[Tensorflow 2.0] 합성곱 신경망: CNN

Erica Bae · Follow

16 min read · Nov 4, 2019

Share

I. 들어가며

이 포스팅에 관해.

이 글은 이미지를 tf.data로 가져오는 이전 포스팅의 다음 단계로, 이미지 데이터에 관한 기본적인 신경망을 이해하는 데 주력합니다. 이번 포스팅에서는 학습 데이터셋과 테스트 데이터셋이 다 있는 CIFAR images를 사용하겠습니다.

CNN에 좀 더 집중하기 위해, tf.data 대신 *(train_images, train_labels), (test_images, test_labels)*를 통해 이미지를 바로 가져오겠습니다.

이제 우리가 할 것은?

여러분은 검색하실 때 좀 더 결과가 잘 나오게 내용을 맞춰서 입력하신 경험 있으신가요? 이미지를 가져오는 것도 마찬가지로 이해될 수 있는 형태로 바꿔줘야 하는데요, CNN을 활용해 바꿔보겠습니다.

II. 준비하기

```
from __future__ import absolute_import, division, print_function,
unicode_literals

try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

이제 익숙하시죠? 근데 알파벳 하나하나까지 다 알려고 하실 필요까진 없을 것 같습니다.

- from **tensorflow.keras** import **datasets, layers, models**: 모델을 만들 때 keras sequential API를 사용할 것입니다.
- import **matplotlib.pyplot** as plt: 이미지를 표현하기 위해 점들을 찍을 것입니다.

III. CIFAR10 데이터셋

CIFAR10 데이터셋에 관해.

이 데이터셋은 6K 이미지들로 구성된 클래스를 10개 갖는 60K 32*32 컬러 이미지들로 구성되어있습니다. 학습 이미지는 50K, 테스트 이미지는 10K이고, 각각 랜덤으로 섞인 클래스들로 구성된 batch를 5개, 1개 가집니다. 이미 전처리가 깔끔하게 되어있네요! 👍

클래스 10개의 이름은; `'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'`.

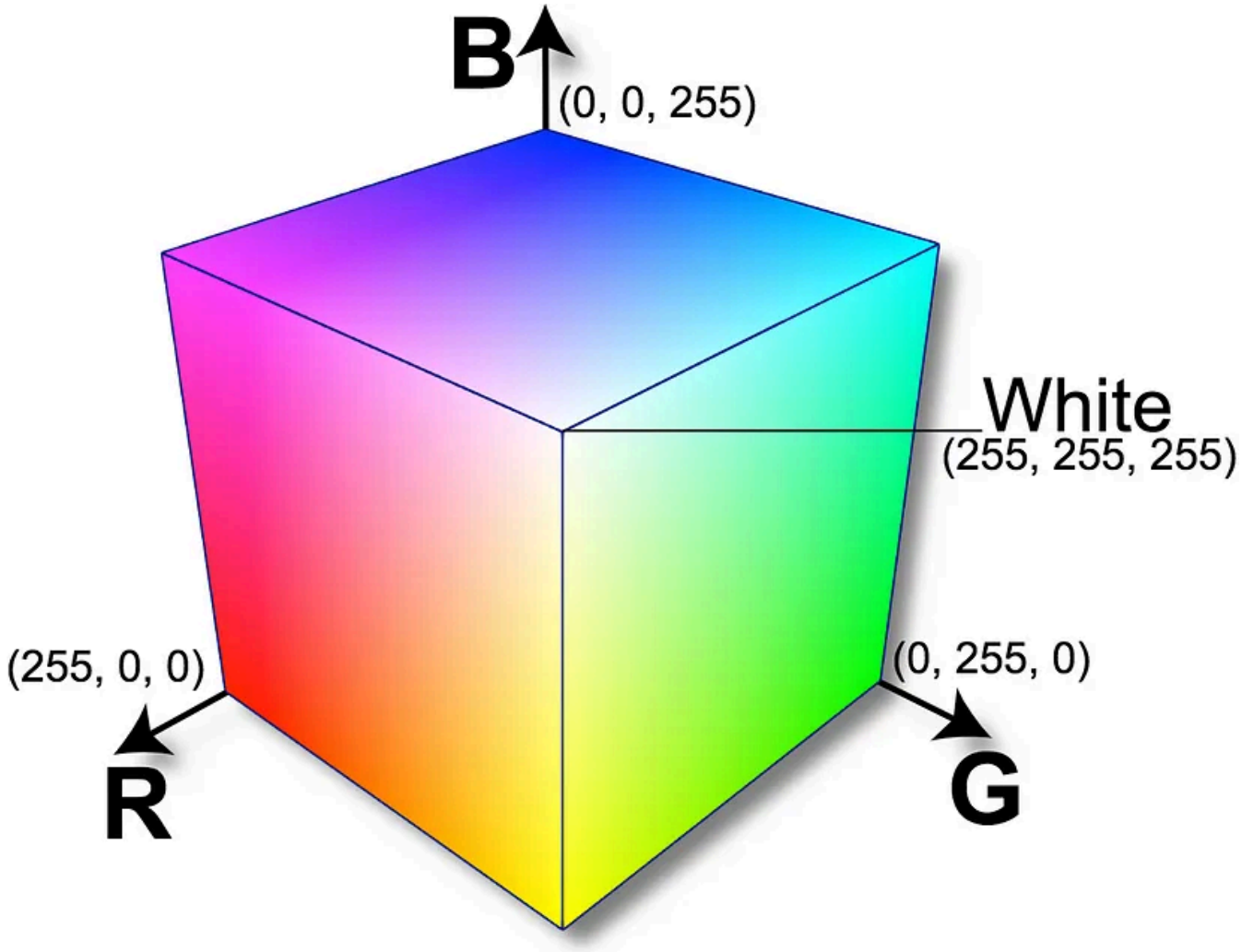
먼저, cifar10 데이터셋을 다운로드 하겠습니다.

