맵 시각화
2 first_layer_activation_result = activations[0]
3 print(first_layer_activation_result.shape) # 합성곱 결과: 높이 148, 너비 148, 배치 1, 필터 적용한 응답 맵 32개
4
5 # 응답 맵 32개 중 12번째 응답 맵 시각화
6 plt.figure(figsize=(2,2))
7 plt.matshow(first_layer_activation_result[0, :, :, 21], cmap='viridis')
8 plt.xticks([])
9 plt.yticks([])
10 plt.show()
```



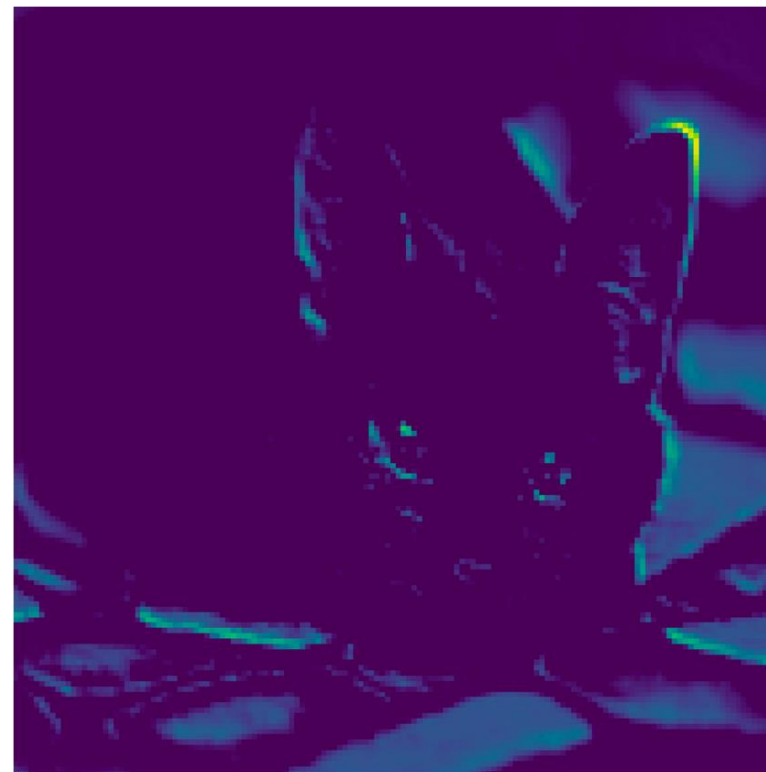
이 필터는 전체 에지를 감지하는 것 같다.

CNN

컨브넷 학습 시각화

활성화 시각화 - 첫번째 합성곱 층 활성화 맵 시각화

```
1 first_layer_activation_result = activations[0]
2 print(first_layer_activation_result.shape) # 합성곱 결과: 높이 148, 너비 148, 배치 1, 필터 적용한 응답 맵 32개
3
4 # 응답 맵 32개 중 2번째 활성화 맵
5 plt.figure(figsize=(2,2))
6 plt.matshow(first_layer_activation_result[0, :, :, 2], cmap='viridis')
7 plt.xticks([])
8 plt.yticks([])
9 plt.show()
10
```



첫번째 층 두번째 필터는 도드라진 엣지 감지하는 것 같다.(고양이 귀 끝 부분)

CNN

컨브넷 학습 시각화

네트워크 모든 활성화 시각화

```
1  # 네트워크 모든 활성화 시각화
2  layer_names = []
3  for layer in model2.layers[:8] : # 상위 8개 층 이름들 추출 (그림 이름으로 사용)
