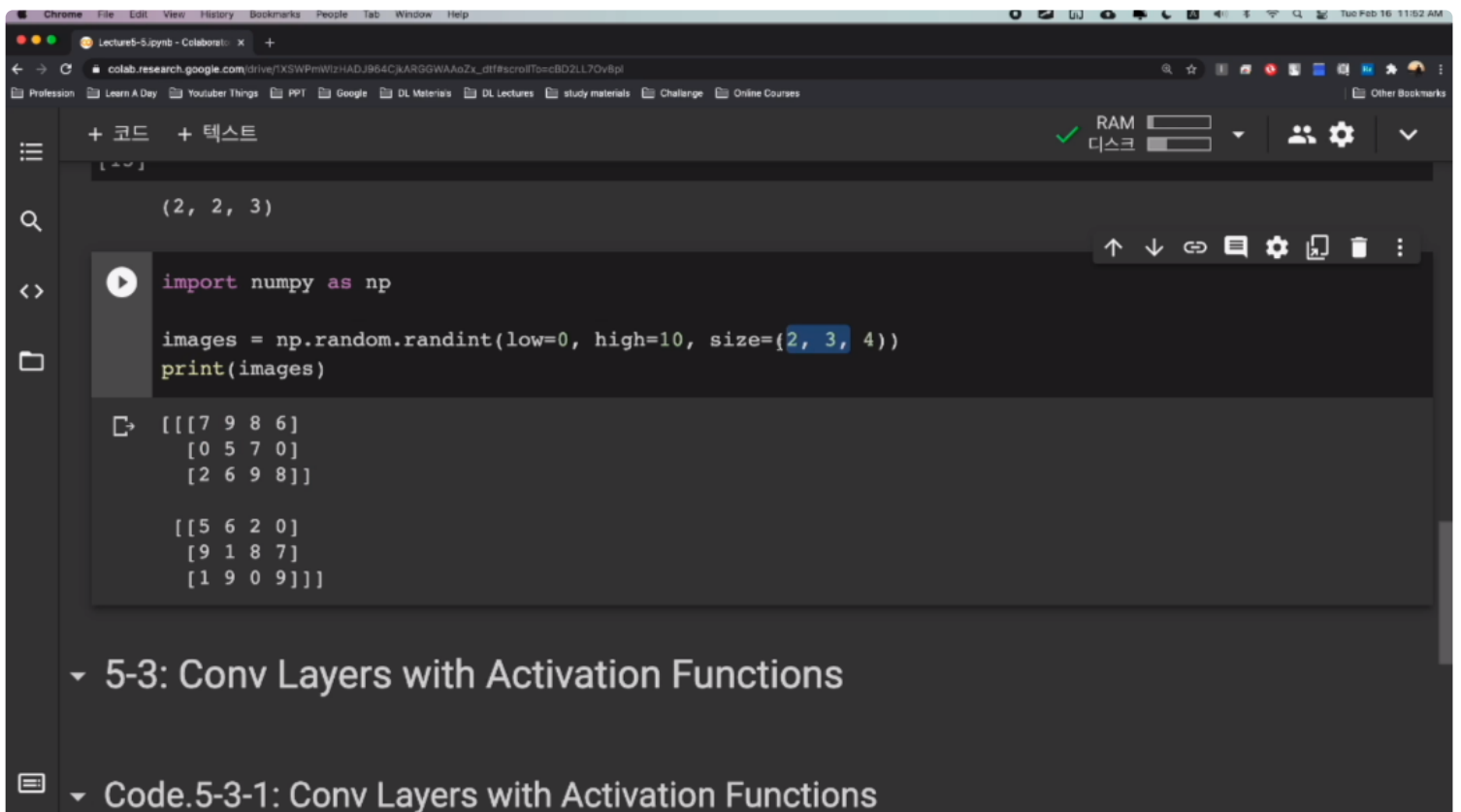


3 x 4가 제일 안쪽에 있는걸 볼 수 있다.



