

The background of the slide is a light gray with a subtle, abstract pattern of interconnected nodes and lines, resembling a neural network or a complex graph. The nodes are small circles of varying shades of gray, and the lines are thin, light gray lines connecting them. The overall effect is a modern, technical, and interconnected aesthetic.

MLDL-I

Machine Learning and Deep Learning - I

tmax	tmin	rain	tmax_tomorrow
------	------	------	---------------

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0
50.0	38.0	0.00	52.0
52.0	43.0	0.00	56.0
56.0	49.0	0.24	54.0
54.0	50.0	0.40	57.0
57.0	50.0	0.00	57.0
57.0	50.0	0.31	58.0
58.0	52.0	0.05	59.0
59.0	54.0	0.25	58.0
58.0	53.0	1.94	56.0
56.0	51.0	0.63	61.0
61.0	56.0	0.62	59.0
59.0	54.0	0.00	58.0
58.0	53.0	0.00	60.0
60.0	53.0	0.14	60.0
60.0	53.0	1.21	61.0

Batch_size = 5

Epoch 2, Step 1:

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0

```
print(len(df))
```

13509

Dataset length, m = 13509

Calculate Prediction

$$\hat{y} = w^T x + b$$

Estimate Error

$$J(w, b) = (\hat{y} - y_{actual})^2$$

Update Parameters

$$w = w - \frac{\partial J}{\partial w} \quad b = b - \frac{\partial J}{\partial b}$$

tmax	tmin	rain	tmax_tomorrow
------	------	------	---------------

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0
50.0	38.0	0.00	52.0
52.0	43.0	0.00	56.0
56.0	49.0	0.24	54.0
54.0	50.0	0.40	57.0
57.0	50.0	0.00	57.0
57.0	50.0	0.31	58.0
58.0	52.0	0.05	59.0
59.0	54.0	0.25	58.0
58.0	53.0	1.94	56.0
56.0	51.0	0.63	61.0
61.0	56.0	0.62	59.0
59.0	54.0	0.00	58.0
58.0	53.0	0.00	60.0
60.0	53.0	0.14	60.0
60.0	53.0	1.21	61.0

Batch_size = 5

Epoch 2, Step 1:

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0

```
print(len(df))
```

13509

Dataset length, m = 13509

Calculate Prediction

$$\hat{y} = w^T x + b$$

Estimate Error

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$$w = w - \frac{\partial J}{\partial w} \quad b = b - \frac{\partial J}{\partial b}$$

tmax	tmin	rain	tmax_tomorrow
------	------	------	---------------

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0
50.0	38.0	0.00	52.0
52.0	43.0	0.00	56.0
56.0	49.0	0.24	54.0
54.0	50.0	0.40	57.0
57.0	50.0	0.00	57.0
57.0	50.0	0.31	58.0
58.0	52.0	0.05	59.0
59.0	54.0	0.25	58.0
58.0	53.0	1.94	56.0
56.0	51.0	0.63	61.0
61.0	56.0	0.62	59.0
59.0	54.0	0.00	58.0
58.0	53.0	0.00	60.0
60.0	53.0	0.14	60.0
60.0	53.0	1.21	61.0

Batch_size = 5

Epoch 2, Step 1:

