1.검증용CNN기초.ipynb

1. 텐서플로우 데이터셋의 흑백 28,28 손글씨 자료 50개를 갖고와서 CNN에 넣기 위하여 28,28,1로 변형

```
from tensorflow.keras import datasets, layers, models
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))[:50]
test_images = test_images.reshape((10000, 28, 28, 1))[:50]
train_labels=train_labels[:50]
test_labels=test_labels[:50]
```

2. 학습의 편의성을 위하여 50개 이미지를 높이, 너비를 10,10 으로 변경

```
from skimage.transform import resize
train_images = resize(train_images, (50,10, 10,1))
test_images = resize(train_images, (50,10, 10,1))
np.shape(train_images)
plt.imshow(np.hstack(train_images[:10]),cmap='gray')
```

3. 계산검증의 편의성을 위하여 float를 int로 변경

```
pd.DataFrame(np.vstack(train_images[0]).reshape(10,10))
```

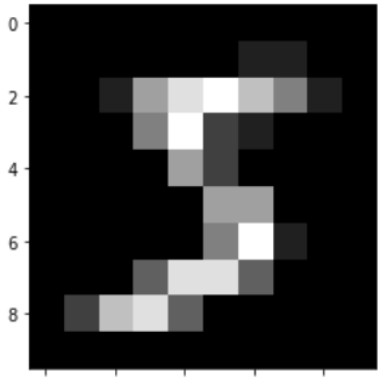
	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.000000e+00	0.000000e+00	3.779813e-09	5.703982e-07	0.000004	1.042152e-05	0.000015	8.009369e-06	1.619335e-08
1	0.000000	3.493966e-07	7.807178e-04	7.999273e-03	2.619184e-02	0.059819	1.155171e-01	0.155306	7.679844e-02	1.519801e-04
2	0.000000	9.857660e-05	1.036307e-01	5.545328e-01	7.500384e-01	0.815209	6.413300e-01	0.491694	1.539254e-01	2.481732e-04
3	0.000000	2.617595e-05	5.656854e-02	4.300613e-01	8.046835e-01	0.233121	1.624373e-01	0.008346	1.218553e-03	5.837837e-07
4	0.000000	9.748591e-09	1.279662e-04	2.502775e-02	5.374982e-01	0.290856	3.380014e-02	0.000305	1.190274e-08	0.000000e+00
5	0.000000	0.000000e+00	6.266862e-09	2.610279e-04	6.696573e-02	0.564458	5.666593e-01	0.047100	1.411354e-05	0.000000e+00
6	0.000000	4.497760e-08	3.063738e-05	3.220381e-03	7.633813e-02	0.401042	8.656932e-01	0.157746	1.073702e-04	0.000000e+00
7	0.000008	5.200728e-03	7.889449e-02	3.434775e-01	7.748787e-01	0.739755	3.174749e-01	0.021589	2.289392e-06	0.000000e+00
8	0.000387	2.102920e-01	6.982965e-01	7.215174e-01	3.777502e-01	0.048582	1.853358e-03	0.000010	1.813034e-10	0.000000e+00
9	0.000029	1.430609e-02	3.384249e-02	1.997844e-02	1.852532e-03	0.000012	1.298005e-08	0.000000	0.000000e+00	0.000000e+00

```
train_images=(train_images*10).astype('int32')
test_images=(test_images*10).astype('int32')
pd.DataFrame(np.vstack(train_images[0]).reshape(10,10))
```

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0
2	0	0	1	5	7	8	6	4	1	0
3	0	0	0	4	8	2	1	0	0	0
4	0	0	0	0	5	2	0	0	0	0
5	0	0	0	0	0	5	5	0	0	0
6	0	0	0	0	0	4	8	1	0	0
7	0	0	0	3	7	7	3	0	0	0
8	0	2	6	7	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0
2	0	0	1	5	7	8	6	4	1	0
3	0	0	0	4	8	2	1	0	0	0
4	0	0	0	0	5	2	0	0	0	0
5	0	0	0	0	0	5	5	0	0	0
6	0	0	0	0	0	4	8	1	0	0
7	0	0	0	3	7	7	3	0	0	0
8	0	2	6	7	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

```
plt.imshow(train_images[0],
cmap='gray')
```



1.검증용CNN기초.ipynb

4. CNN 모델 제작

```
import tensorflow as tf
tf.random.set_seed(123)
model = models.Sequential()
model.add(layers.Conv2D(1, (3, 3), activation='relu', input_shape=(10, 10, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(2, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(2, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

output(10-3+1) ➔8,8, 채널1
output(4,4), 채널1
output(4-3+1) → 2,2, 채널3

output(2*2*3) --> 12
output(2) , 총13*2➔26파람
output(2) , 총13*2➔26파람

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 8, 8, 1)           10
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 1)           0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 2, 2, 2)           20
_________________________________________________________________
flatten_10 (Flatten)         (None, 8)                 0
_________________________________________________________________
dense_20 (Dense)             (None, 2)                 18
_________________________________________________________________
dense_21 (Dense)             (None, 10)                30
=================================================================
Total params: 78
Trainable params: 78
Non-trainable params: 0
_________________________________________________________________
```

```
1  import tensorflow as tf
2
3  get_layer_name = [layer.name for layer in model.layers]
4  get_output = [layer.output for layer in model.layers]
5  get_output
```

```
[<KerasTensor: shape=(None, 8, 8, 1) dtype=float32 (created by layer 'conv2d_19')>,
 <KerasTensor: shape=(None, 4, 4, 1) dtype=float32 (created by layer 'max_pooling2d_8')>,
 <KerasTensor: shape=(None, 2, 2, 2) dtype=float32 (created by layer 'conv2d_20')>,
 <KerasTensor: shape=(None, 8) dtype=float32 (created by layer 'flatten_10')>,
 <KerasTensor: shape=(None, 2) dtype=float32 (created by layer 'dense_20')>,
 <KerasTensor: shape=(None, 10) dtype=float32 (created by layer 'dense_21')>]
```