하지만 우리가 보는 것들은 2 x 3 즉 2, 3, 0일 때 첫 번째 이미지 2 x 3 1일 때 두 번째 이미지 2 x 3 2일 때 세 번째 이미지 등등.

Chrome | File | Edit | View | History | Bookmarks | People | Tab | Window | Help

Lecture5-5.ipynb - Colaboratory | colab.research.google.com | RAM | 디스크

```
import numpy as np

images = np.random.randint(low=0, high=10, size=(2, 3, 4))
for c in range(4):
    print(images[:, :, c])
```

```
[[4 7 7]
 [6 7 9]]
[[2 7 8]
 [8 7 8]]
[[9 7 4]
 [4 6 1]]
[[2 6 1]
 [5 9 8]]
```

5-3: Conv Layers with Activation Functions

Code.5-3-1: Conv Layers with Activation Functions

Chrome | File | Edit | View | History | Bookmarks | People | Tab | Window | Help

Lecture5-5.ipynb - Colaboratory | colab.research.google.com | RAM | 디스크

```
import numpy as np

images = np.random.randint(low=0, high=10, size=(2, 3, 4))
for c in range(4):
    print(images[:, :, c])

tmp = np.transpose(images, (b))
```

```
[[0 2 5]
 [6 1 7]]
[[9 2 0]
 [3 0 3]]
[[9 2 9]
 [6 9 0]]
[[9 4 0]
 [7 1 8]]
```

5-3: Conv Layers with Activation Functions

np. transpose()이 뭐냐면 images를 위치를 바꿔주겠다는 것이다.

Colab interface showing a Jupyter Notebook with the following code:

```
import numpy as np

images = np.random.randint(low=0, high=10, size=(2, 3, 4))
for c in range(4):
    print(images[:, :, c])

tmp = np.transpose(images, (2, 0, 1))
```

Handwritten annotations show the dimensions of the arrays:

- `images` has shape $(2, 3, 4)$ (height, width, channels).
- `tmp` has shape $(4, 2, 3)$ (channels, height, width).

The output of the code is:

```
[[0 2 5]
 [6 1 7]]
[[9 2 0]
 [3 0 3]]
[[9 2 9]
 [6 9 0]]
[[9 4 0]
 [7 1 8]]
```

5-3: Conv Layers with Activation Functions

images의 shape를 보면 $2 \times 3 \times 4$ 가 있는데 $2 \rightarrow 0$ $3 \rightarrow 1$ $4 \rightarrow 2$

`transpose(images, (2, 0, 1))`로 해주면 4를 맨앞으로 2 하고 3은 한칸씩 뒤로 밀림 그래서 shape를 찍어보면 $4 \times 2 \times 3$ 그래서 우리가 기존에 가지고 있던 차원을 재배치해준다고 이해하면 된다.

Colab interface showing a Jupyter Notebook with the following code:

```
# Forward Propagation(Manual)
images = images.numpy().squeeze()

Y_man = np.zeros(shape=(n_H - k_size + 1, n_W - k_size + 1, n_filter))
for c in range(n_filter):
    c_W = W[:, :, :, c]
    c_b = B[c]

    for h in range(n_H - f + 1):
        for j in range(n_W - f + 1):
            window = images[h:h+f, w:w+f, :]
            conv = np.sum(window*c_W) + c_b

    Y_man[h, w, c] = conv

print(Y_man.shape)
```

...

```
import numpy as np
```

Chrome | File | Edit | View | History | Bookmarks | People | Tab | Window | Help

Lecture5-5.pynb - Colaboratory | [Python 3] | np.swapaxes, np. | X | +

colab.research.google.com/drive/7XSWPmWizHADJ964CjARGGWAoZx_dtf#scrollTo=cBO2LL7Ov8pl

Profession | Learn A Day | Youtube Things | PPT | Google | DL Materials | DL Lectures | study materials | Challenge | Online Courses