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0

```
print(len(df))
```

13509

Dataset length, m = 13509

Calculate Prediction

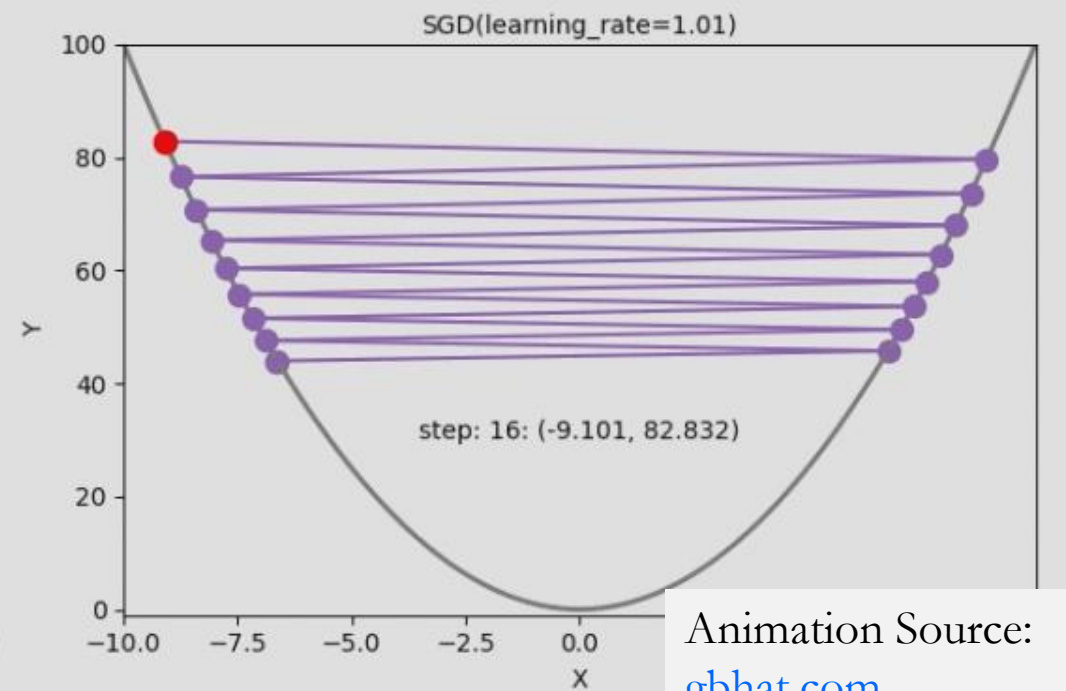
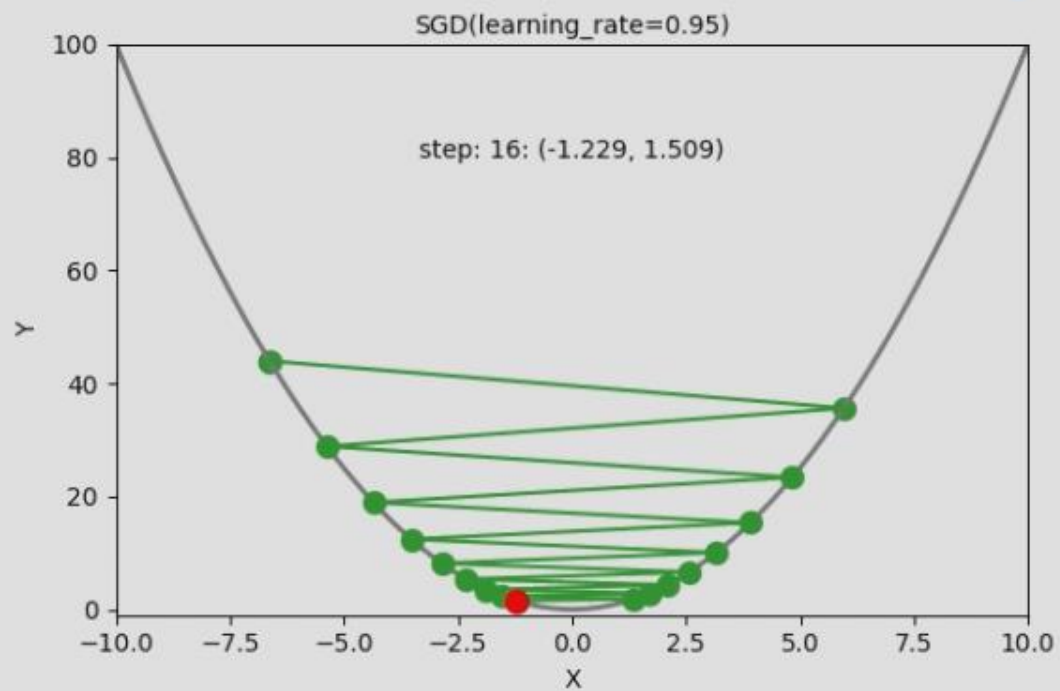
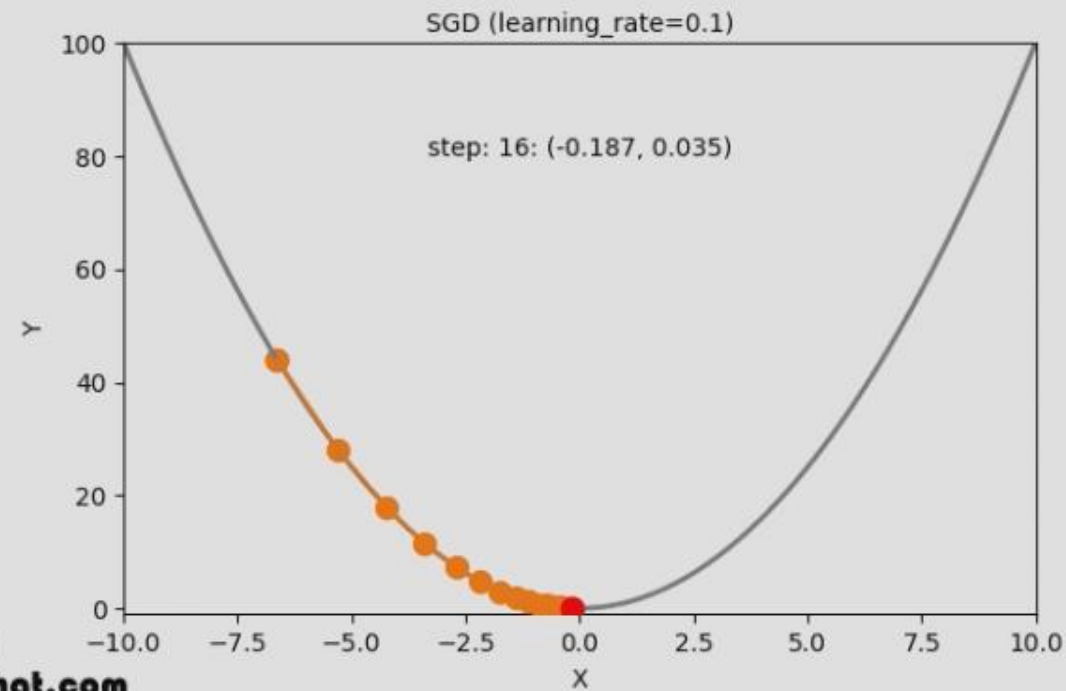
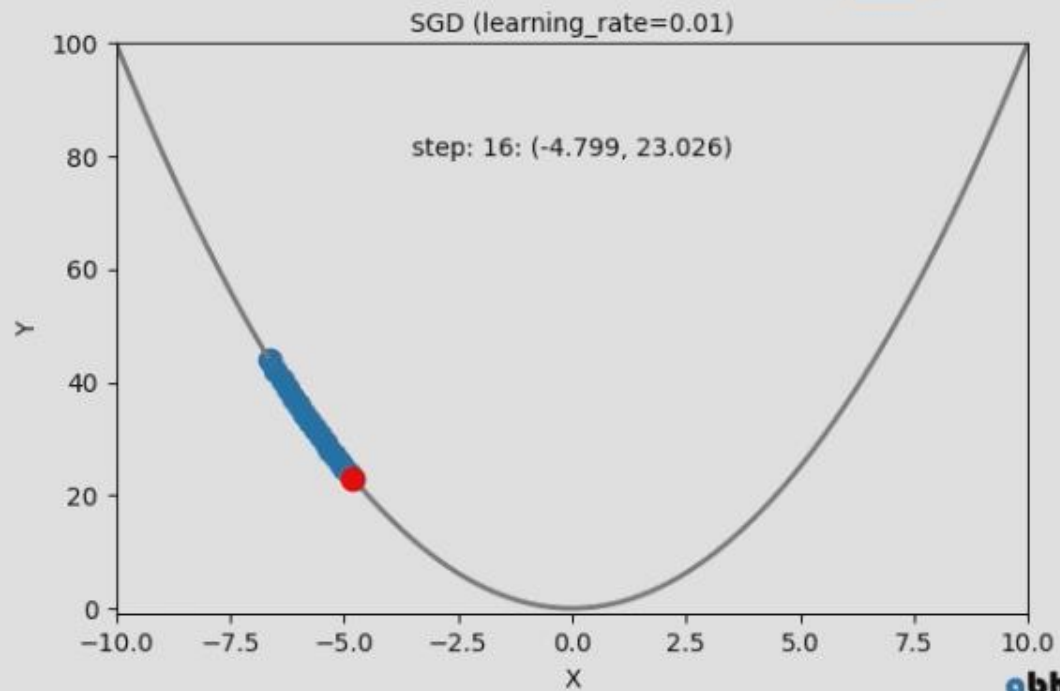
$$\hat{y} = w^T x + b$$

Estimate Error

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$$w = w - \alpha \frac{\partial J}{\partial w} \quad b = b - \alpha \frac{\partial J}{\partial b}$$



Animation Source:
gbhat.com

tmax	tmin	rain	tmax_tomorrow
------	------	------	---------------

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0
50.0	38.0	0.00	52.0
52.0	43.0	0.00	56.0
56.0	49.0	0.24	54.0
54.0	50.0	0.40	57.0
57.0	50.0	0.00	57.0
57.0	50.0	0.31	58.0
58.0	52.0	0.05	59.0
59.0	54.0	0.25	58.0
58.0	53.0	1.94	56.0
56.0	51.0	0.63	61.0
61.0	56.0	0.62	59.0
59.0	54.0	0.00	58.0
58.0	53.0	0.00	60.0
60.0	53.0	0.14	60.0
60.0	53.0	1.21	61.0

Batch_size = 5

Epoch 2, Step 1:

60.0	35.0	0.00	52.0
52.0	39.0	0.00	52.0
52.0	35.0	0.00	53.0
53.0	36.0	0.00	52.0
52.0	35.0	0.00	50.0

```
print(len(df))
```