```
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()
```

그리고, 픽셀값들을 0과 1사이의 값으로 정규화하겠습니다.

```
train_images, test_images = train_images / 255.0, test_images /
255.0
```

픽셀값이란?



from instructables.com

왜 이미지들이 255로 나눠질까요? 살펴봅시다.

RGB는 빨간색(Red), 초록색(Green), 파란색(Blue)을 나타냅니다. 인간이 색을 어떻게 인식하는지 자세히 모르더라도, 어쨌든 컴퓨터는 이미지와 색을 RGB 픽셀값으로 저장하고 인식합니다.

R, G, B는 각각 0부터 255까지의 숫자를 가질 수 있고, 컬러 픽셀은 1X3 (R, G, B)벡터로 저장됩니다. 그래서 **color_channels** 역시 3입니다.

그러면 한 컬러 이미지는 각각 R, G, B를 나타내는 격자무늬 3개를 갖고, *image_height*와 *image_width*값을 갖습니다.

'정규화'는 다양한 의미를 갖지만, 여기서는 계산의 편리함을 위해 0과 1 사이의 숫자로 표현하는 것을 의미합니다.

이미지 몇 가지 직접 보기

이미지 몇 개를 직접 볼까요? 이미지 10개를 띄우고, 각 이미지 밑에 클래스 이름을 달아서 살펴보겠습니다!

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

IV. Convolutional 기반 만들기

i) 우선, 모델을 세우는 방법을 간단히 해보겠습니다.

아래는, 우리가 이 포스팅을 통해 만들 최종 모델입니다.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])
```

이렇게 할 수도 있지만, **model.add** 방법을 사용해서 **tf.keras.layers** 반복을 생략할 수 있습니다. 즉, 비어있는 sequential 모델을 만들고, 레이어를 추가하는 것입니다.

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=
(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

ii) *Convolution*과 *MaxPooling*이란?

1 — Convolution: 이미지에 kernel이나 필터를 적용하는 과정

```
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3))
```

- **input_shape=(32,32,3):** ***image_height=32, image_width=32,*** 그리고 ***color_channels*** 는 *3*!
- **activation='relu'**
  relu 는 ***Rectified Linear Unit***의 약자입니다. 보통 활성화 함수로 쓰이고, input값이 0이하이면 0으로 나타내고, input값이 양수이면 input값을 그대로 나타냅니다. 비선형 문제를 해결하기 용이하게 만들어줍니다. 활성화 함수로 뭘 써야할지 잘 모르겠다면, 일단 *relu!*
- **(3,3): kernel**이 **3*3** 격자무늬
- **32**는 **kernel** 필터의 개수

[Example] 0(흑)부터 255(백) 사잇값을 갖는 회색조 **6 by 6 pixel image**를 간단하게 예시로 보겠습니다.