1.검증용CNN기초.ipynb

5. 모델 w값 setting

```
# 모델의 첫번째 레이어만 사용
visual_model = tf.keras.models.Model(inputs = model.input, outputs = get_output[0])
visual_model.summary()
test_img = np.expand_dims(test_images[0], axis = 0)
feature_maps = visual_model.predict(test_img)
```

0

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6_input (InputLayer)  [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 1)           10
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
```

```
len(visual_model.get_weights()),₩
visual_model.get_weights()[0],₩
visual_model.get_weights()[1]
```

```
(2,
 array([[[[-0.4318703 ]],

         [[ 0.08380783]],

         [[-0.23193197]]],


        [[[ 0.05313462]],

         [[ 0.25442973]],

         [[ 0.33345792]]],


        [[[-0.22227043]],

         [[-0.02131751]],

         [[ 0.17737235]]]], dtype=float32),
 array([0.00019932], dtype=float32))
```

```
import tensorflow as tf
tf.random.set_seed(123)
model = models.Sequential()
model.add(layers.Conv2D(1, (3, 3), activation='relu', input_shape=(10, 10, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(2, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(2, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

검증을 위하여 10,10,1을 input한 filter=1, kernel=3,3, stride=1, Padding 없음만 모델링하면
10,10,1 => 8,8,1 로 출력됨

```
setW =np.array( [[-0.4318703, 0.08380783, -0.23193197],
                 [0.05313462, 0.25442973, 0.33345792],
                 [-0.22227043, -0.02131751, 0.17737235]])
setB= np.array([-0.00019932])


setW=setW.reshape(3,3,1,1)
weights =setW,setB
visual_model.set_weights(weights)
```

모델값 세팅

```
### 검증을 위한 w,b값 출력
w=visual_model.get_weights()[0]
b=visual_model.get_weights()[1]
np.shape(w),np.vstack(w).reshape(3,3),b
```

레이어1개당 w,b 총2개

```
((3, 3, 1, 1),
 array([[-0.4318703 ,  0.08380783, -0.23193197],
        [ 0.05313462,  0.25442973,  0.33345792],
        [-0.22227043, -0.02131751,  0.17737235]], dtype=float32),
 array([-0.00019932], dtype=float32))
```

5. 모델 w값 setting

model.add(layers.Conv2D(1, (3, 3), activation='relu', input_shape=(10, 10, 1)))

10*10*1

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0
2	0	0	1	5	7	8	6	4	1	0
3	0	0	0	4	8	2	1	0	0	0
4	0	0	0	0	5	2	0	0	0	0
5	0	0	0	0	0	5	5	0	0	0
6	0	0	0	0	0	4	8	1	0	0
7	0	0	0	3	7	7	3	0	0	0
8	0	2	6	7	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

커널사이즈 3,3이 1개(filter)

```
### 검증을 위한 w,b값 출력
w=visual_model.get_weights()[0]
np.vstack(w).reshape(3,3)
```

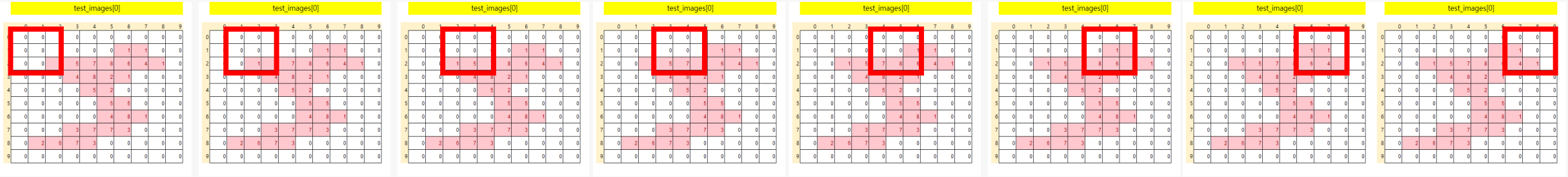
```
array([[-0.4318703 ,  0.08380783, -0.23193197],
       [ 0.05313462,  0.25442973,  0.33345792],
       [-0.22227043, -0.02131751,  0.17737235]], dtype=float32)
```

```
b=visual_model.get_weights()[1]
b
```

```
array([-0.00019932], dtype=float32)
```

합성곱 최종 출력 괄호 빨간색은 음수값
(relu함수에 의해서 0값으로 변경되어야함)

	0	1	2	3	4	5	6	7
0	0.1771730	0.8653449	0.9125492	0.1582048	(0.3289404)	(0.6088908)	(0.9341552)	(0.8574639)
1	0.3332586	2.6310094	4.9929982	4.0092682	2.5325703	2.6713052	1.0994523	0.0348986
2	(0.2321313)	0.2577803	2.9356898	(0.2653025)	(3.6310134)	(3.9638845)	(2.4349872)	(1.6438727)
3	(0.0001993)	(0.9279272)	0.1468658	1.3048444	(1.9646713)	(1.8918026)	(1.5434218)	(0.0001993)
4	(0.0001993)	(0.0001993)	(1.1598592)	2.3317549	2.2812119	(0.2083676)	(1.5340072)	(0.2224698)
5	(0.0001993)	0.5319177	1.1774546	0.5995451	1.7715637	(0.7789230)	(2.1468554)	0.0529353
6	1.0213998	1.6693350	2.1465672	1.7268447	0.9660892	(0.1539183)	(3.2119500)	(0.4320696)
7	2.5094077	3.2710578	1.7278900	(1.1974477)	(2.9730287)	(2.7718679)	(1.2958102)	(0.0001993)



6. 모델 결과 계산 (feature_maps 계산)

test_images[0]

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	0	0	
2	0	0	1	0	8	6	4	1	0	
3	0	0	0	4	8	2	1	0	0	0
4	0	0	0	0	5	2	0	0	0	0
5	0	0	0	0	0	5	5	0	0	0
6	0	0	0	0	0	4	8	1	0	0
7	0	0	0	3	7	7	3	0	0	0
8	0	2	6	7	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