13509

Dataset length, m = 13509

Calculate Prediction

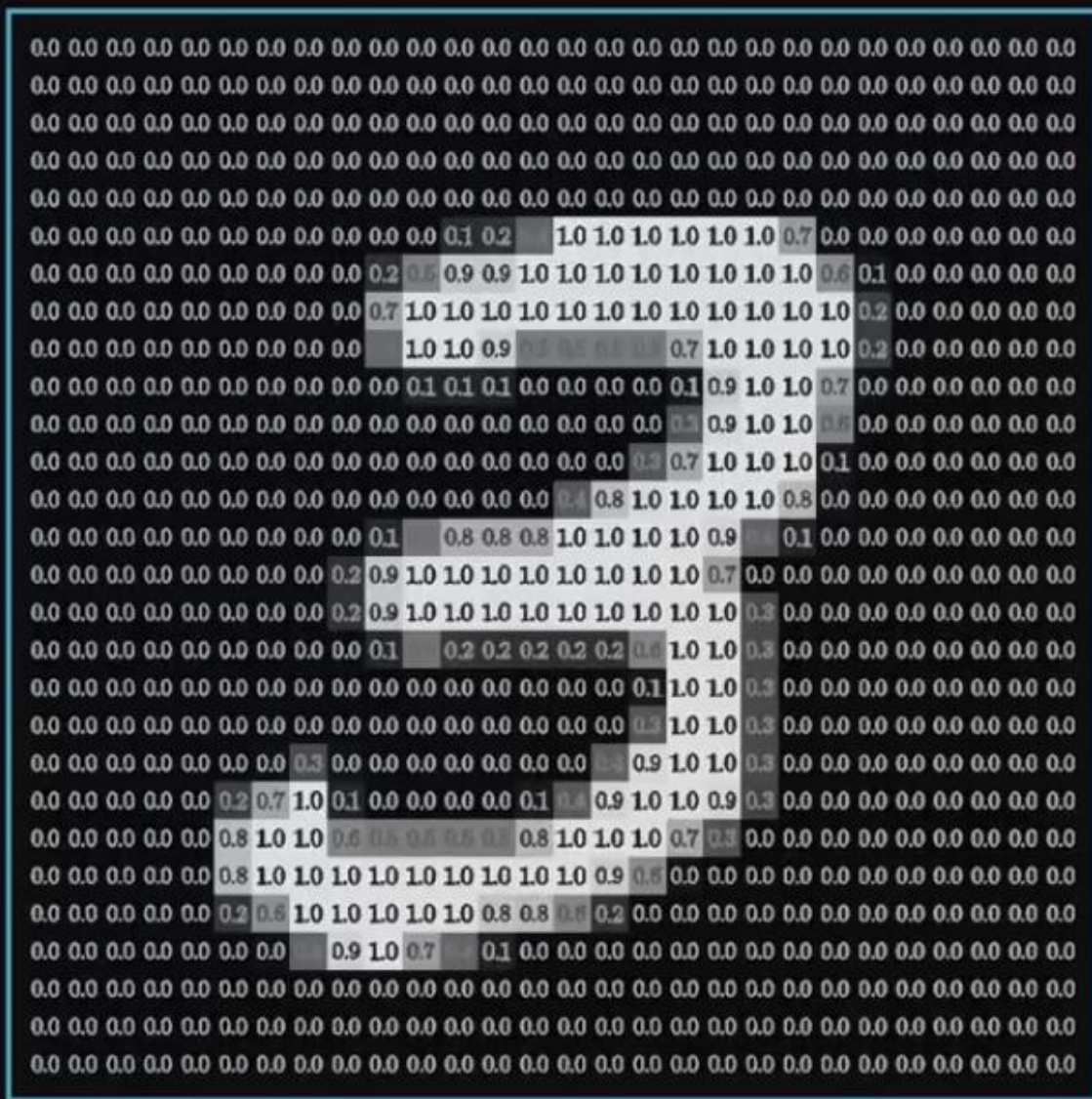
$$\hat{y} = w^T x + b$$

Estimate Error

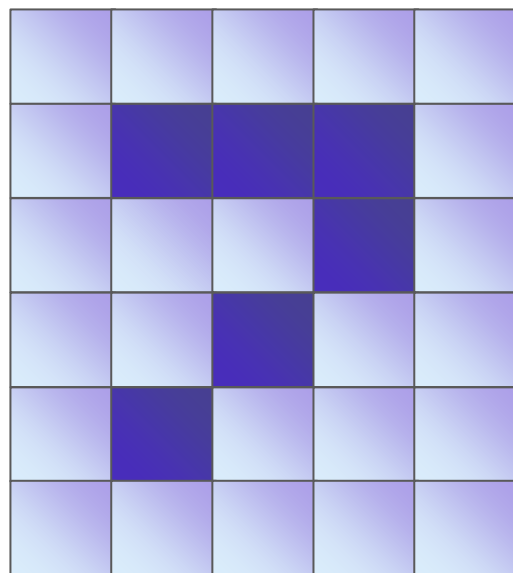
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_{i_{actual}})^2$$

Update Parameters

$$w = w - \alpha \frac{\partial J}{\partial w} \quad b = b - \alpha \frac{\partial J}{\partial b}$$