4      layer_names.append(layer.name)
5  print(layer_names)
```

['conv2d_13', 'max_pooling2d_11', 'conv2d_14', 'max_pooling2d_12', 'conv2d_15', 'max_pooling2d_13', 'conv2d_16', 'max_pooling2d_14']

CNN

컨브넷 학습 시각화

네트워크 모든 활성화 시각화

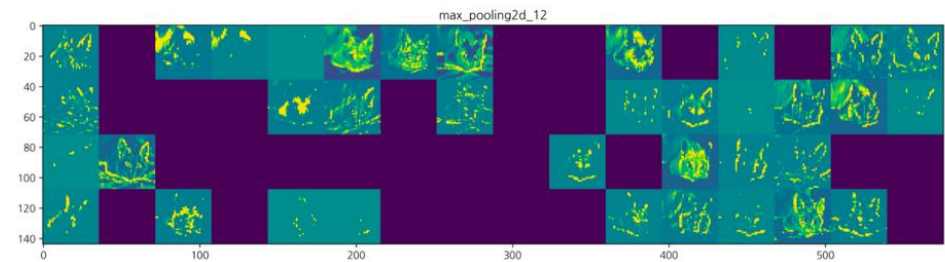
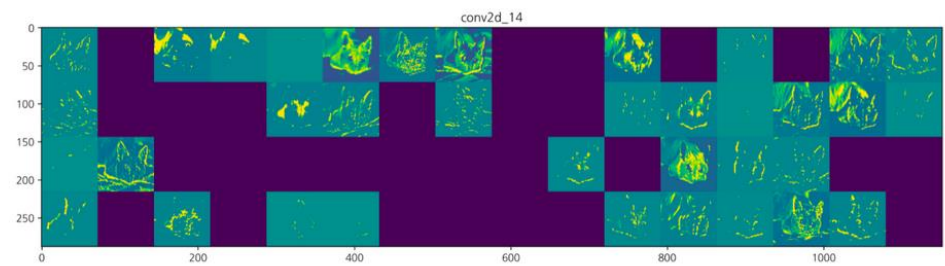
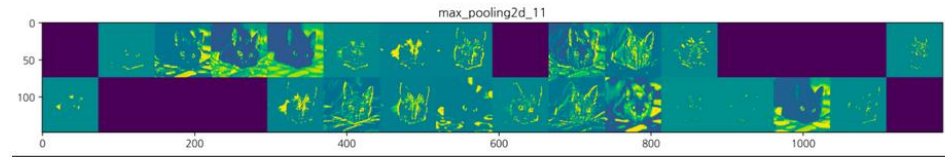
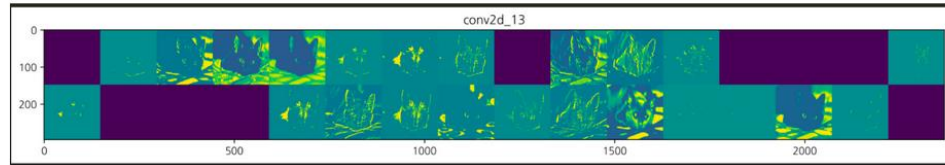
```
1  images_per_row = 16
2
3  for layer_name, layer_activation in zip(layer_names, activations) : # 층 - 활성화 매핑
4      n_features = layer_activation.shape[-1] # 활성화 마다 채널 수
5
6      size = layer_activation.shape[1] # 높이와 너비 어차피 같다.
7
8      n_cols = n_features // images_per_row
9      display_grid = np.zeros((size*n_cols, images_per_row*size))
10
11     for col in range(n_cols) :
12         for row in range(images_per_row) :
13             channel_image = layer_activation[0, :, :, col*images_per_row+row]