1

w값 = 커널 3,3		
-0.4318703	0.0838078300	-0.2319319700
0.05313462	0.2544297300	0.3334579
-0.2222704	-0.0213175100	0.1773723

2

b값
-0.00019932

3

	0	1	2	3	4	5	6	7
0	0.1771730							
1								
2								
3								
4								
5								
6								
7								

```
# 파이썬에서 계산하여 봅니다.
wVal=np.vstack(w).reshape(3,3)
img=np.vstack(train_images[0]).reshape(10,10)
img[:3,:3]*wVal, np.sum(img[:3,:3]*wVal)+b
```

```
(array([[-0.        ,  0.        , -0.        ],
        [ 0.        ,  0.        ,  0.        ],
        [-0.        , -0.        ,  0.17737235]]),
 array([0.17717303], dtype=float32))
```

1번*2번의합 + 3번

0*-0.4318703 + 0*0.08380783 + 0* -0.23193197 + 0*0.05313462 + 0* 0.25442973 + 0* 0.33345792 + 0*-0.22227043 + 0* -0.02131751 + 1 * 0.17737235 +	[0:3, 0:3] * w값(커널3,3)
+ -0.00019932	B값

6. 모델 결과 계산 (feature_maps 계산)

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0
2	0	0	1	5	7	8	6	4	1	0
3	0	0	0	4	8	2	1	0	0	0
4	0	0	0	0	5	2	0	0	0	0
5	0	0	0	0	0	5	5	0	0	0
6	0	0	0	0	0	4	8	1	0	0
7	0	0	0	3	7	7	3	0	0	0
8	0	2	6	7	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

w값 = 커널 3,3		
-0.43187952	0.08381204	-0.23193252
0.05313142	0.2544299	0.3334544
-0.2222264	-0.02131628	0.1773645

b값
-0.00020013

빨간색 괄호는 - 값임

	0	1	2	3	4	5	6	7
0	0.1771730	0.8653449	0.9125492	0.1582048	(0.3289404)	(0.6088908)	(0.9341552)	(0.8574639)
1	0.3332586	2.6310094	4.9929982	4.0092682	2.5325703	2.6713052	1.0994523	0.0348986
2	(0.2321313)	0.2577803	2.9356898	(0.2653025)	(3.6310134)	(3.9638845)	(2.4349872)	(1.6438727)
3	(0.0001993)	(0.9279272)	0.1468658	1.3048444	(1.9646713)	(1.8918026)	(1.5434218)	(0.0001993)
4	(0.0001993)	(0.0001993)	(1.1598592)	2.3317549	2.2812119	(0.2083676)	(1.5340072)	(0.2224698)
5	(0.0001993)	0.5319177	1.1774546	0.5995451	1.7715637	(0.7789230)	(2.1468554)	0.0529353
6	1.0213998	1.6693350	2.1465672	1.7268447	0.9660892	(0.1539183)	(3.2119500)	(0.4320696)
7	2.5094077	3.2710578	1.7278900	(1.1974477)	(2.9730287)	(2.7718679)	(1.2958102)	(0.0001993)

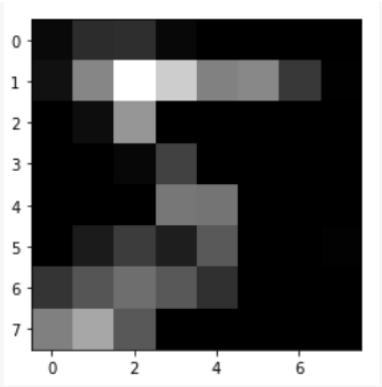
```
# 파이썬에서 계산하여 봅니다.
wVal=np.vstack(w).reshape(3,3)
img=np.vstack(train_images[0]).reshape(10,10)
print('▶--- 0:3, 0:3 에서 스트라이드 1한  높이 0:3, 너비 1:4 인덱싱값 ')
print(img[0:3, 1:4])

print('₩n▶-- img[0:3, 1:4] * w')
print(img[1:4,1:4]*wVal)

print('₩n▶-- img[0:3, 1:4] * w의 합 + b')
np.sum(img[1:4,1:4]*wVal)+b
```

```
▶--- 0:3, 0:3 에서 스트라이드 1한  높이 0:3, 너비 1:4 인덱싱값
[[0 0 0]
 [0 0 0]
 [0 1 5]]

▶-- img[0:3, 1:4] * w
[[-0.          0.         -0.        ]
 [ 0.          0.          0.        ]
 [-0.         -0.02131751  0.88686176]]

▶-- img[0:3, 1:4] * w의 합 + b
array([0.86534494], dtype=float32)
```

7. 모델 최종값 계산- Relu 함수에 의해서 – 값은 0으로 변경됨

(10,10, 1) * w 의 합 + b를 스트라이드 1로한결과

	0	1	2	3	4	5	6	7
0	0.1771730	0.8653449	0.9125492	0.1582048	(0.3289404)	(0.6088908)	(0.9341552)	(0.8574639)
1	0.3332586	2.6310094	4.9929982	4.0092682	2.5325703	2.6713052	1.0994523	0.0348986
2	(0.2321313)	0.2577803	2.9356898	(0.2653025)	(3.6310134)	(3.9638845)	(2.4349872)	(1.6438727)
3	(0.0001993)	(0.9279272)	0.1468658	1.3048444	(1.9646713)	(1.8918026)	(1.5434218)	(0.0001993)
4	(0.0001993)	(0.0001993)	(1.1598592)	2.3317549	2.2812119	(0.2083676)	(1.5340072)	(0.2224698)
5	(0.0001993)	0.5319177	1.1774546	0.5995451	1.7715637	(0.7789230)	(2.1468554)	0.0529353
6	1.0213998	1.6693350	2.1465672	1.7268447	0.9660892	(0.1539183)	(3.2119500)	(0.4320696)
7	2.5094077	3.2710578	1.7278900	(1.1974477)	(2.9730287)	(2.7718679)	(1.2958102)	(0.0001993)