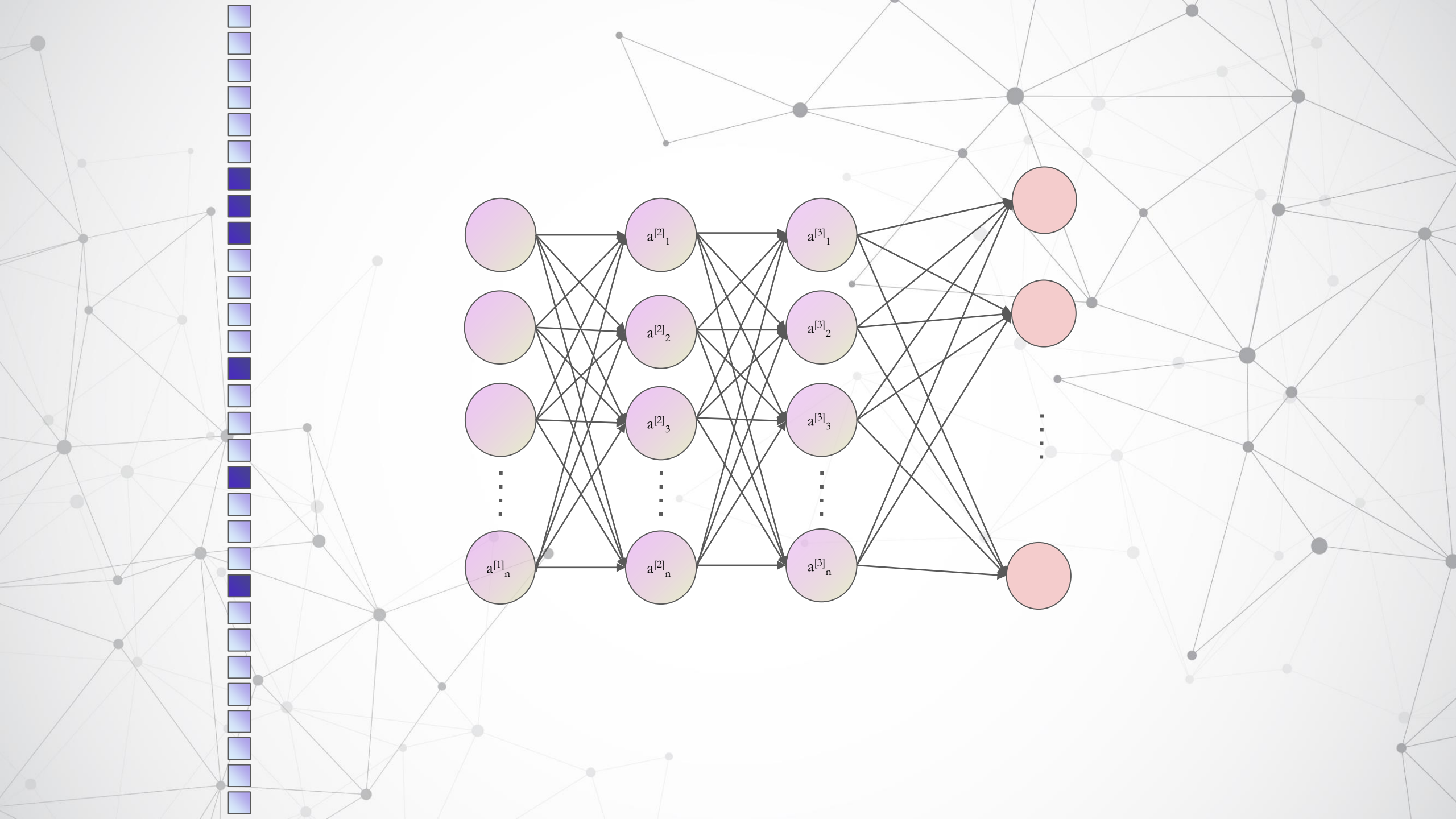


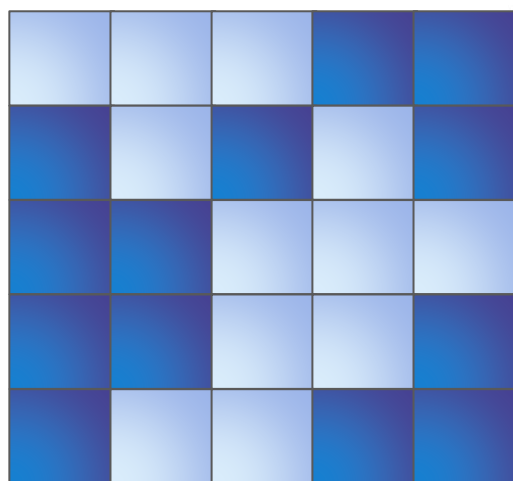
0
1
2
3
4
5
6
7
8
9



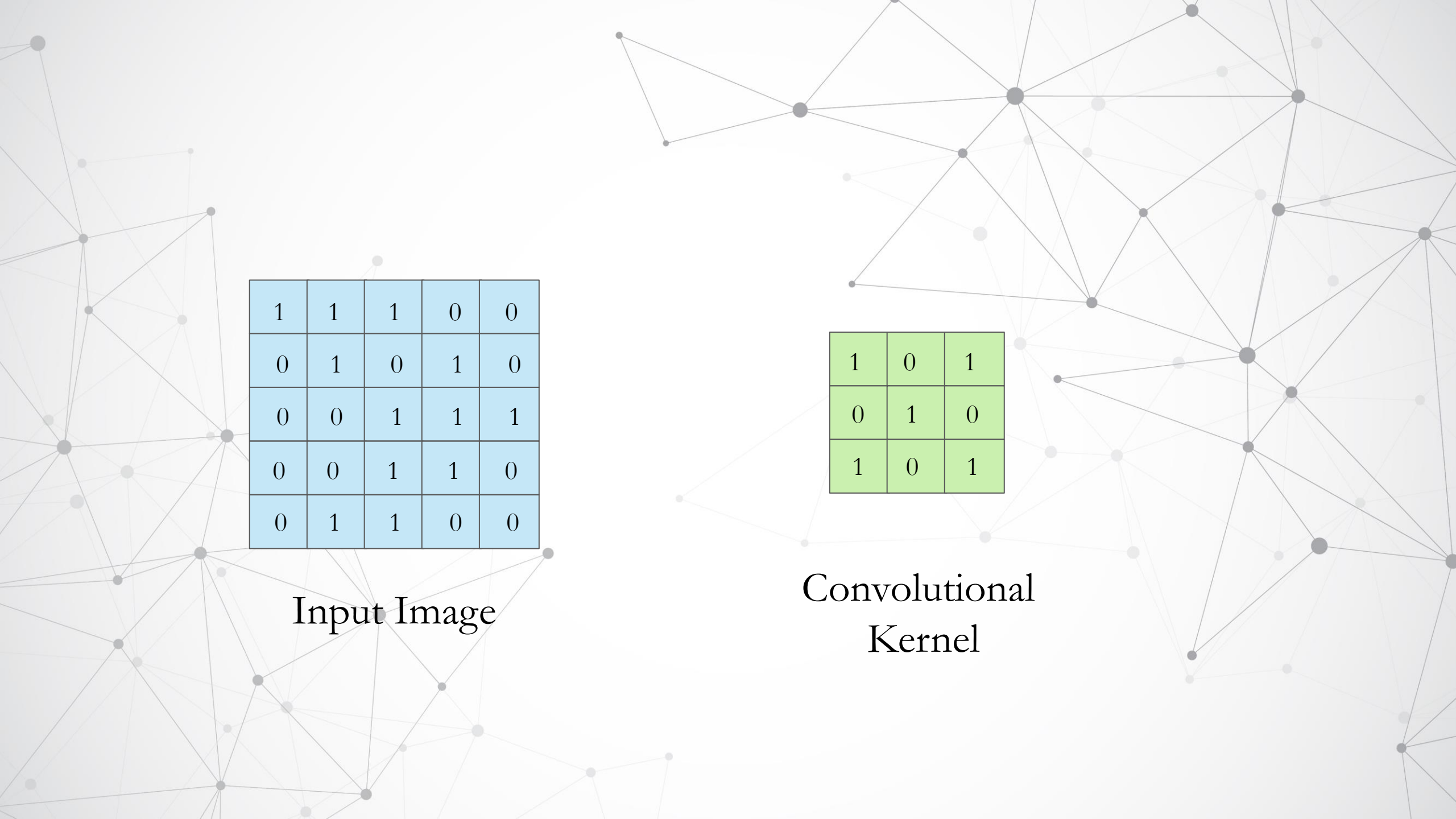
Input Image

```
tf.keras.layers.Flatten()
```



Input Image

A background network diagram consisting of a complex web of interconnected nodes and edges. The nodes are represented by small gray circles of varying sizes, and the edges are thin gray lines. The network is dense and spans the entire width and height of the image.