한 픽셀값의 주변을 다 둘러 3X3 kernel에 곱하고 그 값을 더해 convoluted image를 나타냅니다.

Pixel Values

1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	200	76
34	66	134	223	201	3
255	123	89	55	32	2

3 by 3 kernel

1	2	1
2	4	2
1	2	1

Convoluted Image

22	27	36	313	722	576
91	110	120	522	984	576
284	257	198	755	1360	798
507	567	687	1312	1689	955
1061	1288	1496	1911	1659	702
1400	1480	1269	1249	870	279

2*1 + 5*2+4*1 + 13*2 + 25*4 +15*2 +8*1+6*2+6*1=**198**

그럼 주변에 값이 없는 가장자리는 어떻게 할까요? 이 때 **Zero-padding**이 쓰일 수 있습니다. 0으로 왼쪽과 위쪽 가장자리를 둘러싼 2X2 행렬을 예시로 들어 보겠습니다.

0 0 0

0 1 0

0 8 2

0*1 +0*2+0*1+0*2+**1*4**+0*2+0*1+**8*2 +2*1**=4+16+2=**22.**

padding= ‘same’을 코드에 작성해 zero-padding을 할 수 있습니다.

예시) model.add(layers.Conv2D(32, (3, 3), **padding=’same’,** activation=’relu’, input_shape=(32, 32, 3)))

Pixel Values

1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	200	76
34	66	134	223	201	3
255	123	89	55	32	2

3 by 3 kernel

1	2	1
2	4	2
1	2	1

Convoluted Image

22	27	36	313	722	576
91	110	120	522	984	576
284	257	198	755	1360	798
507	567	687	1312	1689	955
1061	1288	1496	1911	1659	702
1400	1480	1269	1249	870	279

zero-padding

또는 가장자리에 있는 값들을 다 고려하지 않음으로써 **‘no padding’**을 사용할 수 있습니다. no-padding을 적용한 경우, output결과가 아래와 같이 6X6에서 4X4로 줄어든 것을 확인할 수 있습니다.

Pixel Values					
1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	200	76
34	66	134	223	201	3
255	123	89	55	32	2

3 by 3 kernel		
1	2	1
2	4	2
1	2	1

Convoluted Image					
	110	120	522	984	
	257	198	755	1360	
	567	687	1312	1689	
	1288	1496	1911	1659	

No padding

relu함수를 사용했지만, 이 kernel이 음수값을 갖지 않아 결과에서 많이 차이가 나지 않네요.

2 — MaxPooling: convoluted 이미지의 최댓값을 줄이는 것(=downsampling)

```
layers.MaxPooling2D((2, 2))
```

이제 convoluted 이미지 격자무늬를 만들었는데요, 2X2 kernel을 활용해 최댓값을 뽑는 작업을 해보겠습니다. stride값은 2X2 kernel이 픽셀을 몇 개씩 건너뛰어야하는지 할당해주는 값입니다.

Convoluted Image					
22	27	36	313	722	576
91	110	120	522	984	576
284	257	198	755	1360	798
507	567	687	1312	1689	955
1061	1288	1496	1911	1659	702
1400	1480	1269	1249	870	279

Max Pooling
(2,2) stride=2

New Image		
110	552	984
567	1312	1689
1480	1911	1659

이미지 사이즈가 반으로 줄어든 것이 보이시나요?