14             # 채널 이미지 스케일 조정
15             # 정규화
16             channel_image -= channel_image.mean()
17             channel_image /= channel_image.std()
18
19             channel_image*=64
20             channel_image += 128
21             channel_image = np.clip(channel_image, 0, 255).astype('uint8')
22             display_grid[col*size : (col+1)*size, row*size:(row+1)*size] = channel_image
23
24     scale = 1./size
25     plt.figure(figsize=(scale*display_grid.shape[1], scale*display_grid.shape[0]))
26     plt.title(layer_name)
27     plt.grid(False)
28     plt.imshow(display_grid, aspect='auto', cmap='viridis')
29     #plt.xticks([])
30     #plt.yticks([])
31     plt.show()
```

CNN

컨브넷 학습 시각화

네트워크 모든 활성화 시각화

하위 4개 층 활성화 출력 시각화 결과

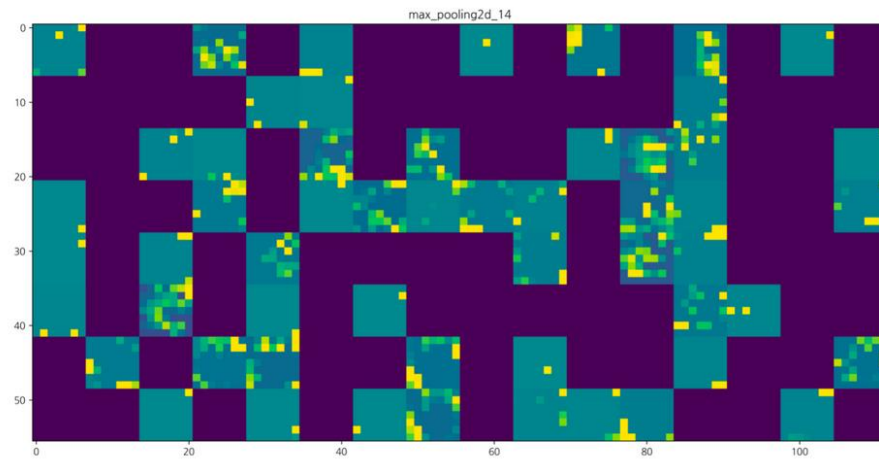
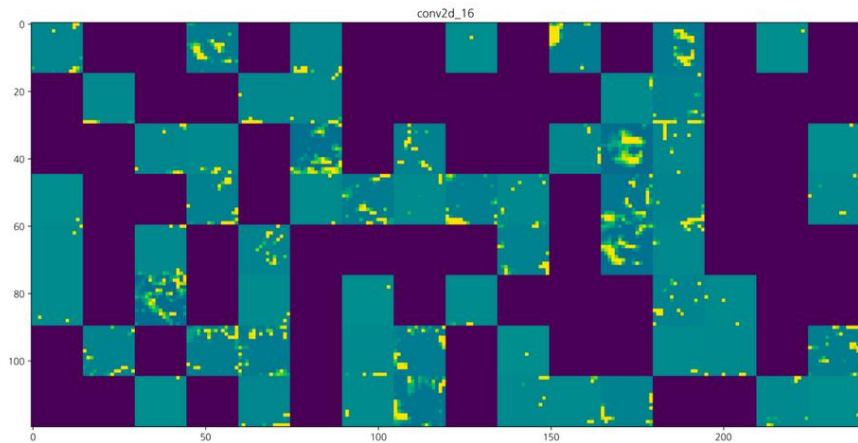
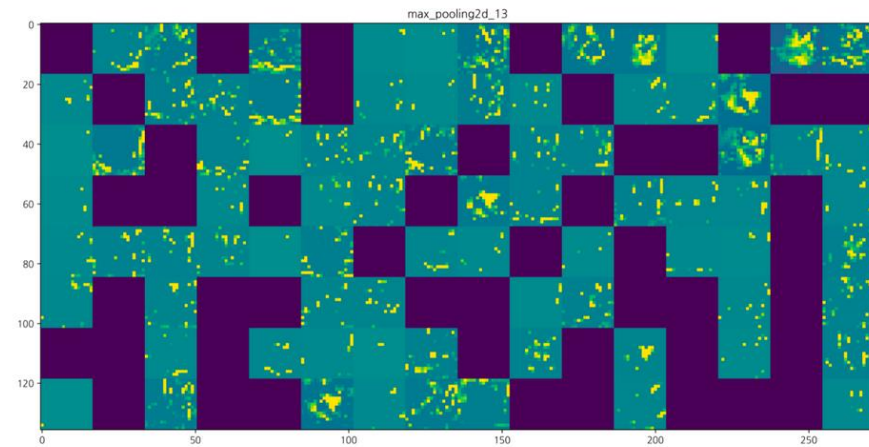
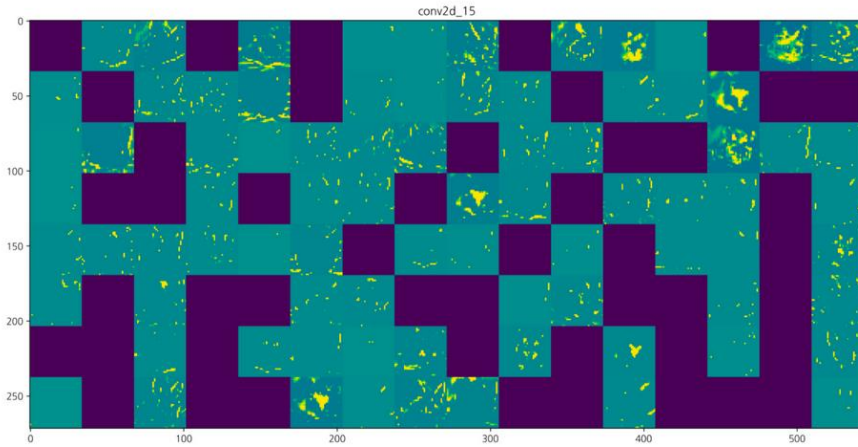


CNN

컨브넷 학습 시각화

네트워크 모든 활성화 시각화