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

Input Image

1	0	1
0	1	0
1	0	1

Convolutional
Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Animation Source:
[Link](#)

1	0	1
0	1	0
1	0	1

Convolutional Filter



400 x 600

-1	0	1
-2	0	2
-1	0	1

3 x 3



Original Image

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

Sobel X

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

Sobel Y

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

Laplacian

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

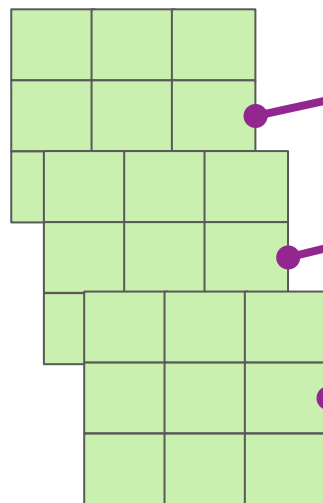
Prewitt X

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

Prewitt Y

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

Intro to CNN



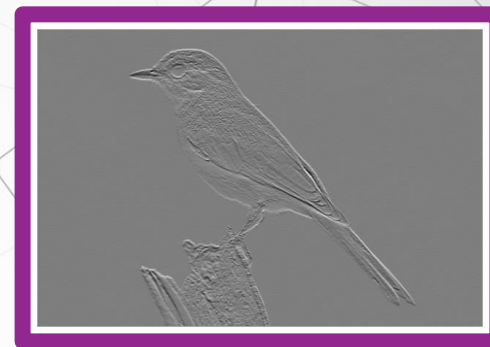
Horizontal Edge

Vertical Edge

Changes in Value

⋮

Angular Edge



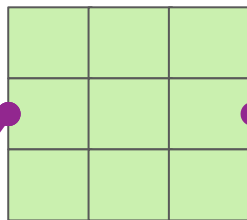
Horizontal Edge

Vertical Edge

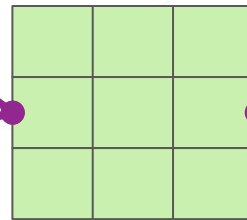
Changes in Value

⋮

Angular Edge

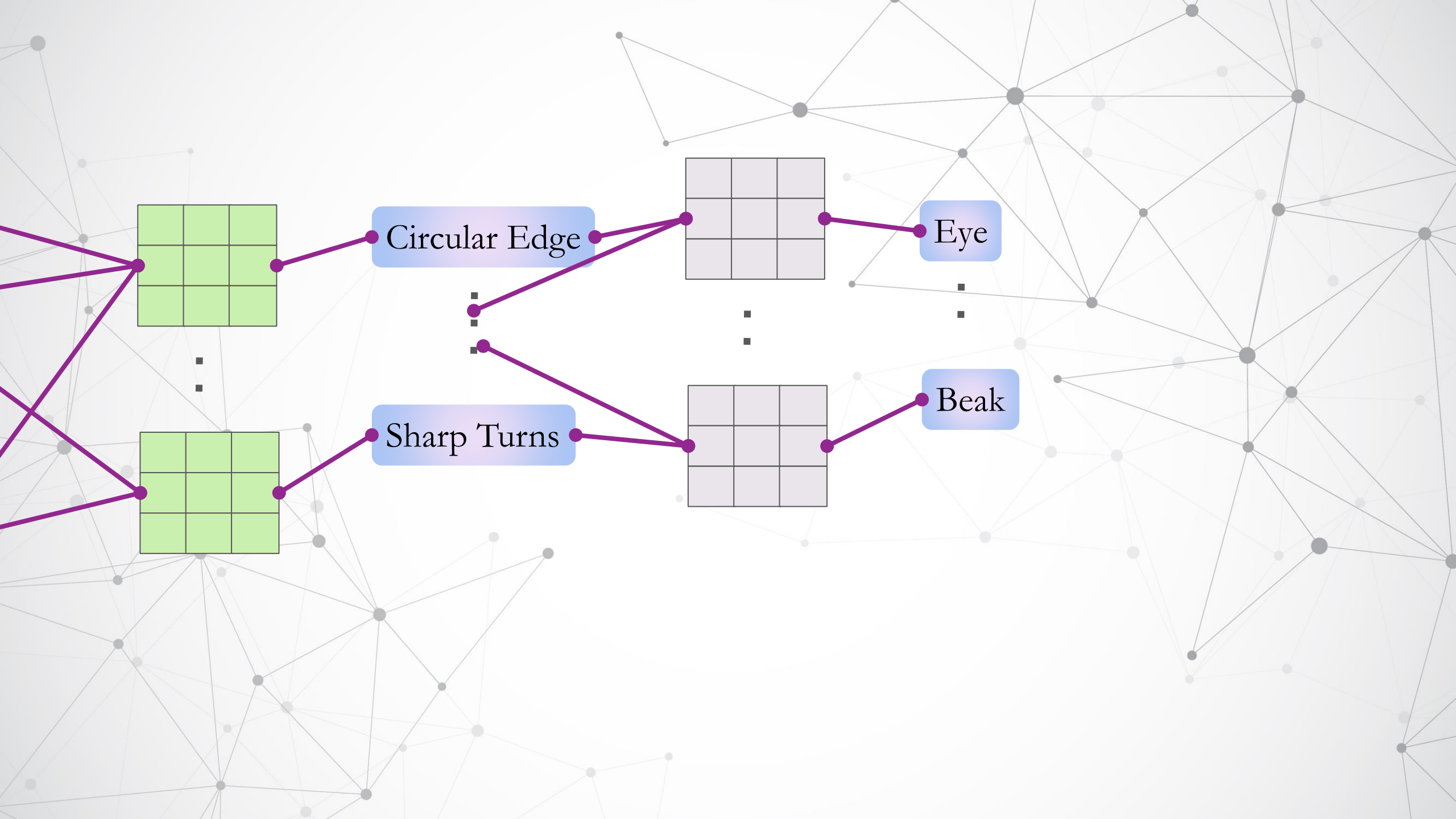


⋮



Circular Edge

Sharp Turns

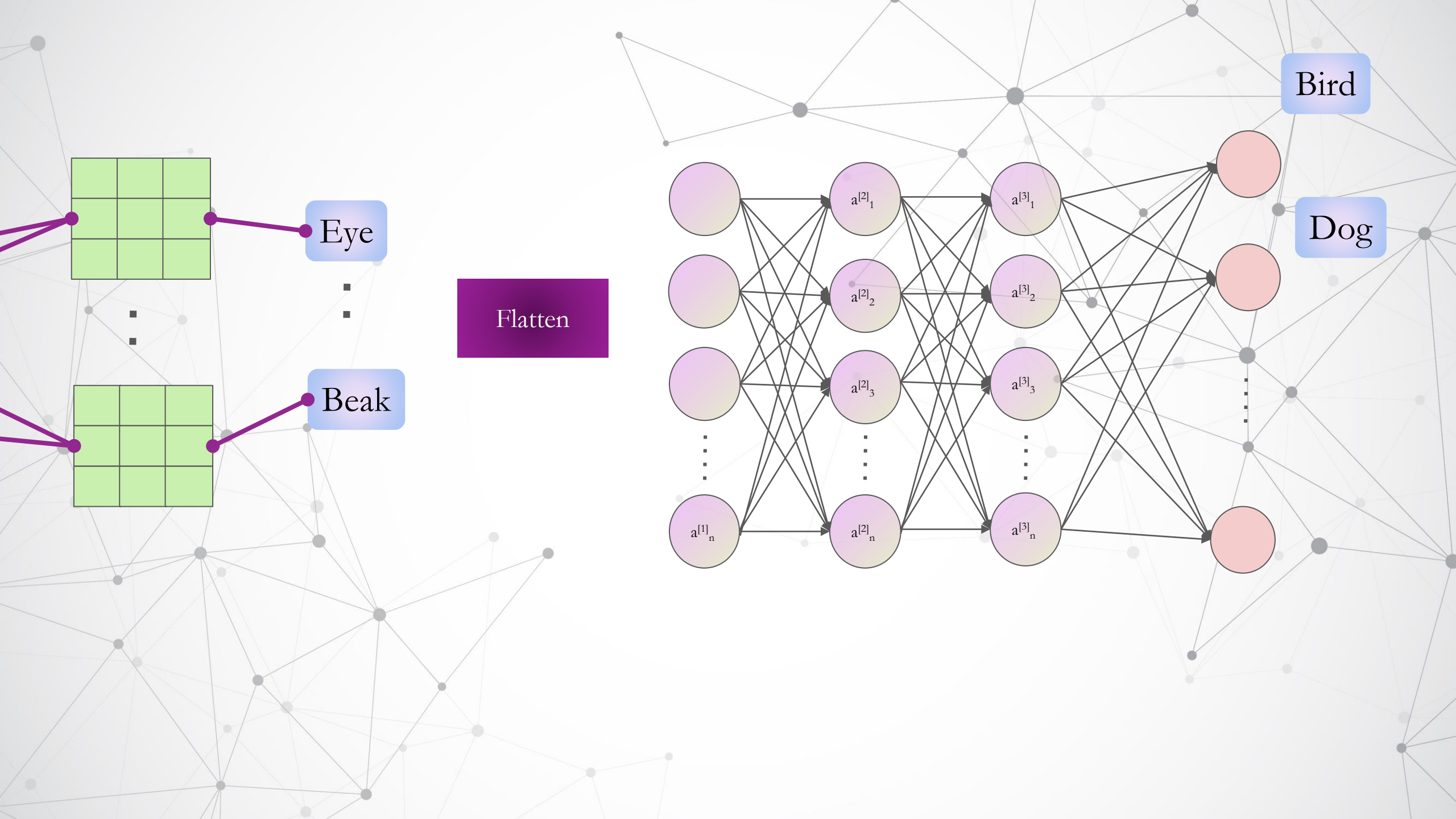


Circular Edge

Eye

Sharp Turns

Beak

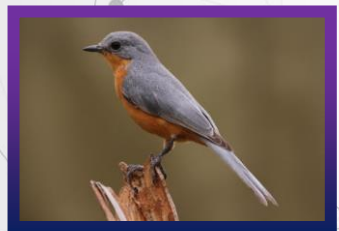


Has_Tail	Number_of_Legs	Has_Fur	Lives_in_Water	Can_Fly	Lays_Eggs	Is_Mammal
----------	----------------	---------	----------------	---------	-----------	-----------

True	4	True	False	False	False	True
True	4	True	False	False	False	True
False	8	False	False	False	True	False
False	0	False	False	False	True	False
False	2	False	False	True	True	False
True	2	True	False	False	False	True
True	4	False	False	False	False	True
False	0	False	True	False	True	False
False	2	False	False	False	True	False
True	4	False	False	False	False	True

Animal

Cat
Dog
Spider
Snake
Parrot
Kangaroo
Elephant
Fish
Chicken
Cow

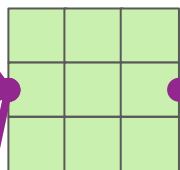


Horizontal Edge

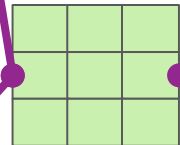
Vertical Edge

Changes in Value

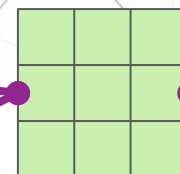
Angular Edge



⋮



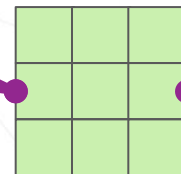
Circular Edge



⋮

⋮

Sharp Turns



Eye

Beak

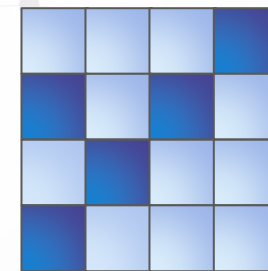
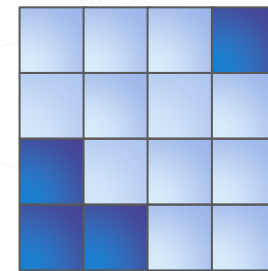
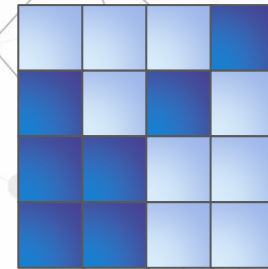
DNN

Bird

Intro to CNN



?	?	?
?	?	?
?	?	?