Relu 함수에 의해 <0 값은 0으로 치환한 결과
pd.DataFrame(np.vstack(feature_maps[0]).reshape(8,8))

```
## 최종출력
len(feature_maps[0][0])
plt.imshow(feature_maps[0],cmap='gray')
pd.DataFrame(np.vstack(feature_maps[0]).reshape(8,8))
```

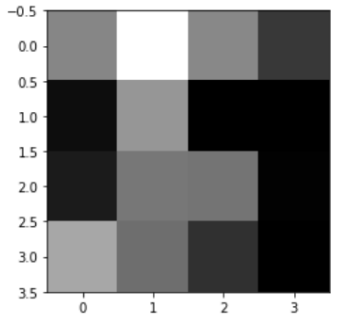
	0	1	2	3	4	5	6	7
0	0.177173	0.865345	0.912549	0.158205	0.000000	0.000000	0.000000	0.000000
1	0.333259	2.631009	4.992998	4.009268	2.532570	2.671305	1.099452	0.034899
2	0.000000	0.257780	2.935690	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.146866	1.304844	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	2.331755	2.281212	0.000000	0.000000	0.000000
5	0.000000	0.531918	1.177455	0.599545	1.771564	0.000000	0.000000	0.052935
6	1.021400	1.669335	2.146567	1.726845	0.966089	0.000000	0.000000	0.000000
7	2.509408	3.271058	1.727890	0.000000	0.000000	0.000000	0.000000	0.000000

8. Pooling 값 확인: max pooling(2,2)는 2*2의 4개의 자료중 가장 큰값을 꺼내는 값임 (stride는 pool값 2)

```
visual_model = tf.keras.models.Model(inputs = model.input, outputs = get_output[1])
visual_model.summary()       # 28-3+1
test_img = np.expand_dims(test_images[0], axis = 0)
feature_maps = visual_model.predict(test_img)
```

```
Model: "model_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17_input (InputLayer) [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 1)           10
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 1)           0
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
```

```
plt.imshow(feature_maps[0],cmap='gray')
plt.show()
pd.DataFrame(np.vstack(feature_maps[0]).reshape(4,4))
```



	0	1	2	3	4	5	6	7
0	0.1771730	0.8653449	0.9125492	0.1582048	0.0000000	0.0000000	0.0000000	0.0000000
1	0.3332586	2.6310094	4.9929982	4.0092682	2.5325703	2.6713052	1.0994523	0.0348986
2	0.0000000	0.2577803	2.9356898	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
3	0.0000000	0.0000000	0.1468658	1.3048444	0.0000000	0.0000000	0.0000000	0.0000000
4	0.0000000	0.0000000	0.0000000	2.3317549	2.2812119	0.0000000	0.0000000	0.0000000
5	0.0000000	0.5319177	1.1774546	0.5995451	1.7715637	0.0000000	0.0000000	0.0529353
6	1.0213998	1.6693350	2.1465672	1.7268447	0.9660892	0.0000000	0.0000000	0.0000000
7	2.5094077	3.2710578	1.7278900	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

	0	1	2	3
0	2.631009	4.992998	2.671305	1.099452
1	0.257780	2.935690	0.000000	0.000000
2	0.531918	2.331755	2.281212	0.052935
3	3.271058	2.146567	0.966089	0.000000

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
print(len(visual_model.get_weights()))
w=visual_model.get_weights()[0]
b=visual_model.get_weights()[1]
print(np.vstack(w).reshape(3,3),b)
print(np.shape(w), np.shape(b))
print('-'*100)
w=visual_model.get_weights()[2]
b=visual_model.get_weights()[3]
print(w,b)
```

```
4
[[-0.4318703   0.08380783 -0.23193197]
 [ 0.05313462  0.25442973  0.33345792]
 [-0.22227043 -0.02131751  0.17737235]] [-0.00019932]
(3, 3, 1, 1) (1,)
----------------------------------------------------------------------------------------------------
[[[[ 0.3674009  -0.35492098]]

  [[ 0.25632307  0.17448828]]

  [[ 0.13600942 -0.39838392]]]


 [[[ 0.2362543   0.11793062]]

  [[-0.46411705  0.05063149]]

  [[ 0.31956223 -0.05979198]]]


 [[[ 0.43049613  0.24405327]]

  [[ 0.37376186 -0.05215767]]

  [[-0.1655781  -0.14671475]]]] [0. 0.]
```

레이어의 일부분만 값을 세팅할수도 있음

```
setW =np.array( [[-0.4318703, 0.08380783, -0.23193197],
                 [0.05313462, 0.25442973, 0.33345792],
                 [-0.22227043, -0.02131751, 0.17737235]])
setB= np.array([-0.00019932])

w2=visual_model.get_weights()[2]
b2=visual_model.get_weights()[3]
setW=setW.reshape(3,3,1,1)