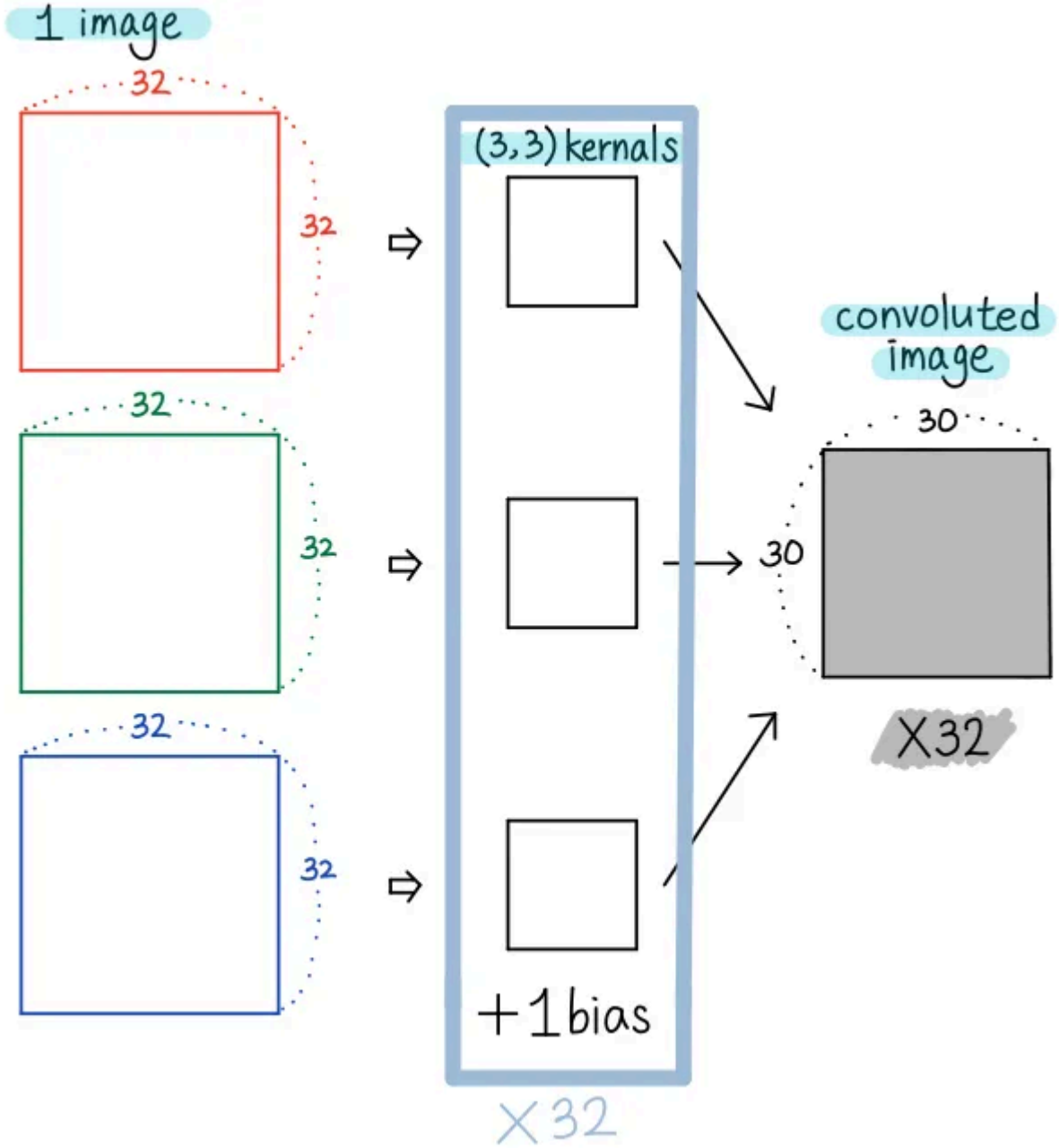
IV-a. model.summary()

model.summary() 을 실행하면 output값은:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

요약표의 숫자들을 해석해볼까요**?**

1 — model.add(layers.Conv2D(32, **(3, 3)**, padding='same', activation='relu', input_shape=(32, 32, 3)) → **(None, 30,30,32), param # 896**