```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

- Filter
- Kernel Size
- Padding
- Strides

```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Kernel Size



256 x 256 x 1

1	0	1
0	1	0
1	0	1

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

```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Stride

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

Stride (1,1)

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

Stride (2,2)


```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Stride

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

Stride (1,1)

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

Stride (2,2)

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

1	0	1
0	1	0
1	0	1

Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

Stride

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

Stride (2,2)



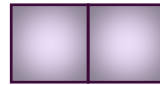
Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

Stride

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

Stride (2,2)



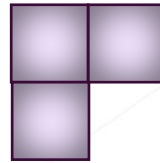
Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

Stride

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

Stride (2,2)



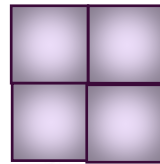
Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

Stride

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

Stride (2,2)



Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

Padding

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

Output Size

$$= \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1$$

```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Padding

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

Output Size

$$= \frac{\text{Input Size} + \text{Padding} - \text{Kernel Size}}{\text{Stride}} + 1$$

‘valid’
‘same’

```
tf.keras.layers.ZeroPadding2D(padding=((1, 0), (0, 0)))
```



```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Filter

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

0	0	1
1	1	0
1	0	3

1	0	1
0	1	0
1	0	1

5	0	3
0	2	0
-1	3	1



```
tf.keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```

Filter

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

0	0	1
1	1	0
1	0	3

1	0	1
0	1	0
1	0	1

5	0	3
0	2	0
-1	3	1

--	--