Other Bookmarks

+ 코드 + 텍스트

✓ RAM 디스크

Y_man[h, j, c] = conv

print("Y(Manual): \n", np.transpose(Y_man, (2, 0, 1)))

...

import numpy as np

Chrome | File | Edit | View | History | Bookmarks | People | Tab | Window | Help

Lecture5-5.pynb - Colaboratory | [Python 3] | np.swapaxes, np. | X | +

colab.research.google.com/drive/7XSWPmWizHADJ964CjARGGWAoZx_dtf#scrollTo=DuNEVwz_sQV5

Profession | Learn A Day | Youtube Things | PPT | Google | DL Materials | DL Lectures | study materials | Challenge | Online Courses

Other Bookmarks

+ 코드 + 텍스트

✓ RAM 디스크

print("Y(Manual): \n", np.transpose(Y_man, (2, 0, 1)))

Y(Tensorflow):

```
[[[-0.66233444  0.03978199]
 [ 0.07426445  0.395805  ]]

 [[ 0.40902352  0.4119984 ]
 [ 0.00744987  0.29576278]]

 [[ 0.31782514  0.5805044 ]
 [ 0.27967095  0.38553327]]]
```

Y(Manual):

```
[[[-0.6523345  0.03978193]
 [ 0.07426444  0.395805  ]]

 [[ 0.40902352  0.41199845]
 [ 0.00744987  0.29576275]]

 [[ 0.31782517  0.58050442]
 [ 0.27967095  0.3855333 ]]]
```