weights =[setW,setB,w2,b2]
visual_model.set_weights(weights)
```

```
len(visual_model.get_weights())
w=visual_model.get_weights()[2]
w, b
```

```
(array([[[[ 0.3674009 , -0.35492098]],

         [[ 0.25632307,  0.17448828]],

         [[ 0.13600942, -0.39838392]]],


        [[[ 0.2362543 ,  0.11793062]],

         [[-0.46411705,  0.05063149]],

         [[ 0.31956223, -0.05979198]]],


        [[[ 0.43049613,  0.24405327]],

         [[ 0.37376186, -0.05215767]],

         [[-0.1655781 , -0.14671475]]]], dtype=float32),
 array([0., 0.], dtype=float32))
```

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
# maxpooling을 거쳐 나온값인 maxpoolResult값하고 3,3 씩 곱하기함
maxpoolResult
```

```
array([[2.6310093 , 4.992998  , 2.6713052 , 1.0994523 ],
       [0.2577803 , 2.93569   , 0.        , 0.        ],
       [0.53191775, 2.331755  , 2.2812119 , 0.0529353 ],
       [3.2710578 , 2.146567  , 0.9660892 , 0.        ]], dtype=float32)
```

```
### 전체 w,b값 확인
visual_model.get_weights()
```

```
[array([[[[-0.4318703 ]],

         [[ 0.08380783]],

         [[-0.23193197]]],


        [[[ 0.05313462]],

         [[ 0.25442973]],

         [[ 0.33345792]]],


        [[[-0.22227043]],

         [[-0.02131751]],

         [[ 0.17737235]]]], dtype=float32),
 array([-0.00019932], dtype=float32),
 array([[[[ 0.3674009 , -0.35492098],
          [ 0.25632307,  0.17448828]],

         [[ 0.2362543 ,  0.11793062],
          [-0.46411705,  0.05063149]],

         [[ 0.43049613,  0.24405327],
          [ 0.37376186, -0.05215767]]],


        [[[ 0.1         , -0.39838392],
          [ 0.11793062  ,  0.05063149]],

         [[-0.05979198  , -0.14671475]]]], dtype=float32),
 array([0., 0.], dtype=float32)]
```

Layer0

$3_{,(10,10,1)}$

Layer1

$2_{,(2,2,3)}$

```
f1=w[:,:,:,0]
f2=w[:,:,:,1]

print('maxpoolResult 값')
print(maxpoolResult[:3,:3])
print('₩n▶피처1값','-'*100, )
print(f1.reshape(3,3))
print('₩n▶피처2값','-'*100, )
print(f2.reshape(3,3))

print('₩n▶[0:3,0:3]*피처1값+b[0]합','-'*50)
np.sum(maxpoolResult[:3,:3]*f1.reshape(3,3))+b[0],₩
np.sum(maxpoolResult[:3,:3]*f2.reshape(3,3))+b[0]
```

```
maxpoolResult 값
[[2.6310093  4.992998   2.6713052 ]
 [0.2577803  2.93569    0.        ]
 [0.53191775 2.331755   2.2812119 ]]

▶피처1값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶피처2값 ----------------------------------------------------------------------------------------------------
[[-0.35492098  0.17448828 -0.39838392]
 [ 0.11793062  0.05063149 -0.05979198]
 [ 0.24405327 -0.05215767 -0.14671475]]

▶[0:3,0:3]*피처1값+b[0]합 ------------------------------------------

(2.0309672, -1.2742374)
```

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
f1=w[:,:,:,0]
f2=w[:,:,:,1]

print('maxpoolResult 값')
print(maxpoolResult[:3,:3])
print('₩n▶피처1값','-'*100, )
print(f1.reshape(3,3))
print('₩n▶피처2값','-'*100, )
print(f2.reshape(3,3))

print('₩n▶[0:3,0:3]*피처1값+b[0]합','-'*50)
print(np.sum(maxpoolResult[:3,:3]*f1.reshape(3,3))+b[0])

print('₩n▶[0:3,0:3]*피처2값+b[1]합','-'*50)
np.sum(maxpoolResult[:3,:3]*f2.reshape(3,3))+b[1]
```

```
maxpoolResult 값
[[2.6310093  4.992998   2.6713052 ]
 [0.2577803  2.93569    0.        ]
 [0.53191775 2.331755   2.2812119 ]]

▶피처1값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶피처2값 ----------------------------------------------------------------------------------------------------
[[-0.35492098  0.17448828 -0.39838392]
 [ 0.11793062  0.05063149 -0.05979198]
 [ 0.24405327 -0.05215767 -0.14671475]]

▶[0:3,0:3]*피처1값+b[0]합 --------------------------------------------------
2.0309672

▶[0:3,0:3]*피처2값+b[1]합 --------------------------------------------------

-1.2742374
```

	0	1	2	3
0	2.631009	4.992998	2.671305	.099452
1	0.257780	2.935690	0.000000	.000000
2	0.531918	2.331755	2.281212	.052935
3	3.271058	2.146567	0.966089	0.000000

필터1

커널사이즈 3*3		
0.3674009	0.256323	0.13600942
0.2362543	-0.46412	0.31956223
0.43049613	0.373762	-0.1655781
bias	0	

결과	
2.030967	5.209931
2.670163022	1.872805

필터2

커널사이즈 3*3		
-0.35492098	0.174488	-0.3983839
0.11793062	0.050631	-0.059792
0.24405327	-0.05216	-0.1467148
bias	0	

결과	
-1.274237336	-0.95548
1.009756132	-0.18113

Relu함수 적용
<0작아서 0으로 됨

```
feature_maps
```

```
array([[[2.0309675, 0.       ],
        [5.2099314, 0.       ]],

       [[2.6701632, 1.0097562],
        [1.8728051, 0.       ]]], dtype=float32)
```

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
f1=w[:,:,:,0]
f2=w[:,:,:,1]

print('maxpoolResult 값')
print(maxpoolResult[0:3,1:4])
print('₩n▶피처1값','-'*100, )
print(f1.reshape(3,3))
print('₩n▶피처2값','-'*100, )
print(f1.reshape(3,3))

print('₩n▶[0:3,0:3]*피처1값+b[0]합','-'*50)
print(np.sum(maxpoolResult[0:3,1:4]*f1.reshape(3,3))+b[0])

print('₩n▶[0:3,0:3]*피처2값+b[1]합','-'*50)
np.sum(maxpoolResult[0:3,1:4]*f2.reshape(3,3))+b[0]
```

```
maxpoolResult 값
[[4.992998  2.6713052 1.0994523]
 [2.93569   0.        0.       ]
 [2.331755  2.2812119 0.0529353]]

▶피처1값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶피처2값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶[0:3,0:3]*피처1값+b[0]합 --------------------------------------------------
5.2099314

▶[0:3,0:3]*피처2값+b[1]합 --------------------------------------------------

-0.9554814
```

	0	1	2	3
0	2.631009	4.992998	2.671305	1.099452
1	0.257780	2.935690	0.000000	0.000000
2	0.531918	2.331755	2.281212	0.052935
3	3.271058	2.146567	0.966089	0.000000