```
tf.keras.layers.ZeroPadding2D(padding=(1, 1))
```

```
tf.keras.layers.Conv2D(filters = 2, kernel_size = (3, 3), padding='valid', strides=(2, 2))
```

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

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

$x[:, :, 1]$

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

$x[:, :, 2]$

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

Filter W0 (3x3x3)

$w0[:, :, 0]$

-1	0	1
0	0	1
1	-1	1

$w0[:, :, 1]$

-1	0	1
1	-1	1
0	1	0

$w0[:, :, 2]$

-1	1	1
1	1	0
0	-1	0

Bias $b0$ (1x1x1)

$b0[:, :, 0]$

1

Filter W1 (3x3x3)

$w1[:, :, 0]$

0	1	-1
0	-1	0
0	-1	1

$w1[:, :, 1]$

-1	0	0
1	-1	0
1	-1	0

$w1[:, :, 2]$

-1	1	-1
0	-1	-1
1	0	0

Bias $b1$ (1x1x1)

$b1[:, :, 0]$

0

Output Volume (3x3x2)

$o[:, :, 0]$

2	3	3
3	7	3
8	10	-3

$o[:, :, 1]$

-8	-8	-3
-3	1	0
-3	-8	-5

toggle movement

Animation Source:
[Stanford CS231n](#)



256 x 256 x 3

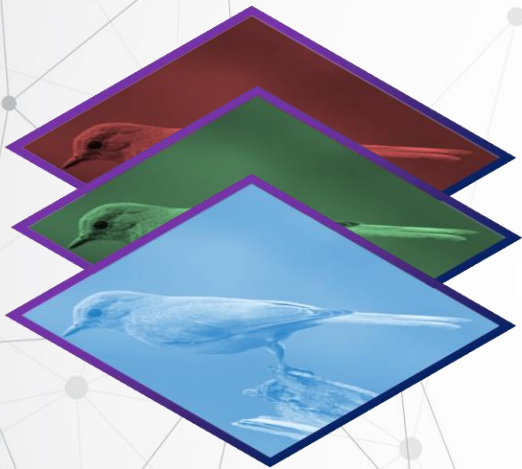
Conv2D

Conv2D

Flatten

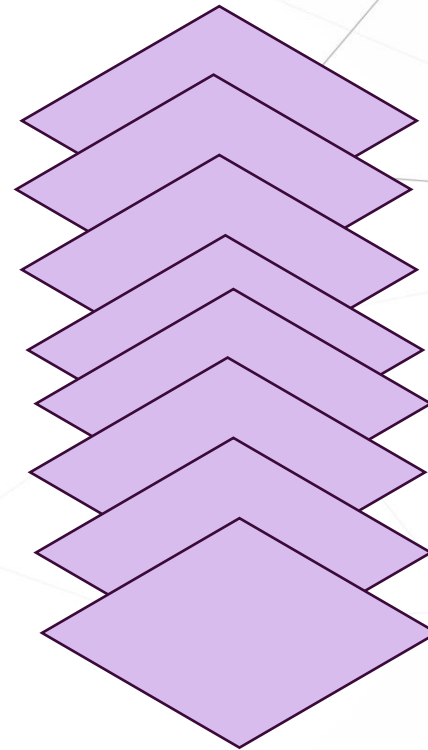
DNN


```
tf.keras.layers.Conv2D(filters = 8, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```



256 x 256 x 3

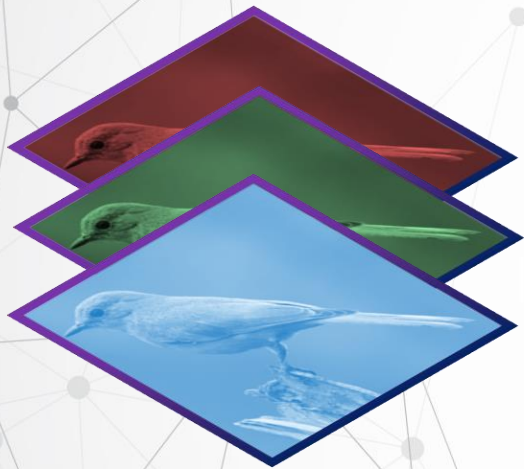
Conv2D



254 x 254 x 8

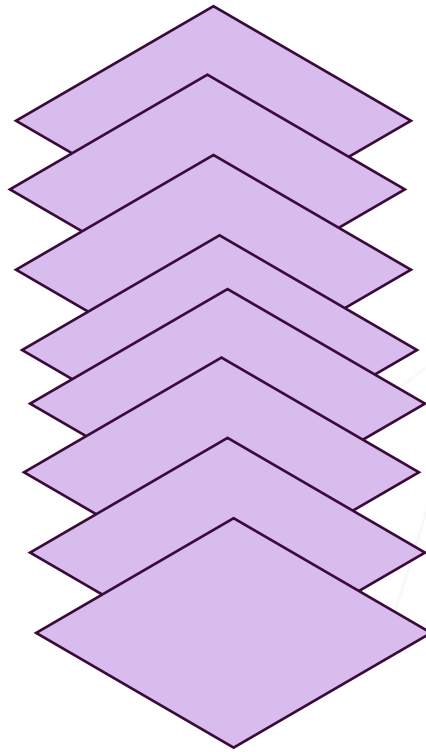
Conv2D

```
tf.keras.layers.Conv2D(filters = 4, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```



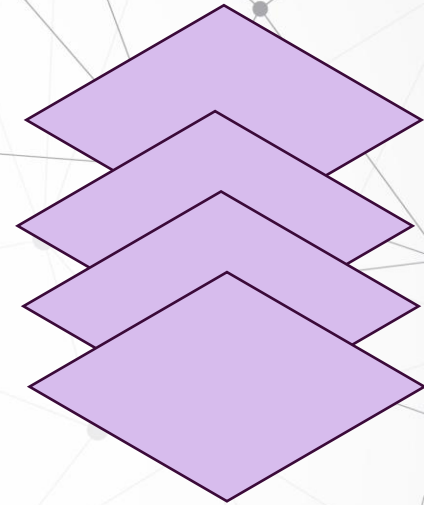
256 x 256 x 3

Conv2D



254 x 254 x 8

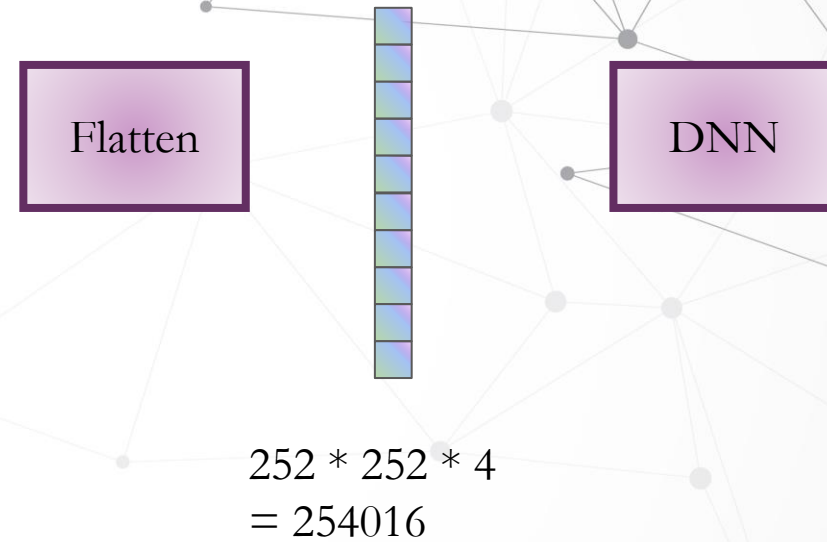
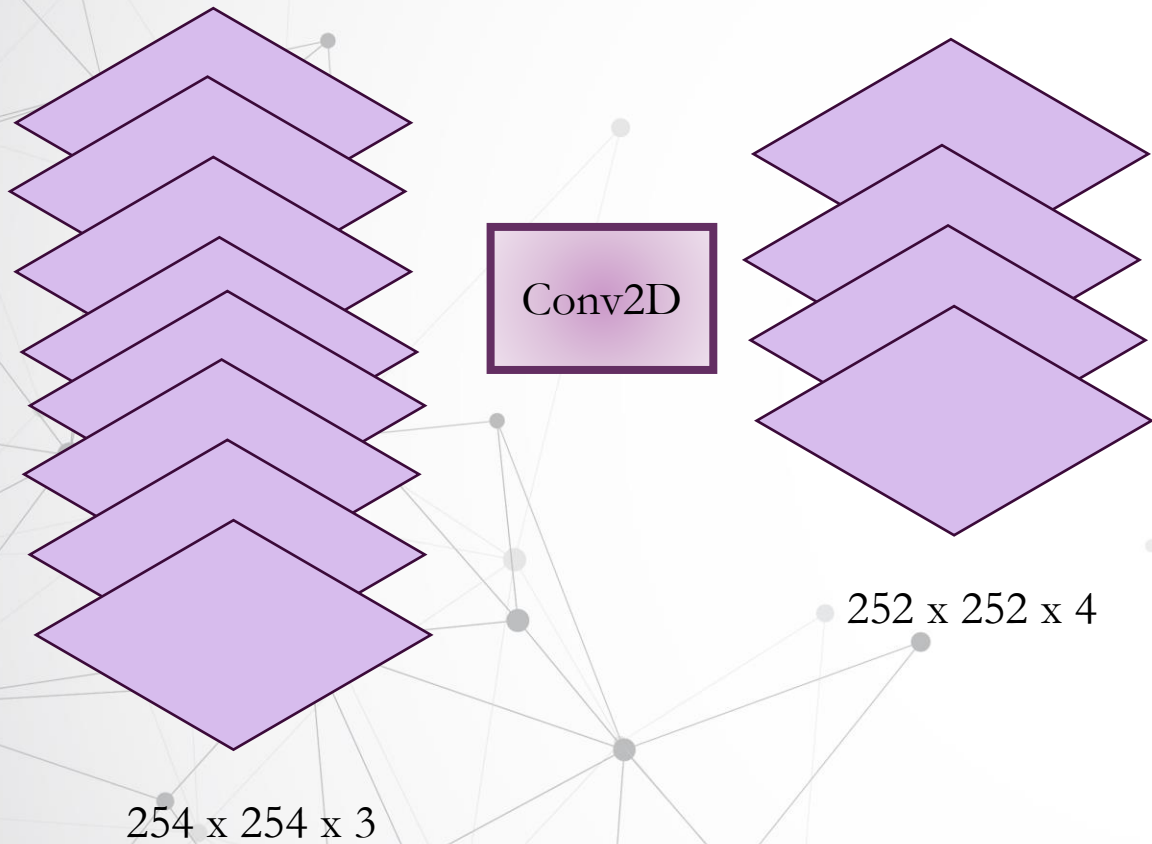
Conv2D



252 x 252 x 4

Flatten