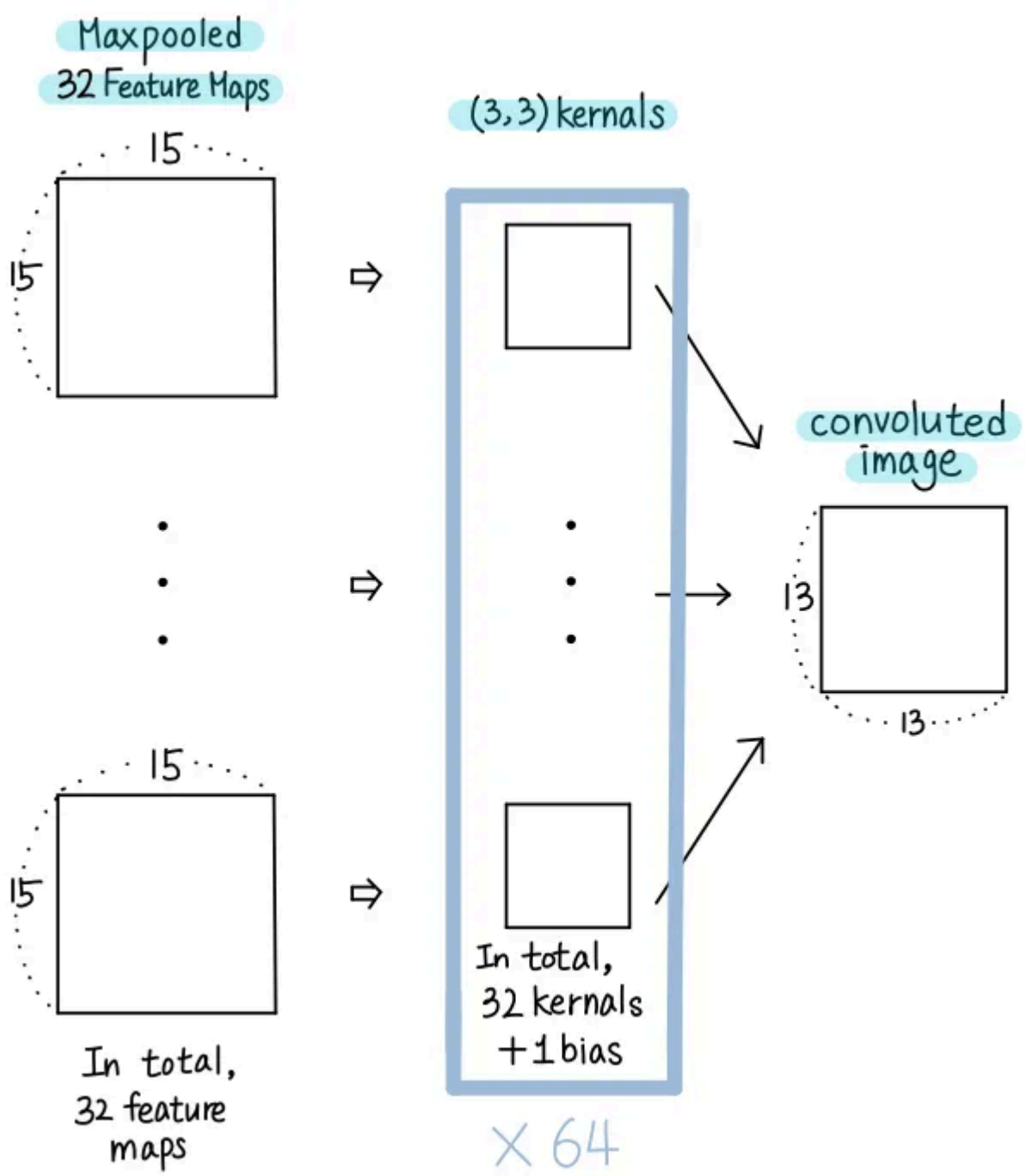


- 예상했던대로, 인풋값**(32,32)**이 no-padding을 통해 **(30,30)**로 줄어들었습니다.
- 32는 나타날 kernel들의 수입니다. 각 kernel은 하나의 convoluted 이미지를 갖는데요. 정리하면, **feature map**이라고도 불리는 **convolution** 채널을 총 **32**개 가지게 됩니다.
- **parameter**는 몇 개?
  우리는 R, G, B각각에 3*3 픽셀 kernel을 적용시켰습니다. 따라서 3*3*3=27개의 픽셀과, bias kernel을 하나 가집니다. 정리하면, parameter의 개수는 **(27+1)*32=896**개 입니다.