필터1

커널사이즈 3*3		
0.3674009	0.256323	0.13600942
0.2362543	-0.46412	0.31956223
0.43049613	0.373762	-0.1655781
bias	0	

결과	
2.030967392	**5.209931**
2.670163022	1.872805149

필터2

커널사이즈 3*3		
-0.35492098	0.174488	-0.3983839
0.11793062	0.050631	-0.059792
0.24405327	-0.05216	-0.1467148
bias	0	

결과	
-1.27423734	**-0.955481**
1.009756132	-0.181128754

<0
→ 0으로 됨

```
feature_maps
```

```
array([[[[2.0309675, 0.       ],
         [5.2099314, 0.       ]]

        [[2.6701632, 1.0097562],
         [1.8728051, 0.       ]]]], dtype=float32)
```

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
f1=w[:,:,:,0]
f2=w[:,:,:,1]

print('maxpoolResult 값')
print(maxpoolResult[1:4,0:3])
print('₩n▶피처1값','-'*100, )
print(f1.reshape(3,3))
print('₩n▶피처2값','-'*100, )
print(f2.reshape(3,3))


print('₩n▶[0:3,0:3]*피처1값+b[0]합','-'*50)
print(np.sum(maxpoolResult[1:4,0:3]*f1.reshape(3,3))+b[0])

print('₩n▶[0:3,0:3]*피처2값+b[1]합','-'*50)
np.sum(maxpoolResult[1:4,0:3]*f2.reshape(3,3))+b[0]
```

```
maxpoolResult 값
[[0.2577803  2.93569    0.        ]
 [0.53191775 2.331755   2.2812119 ]
 [3.2710578  2.146567   0.9660892 ]]

▶피처1값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶피처2값 ----------------------------------------------------------------------------------------------------
[[-0.35492098  0.17448828 -0.39838392]
 [ 0.11793062  0.05063149 -0.05979198]
 [ 0.24405327 -0.05215767 -0.14671475]]

▶[0:3,0:3]*피처1값+b[0]합 --------------------------------------------------
2.6701632

▶[0:3,0:3]*피처1값+b[0]합 --------------------------------------------------

1.0097562
```

	0	1	2	3
0	2.631009	4.992998	2.671305	1.099452
1	0.257780	2.935690	0.000000	0.000000
2	0.531918	2.331755	2.281212	0.052935
3	3.271058	2.146567	0.966089	0.000000

필터1

커널사이즈 3*3		
0.3674009	0.256323	0.13600942
0.2362543	-0.46412	0.31956223
0.43049613	0.373762	-0.1655781
bias	0	

결과	
2.030967392	5.209930882
2.670163	1.872805149

필터2

커널사이즈 3*3		
-0.35492098	0.174488	-0.3983839
0.11793062	0.050631	-0.059792
0.24405327	-0.05216	-0.1467148
bias	0	

결과	
-1.27423734	-0.955481445
1.009756	-0.181128754

```
feature_maps
```

```
array([[[[2.0309675, 0.       ],
         [5.2099314, 0.       ]],

        [[2.6701632, 1.0097562],
         [1.8728051, 0.       ]]]], dtype=float32)
```

9. Layer2번층 : 풀링층을 거친 자료를 다시 Conv2d할때

```
f1=w[:,:,:,0]
f2=w[:,:,:,1]

print('maxpoolResult 값')
print(maxpoolResult[1:4,1:4])
print('₩n▶피처1값','-'*100, )
print(f1.reshape(3,3))
print('₩n▶피처2값','-'*100, )
print(f1.reshape(3,3))

print('₩n▶[0:3,0:3]*피처1값+b[0]합','-'*50)
print(np.sum(maxpoolResult[1:4,1:4]*f1.reshape(3,3))+b[0])

print('₩n▶[0:3,0:3]*피처2값+b[1]합','-'*50)
np.sum(maxpoolResult[1:4,1:4]*f2.reshape(3,3))+b[0]
```

```
maxpoolResult 값
[[2.93569   0.        0.       ]
 [2.331755  2.2812119 0.0529353]
 [2.146567  0.9660892 0.       ]]

▶피처1값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶피처2값 ----------------------------------------------------------------------------------------------------
[[ 0.3674009   0.25632307  0.13600942]
 [ 0.2362543  -0.46411705  0.31956223]
 [ 0.43049613  0.37376186 -0.1655781 ]]

▶[0:3,0:3]*피처1값+b[0]합 --------------------------------------------------
1.8728052

▶[0:3,0:3]*피처2값+b[1]합 --------------------------------------------------

-0.1811288
```

	0	1	2	3
0	2.631009	4.992998	2.671305	1.099452
1	0.257780	2.935690	0.000000	0.000000
2	0.531918	2.331755	2.281212	0.052935
3	3.271058	2.146567	0.966089	0.000000

필터1

커널사이즈 3*3		
0.3674009	0.256323	0.13600942
0.2362543	-0.46412	0.31956223
0.43049613	0.373762	-0.1655781
bias	0	

결과	
2.030967392	5.209930882
2.670163022	**1.872805**

필터2

커널사이즈 3*3		
-0.35492098	0.174488	-0.3983839
0.11793062	0.050631	-0.059792
0.24405327	-0.05216	-0.1467148
bias	0	

결과	
-1.27423734	-0.955481445
1.009756132	**-0.181129**

<0
→ 0으로 됨

```
feature_maps
```

```
array([[[[2.0309675, 0.       ],
         [5.2099314, 0.       ]],

        [[2.6701632, 1.0097562],
         [1.8728051, 0.       ]]]], dtype=float32)
```

8. flatten()