- 상위 4개 층 활성화 출력 시각화 결과

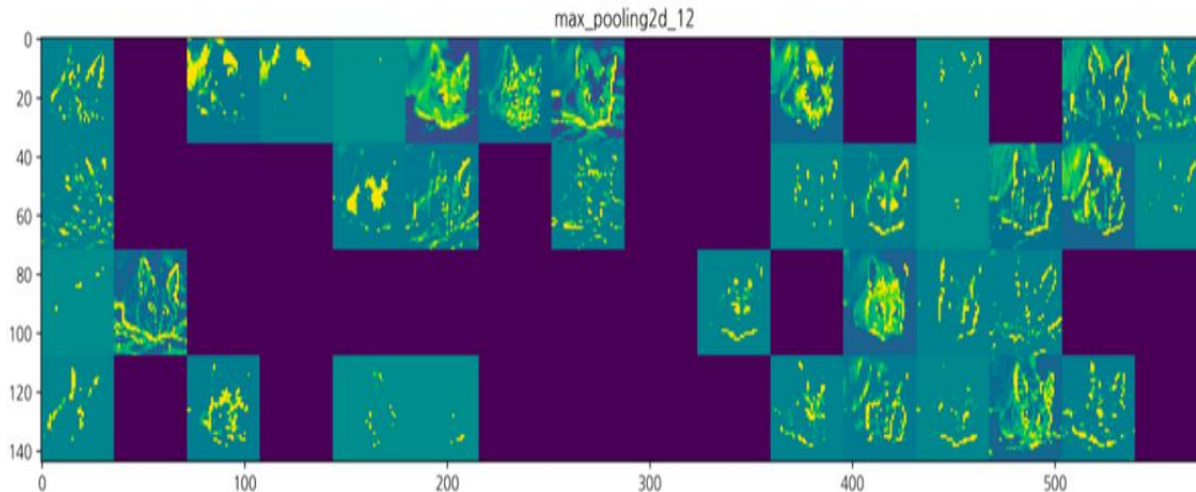


CNN

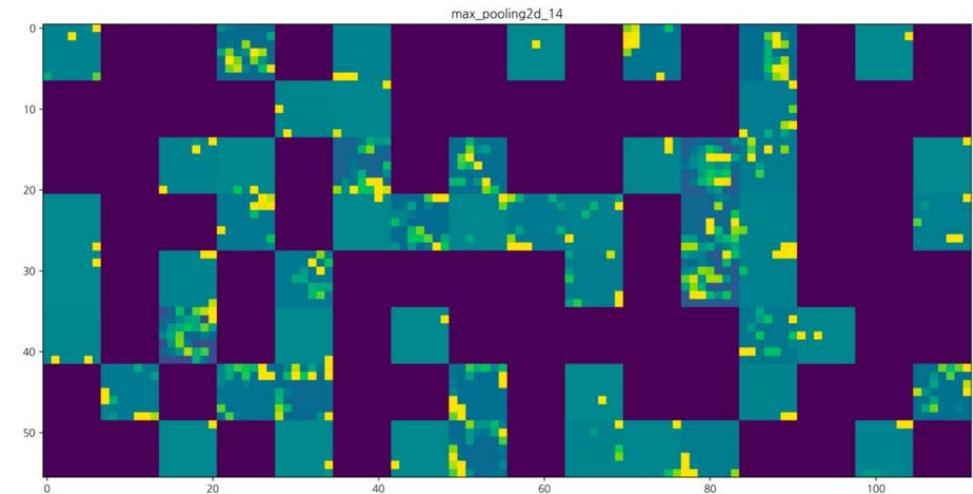
컨브넷 학습 시각화

네트워크 모든 활성화 시각화

하위 층 출력 예시



상위 층 출력 예시



- 상위 층으로 갈 수록 활성화 맵 의미 시각적 파악 어려워진다.
- 상위 층 갈 수록, 모델이 학습한 몇 가지 '고양이라면 갖고 있을 만한 주요 특징들'만 남게 된다. 곧, 상위 층 각 필터는 '고양이의 특징'들이다.
- 모델은 입력 이미지 위를 슬라이딩 하면서 자신이 학습한 '고양이 특징'들이 입력 이미지에 나타나 있는지 확인한다.
- 위 마지막 이미지가 듬성듬성 빈 건, 입력 이미지가 모델이 학습한 고양이 특징들 중 일부를 안 갖고 있었다는 말이다. 곧, 필터에 응답 발생하지 않았다.
- 결과로 나온 활성화 맵 하나하나가 추상화 된 고차원적 의미 담게 된다. 예컨대 고양이 귀, 입, 눈 등이다.
- CNN이 객체를 인식하는 방법은 인간 두뇌가 현실세계의 객체 인식하는 방법과 매우 유사하다.

CNN

컨브넷 학습 시각화



기억에 의존해 그린 고양이 얼굴 / 실제 고양이 사진

[오른쪽 고양이 사진 출처: <https://petdoc.co.kr/ency/280>]

우리 인간은 객체를 주요 특징 몇 가지를 가지고 인식한다. 기억에 의존해 고양이 얼굴을 그릴 때 나는 머릿속으로 생각했다.

고양이는 뾰족한 귀가 있다.

고양이는 입, 코, 그리고 눈이 있다.

얼룩 고양이 였다.

고양이는 인종에 콧수염이 있다.

고양이에 대한 추상화 된 몇 가지 특징들만 가지고 '이것은 고양이다'라고 인식했다.

세부 디테일은 기억하지 않는다. 온전하게 기억할 수도 없다.

세부 디테일은 가지치기 하고 주요 특징 만을 기억하고. 객체 인식한다.

CNN 모델도 그러하다.