2 — model.add(layers.MaxPooling2D((2, 2))) → **(None,15,15,32)**

- 우리는 아까 2*2 격자무늬로 MaxPooling, 즉 'downsampling'을 했습니다. 따라서 feature 개수는 그대로 **32**개이고, 이미지의 크기는 반입니다!

3 — model.add(layers.Conv2D(64, (3, 3), activation='relu')) → **(None,13,13,64), param# 18496**



- MaxPooling을 적용한 15*15 featuremap은 convoltion 필터를 통과하며 가장자리가 삭제되어 13*13으로 크기가 줄어듭니다.
- 이 convolution 레이어(이하 합성곱 레이어)는 3*3픽셀로 구성된 kernel을 64개 갖습니다.

- 이제 두번째 합성곱 레이어를 만들건데, 음… 64개로 해볼까요?

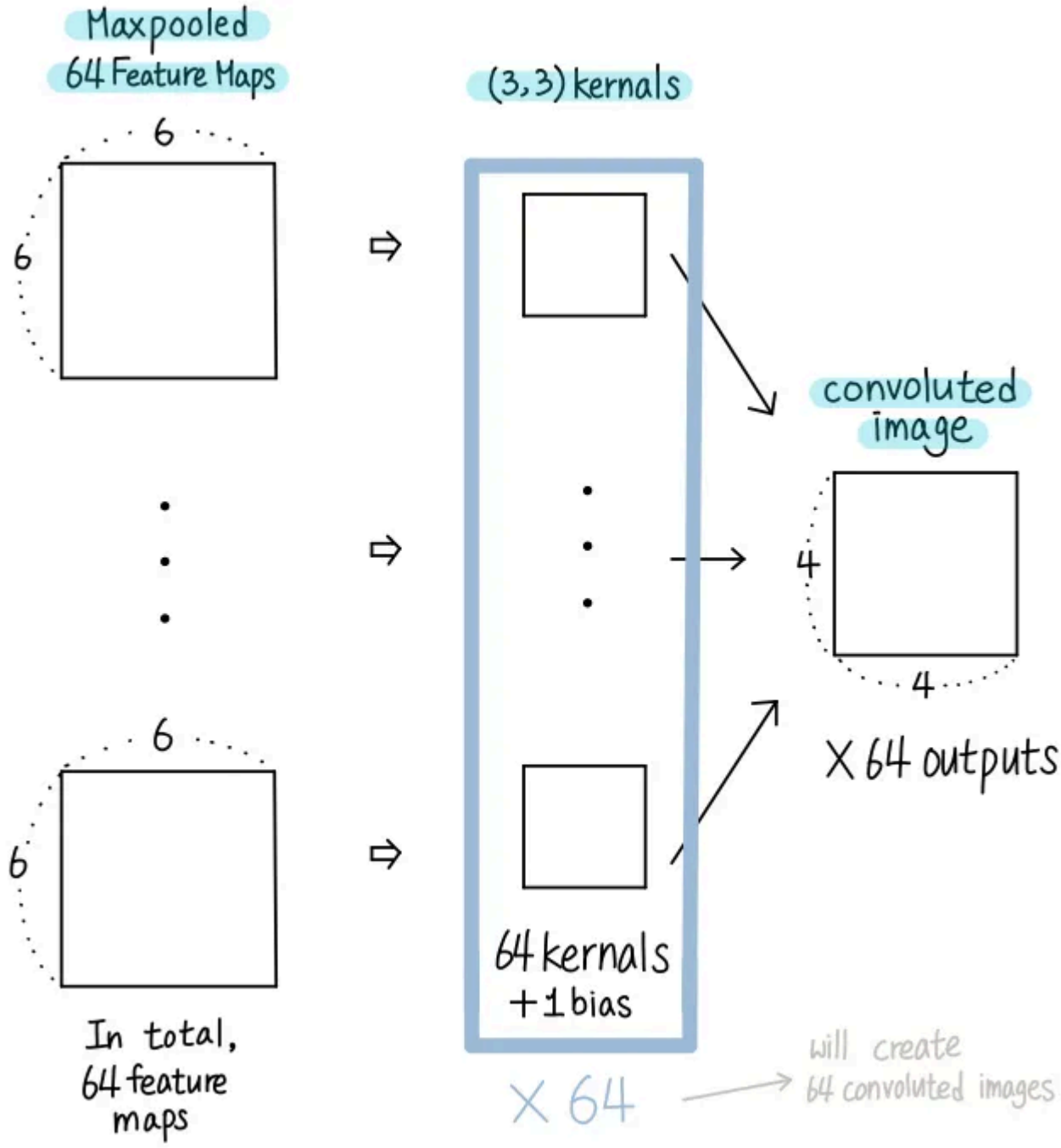
- 몇 개의 **parameter?**
  첫번째 합성곱 레이어는 feature map 32개로 구성되었고, 두번째 레이어는 64개로 구성되었고, 결과적으로 아웃풋 64개를 얻었습니다. 필터의 사이즈가 3*3이고 개수는 32개 였으므로, 두번째 레이어에서는 총 (3*3*32+1)*64=18496개의 parameter를 가집니다.