```
1 feature_maps
```

```
array([[[[2.0309675, 0.       ],
         [5.2099314, 0.       ]],

        [[2.6701632, 1.0097562],
         [1.8728051, 0.       ]]]], dtype=float32)
```

```
4]: 1 ###
    2 visual_model = tf.keras.models.Model(inputs = model.input, outputs = get_output[3])
    3 visual_model.summary()
    4 test_img = np.expand_dims(test_images[0], axis = 0)
    5 feature_maps = visual_model.predict(test_img)
    6
```

```
Model: "model_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6_input (InputLayer)  [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 1)           10
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 1)           0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 2, 2, 2)           20
_________________________________________________________________
flatten_3 (Flatten)          (None, 8)                 0
=================================================================
Total params: 30
Trainable params: 30
Non-trainable params: 0
_________________________________________________________________
```

```
5]: 1 len(visual_model.get_weights())
5]: 4
```

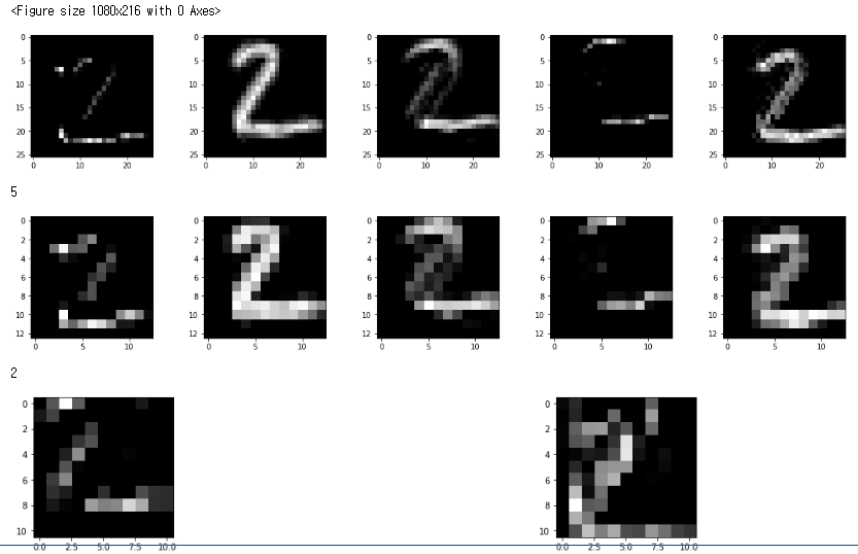
```
7]: 1 feature_maps
7]: array([[2.0309675, 0.       , 5.2099314, 0.       , 2.6701632, 1.0097562,
            1.8728051, 0.       ]], dtype=float32)
```

2. CNN 레이어별 출력물 확인.ipynb

활성화함수에 따른 결과 확인

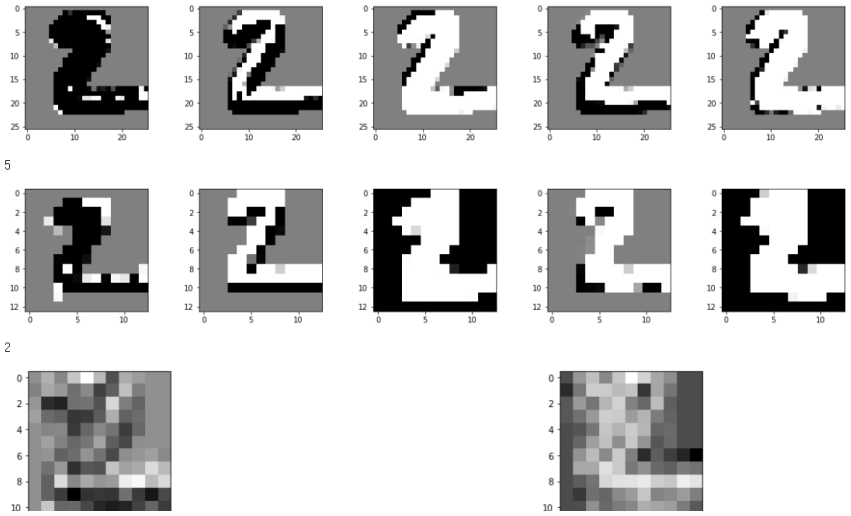
```
model = models.Sequential()
model.add(layers.Conv2D(5, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(2, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(2, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))


get_layer_name = [layer.name for layer in model.layers]
get_output = [layer.output for layer in model.layers]
get_output
```

relu




```
model = models.Sequential()
model.add(layers.Conv2D(5, (3, 3), activation='sigmoid' ,input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(2, (3, 3),activation='sigmoid'))

model.add(layers.Flatten())
model.add(layers.Dense(2, activation='sigmoid'))
model.add(layers.Dense(10, activation='softmax'))

get_layer_name = [layer.name for layer in model.layers]
get_output = [layer.output for layer in model.layers]
get_output
```

sigmoid



2. CNN 레이어별 출력물 확인.ipynb

활성화함수에 따른 결과 확인

```
model = models.Sequential()
model.add(layers.Conv2D(5, (3, 3), input_shape=(28, 28, 1)))
model.add(layers.LeakyReLU(1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(2, (3, 3)))
model.add(layers.LeakyReLU())