[25] import numpy as np

images = np.random.randint(low=0, high=10, size=(2, 3, 4))

```
W, B = conv.get_weights()

# Froward Propagation(Manual)
images = images.numpy().squeeze()

Y_man = np.zeros(shape=(n_H - k_size + 1, n_W - k_size + 1, n_filter))
for c in range(n_filter):
    c_W = W[:, :, :, c]
    c_b = B[c]

    for h in range(n_H - k_size + 1):
        for j in range(n_W - k_size + 1):
            window = images[h:h+k_size, j:j+k_size, :]
            conv = np.sum(window*c_W) + c_b

            Y_man[h, j, c] = conv

print("Y(Manual): \n", np.transpose(Y_man, (2, 0, 1)))

Y(Tensorflow):
[[[-0.66233444  0.03978199]
  [ 0.07426445  0.395805   ]]
```

```
shape=(N, n_H, n_W, n_C))

# Forward Propagation(Tensorflow)
conv = Conv2D(filters=n_filter, kernel_size=k_size)
Y = conv(images)
Y = np.transpose(Y.numpy().squeeze(), (2, 0, 1))
# print("Y(Tensorflow): \n", Y)

W, B = conv.get_weights()

# Froward Propagation(Manual)
images = images.numpy().squeeze()

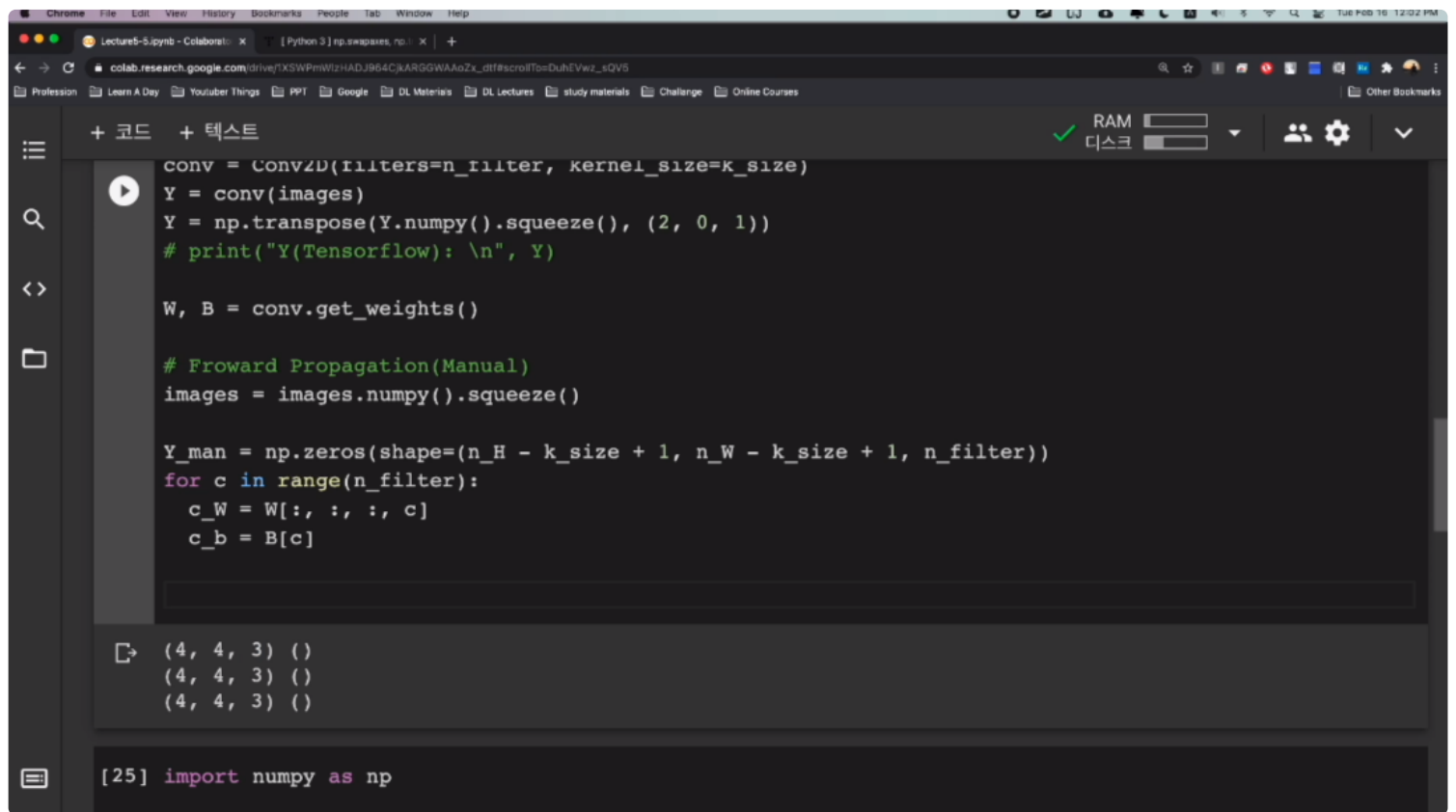
Y_man = np.zeros(shape=(n_H - f + 1, n_W - f + 1, n_filter))
for c in range(n_neuron):
    c_W =

(4, 4, 3, 3) (3,)

[25] import numpy as np

images = np.random.randint(low=0, high=10, size=(2, 3, 4))
```

4 x 4 x 3 x3이 전체 kernel인데 그중에서 4 x 4 x 3 를 하나씩 가져와야한다. 그래서 마지막 3을 기준으로 인덱싱을 해줘야한다.



```
conv = Conv1D(filters=n_filter, kernel_size=k_size)
Y = conv(images)
Y = np.transpose(Y.numpy().squeeze(), (2, 0, 1))
# print("Y(Tensorflow): \n", Y)

W, B = conv.get_weights()

# Froward Propagation(Manual)
images = images.numpy().squeeze()

Y_man = np.zeros(shape=(n_H - k_size + 1, n_W - k_size + 1, n_filter))
for c in range(n_filter):
    c_W = W[:, :, :, c]
    c_b = B[c]
```

(4, 4, 3) ()
(4, 4, 3) ()
(4, 4, 3) ()

```
[25] import numpy as np
```

4 x 4 x 3 metrix가 뽑히는걸 볼 수 있다.