> *Side note) 32*개의 *feature maps* 첫번째 합성곱 레이어의 *output*이면서, 동시에 두번째 합성곱 레이어의 *input*입니다.

4 — model.add(layers.MaxPooling2D((2, 2))) → **(None, 6,6,64)**

- 개수는 그대로지만, 크기는 또 다시 반으로 줄여졌습니다.

5 — model.add(layers.Conv2D(64, (3, 3), activation='relu')) → **(None, 4,4,64)**
param#36928



- no-padding을 적용해서 다시 높이와 너비 모두 2픽셀씩 줄어들었고, 결과적으로 이 합성곱 레이어는 4*4 픽셀을 갖는 64개의 레이어로 구성됩니다.

- 몇 개의 **parameters?**
  두번째 합성곱 레이어로부터 우리는 64개의 feature map을 input으로 얻어냈고, 3*3 필터사이즈를 유지하면서 64개의 feature을 output으로 가졌습니다. 따라서, (3*3*64+1)*64=36928 **parameters**를 가집니다.

최종적으로 다 합치면, **896+18496+ 36928=56320.**

V. 맨 위에 Dense layer 추가하기

이제 분류하기 위해서, dense layer가 필요합니다. Dense layer는 1차원 벡터를 갖기 때문에, 앞의 convolutional 레이어의 output을 쪼개야 합니다. 3차원의 (4, 4, 64)

tensor를 1차원의 tensor나 4*4*64==1024개 원소로 구성된 벡터로 바꾸겠습니다. 이를 위해 **flatten() layer**를 사용하겠습니다.

몇 개의 dense layer를 갖든, 최종 레이어는 softmax 활성화 함수를 활용해 레이어 개수를 클래스 개수와 같은 10개로 만들어야 합니다.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

V-a. Model 요약

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

6 — model.add(layers.Flatten()) →**(None,1024)**
마지막 합성곱 레이어에서 4*4 사이즈의 output channel 64개를 인풋으로 가지는데요. flatten layer는 1차원 벡터이므로 4*4*64개의 원소를 갖습니다.

7 — model.add(layers.Dense(64, activation='relu')) →**(None,64) param#65600**
이 dense layer에는 64개의 노드가 있고, flatten layer로부터의 1024개의 input이 있습니다. 각 노드에 1024개의 weight와 bias 1개가 있습니다. **(1024+1)*64=65600.**

8 — model.add(layers.Dense(10, activation='softmax')) →**(None,10), param#650**
이 dense layer에는 노드 10개와 이전 dense layer로부터 나온 64개의 input이 있습니

다. **(64+1)*10=650.**

최종적으로, 896+18496+36928+65600+650=122570

VI. 모델 편찬하고 학습시키기

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

VII. 모델 평가하기

```
test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
```

output 10000/1–1s — **loss: 0.8848 — accuracy: 0.7142**를 얻었습니다. 좋지도, 나쁘지도 않은 수치네요!

계속 배워 정확도를 올립시다!

이번 합성곱신경망 포스팅에서는 CNN을 활용해 이미지를 다뤄보았습니다.도움이 많이 되셨나요?
궁금하신 점이나 개선할 점 등 댓글은 언제나 환영합니다.
읽어주셔서 감사합니다.