model.add(layers.Flatten())
model.add(layers.Dense(2,activation='relu'))
#model.add(layers.LeakyReLU())
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_48"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_95 (Conv2D)           (None, 26, 26, 5)         50
_________________________________________________________________
leaky_re_lu_36 (LeakyReLU)   (None, 26, 26, 5)         0
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 13, 13, 5)         0
_________________________________________________________________
conv2d_96 (Conv2D)           (None, 11, 11, 2)         92
_________________________________________________________________
leaky_re_lu_37 (LeakyReLU)   (None, 11, 11, 2)         0
_________________________________________________________________
flatten_47 (Flatten)         (None, 242)               0
_________________________________________________________________
dense_91 (Dense)             (None, 2)                 486
_________________________________________________________________
dense_92 (Dense)             (None, 10)                30
=================================================================
Total params: 658
Trainable params: 658
Non-trainable params: 0
_________________________________________________________________
```

생성신경망에 필요한 기초지식

확률분포 개념

딥러닝 이야기를 꺼내기 전에 확률분포가 무엇인지를 알아보도록 하겠습니다! '**확률분포(probability distribution)**'는
확률 변수가 특정한 값을 가질 확률을 나타내는 함수를 의미함.
가장 쉬운 예로는 주사위를 던지는 상황이 있습니다. 여기서 확률변수 X는 주사위를 던져 나올 수 있는 눈의 수로, 1부터 6까지의 자연수임.
그리고 각각의 확률변수 값이 나올 확률이 존재하는데, 이 경우에는 모두 1/6로 동일합니다.

주사위를 던졌을 때 6번 중에서 다음과 같은 횟수의 숫자가 나왔다고 하자.

X	1	2	3	4	5	6
P(X)	1/6	1/6	1/6	0/6	0/6	3/6

이를 그래프로 표현한다면, 이 비슷한 그림이 나올 것이다.

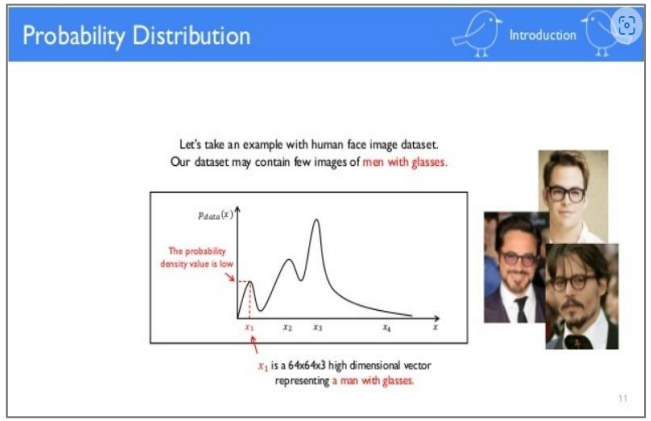
이상형확률분포

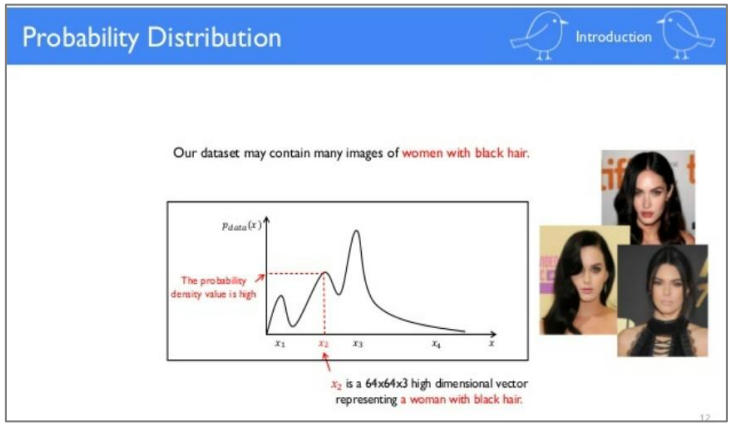
주사위의 눈과 같이 확률변수 X가 이산적으로 나타나는 확률분포를 우리는 이산형 확률분포,
또는 확률질량함수라고 함

연속형확률분포

키, 몸무게와 같은 데이터는 이산형으로 나타내기가 어렵습니다. 이런 데이터를 나타내는 확률분포를
연속형 확률분포, 또는 '확률 밀도 함수 ' 라고 함.

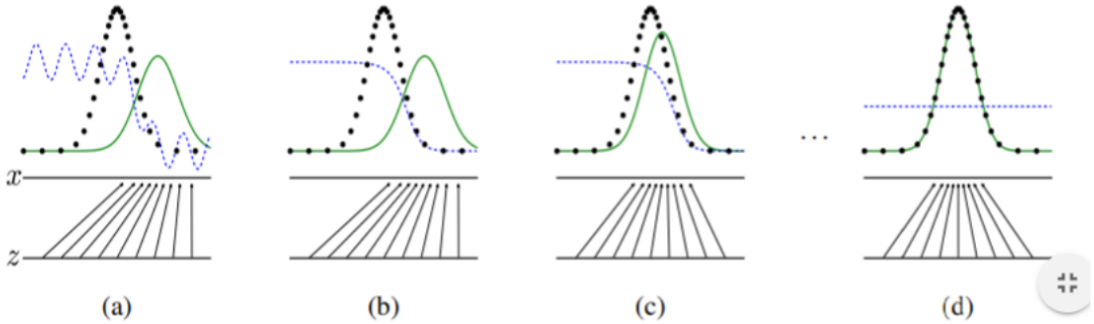
GAN에는 기본적으로
Generator와
Discriminator,
2개의 네트워크가 존재하며그 중
확률분포와 관련이 있는 모델은
바로 Generator(생성자) 임





https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

생성신경망에 필요한 기초



검정 점선이 학습 데이터의 분포를, 초록 실선이 모델의 분포를 나타냅니다.

학습이 되지 않은 상태(a)의 generator는 z를 x_4와 같이 실제 데이터셋에 거의 존재하지 않는 사진으로 만들어냅니다. Discriminator의 피드백을 받으며 점차 generator는 실제 데이터 셋에 존재하는 사진들을 만들어냅니다.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))]$$

생성자 (generator)

판별자 (Discriminator)

좋아! 이제 앞서 정의한 minimax problem을 잘 풀기만 하면 (즉, global optimal을 찾으면), generator가 만드는 probability distribution(p_g)이 data distribution(p_{data})과 정확히 일치하도록 할 수 있다는 것을 알았습니다. 결국, Generator로 부터 뽑은 sample을 Discriminator가 실제와 구별할 수 없게 된다는 것이죠.

다만, "**어떤 모델**을 사용하고 **어떤 알고리즘**을 사용해야 이 문제를 "***잘***" 풀어줄 것이냐?"는 또 별개의 문제인데...

https://jaejunyoo.blogspot.com/2017/01/generative-adversarial-nets-2.html

https://comlini8-8.tistory.com/9?category=911110