```
tf.keras.layers.Conv2D(filters = 4, kernel_size = (3, 3), padding='valid', strides=(1, 1))
```



Pooling

```
tf.keras.layers.MaxPooling2D(pool_size = (2, 2))
```

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

6

Pooling

```
tf.keras.layers.MaxPooling2D(pool_size = (2, 2))
```

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

6	8
---	---

Pooling

```
tf.keras.layers.MaxPooling2D(pool_size = (2, 2))
```

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

6	8
3	

Pooling

```
tf.keras.layers.MaxPooling2D(pool_size = (2, 2))
```

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

6	8
3	4

Pooling

```
tf.keras.layers.MaxPooling2D(pool_size = (2, 2))
```



512,512,3



256,256,3



128,128,3



64,64,3



32,32,3

Putting it all together

```
tf.keras.layers.Input(shape=(32, 32, 3)),
tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='valid'),
tf.keras.layers.MaxPooling2D((2, 2)),
tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='valid'),
tf.keras.layers.MaxPooling2D((2, 2)),
tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='valid'),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(10, activation='softmax'),
```

Conv2D

MaxPooling2D

Conv2D

MaxPooling2D

Conv2D

Flatten

Dense

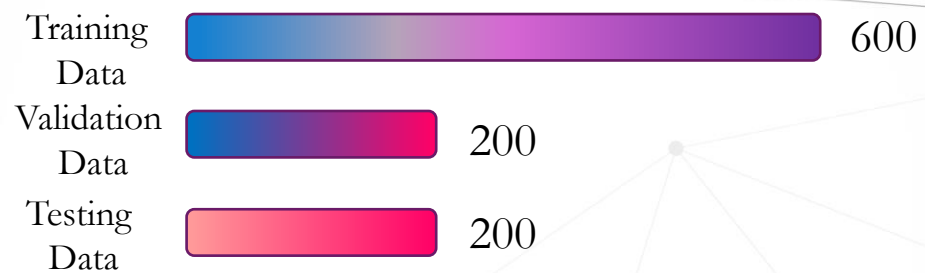
Dense

DNN vs CNN

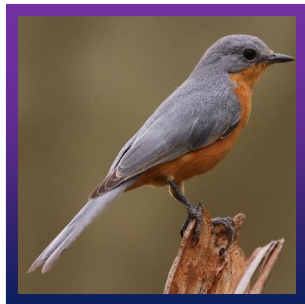
- Less Parameters
- CNNs are more robust to changes in Image



Data Augmentations



Data Augmentations

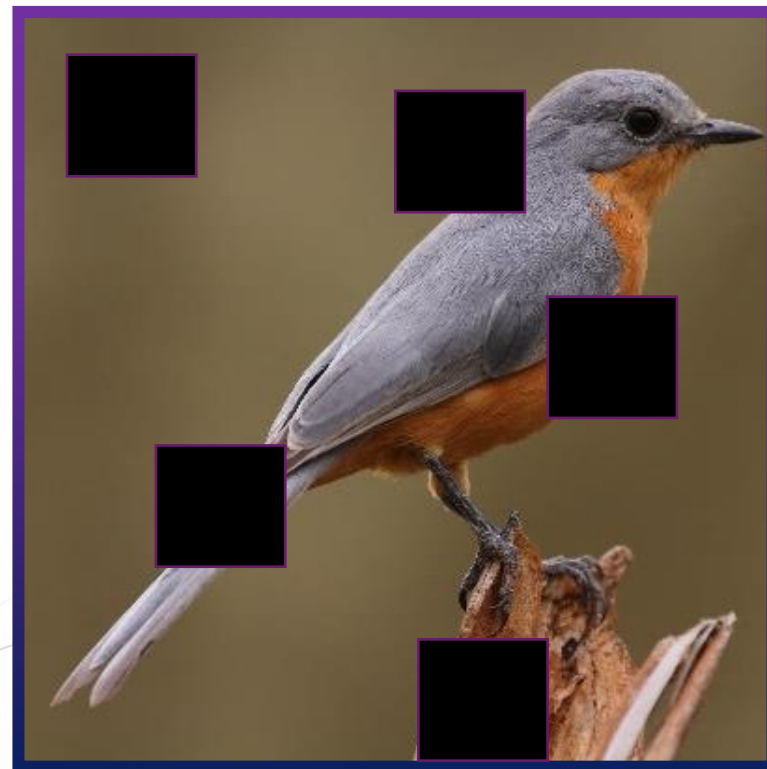


Why Augmentation?

- Increase Data
- More Generalizability
- Challenging cases
- Stopping over-reliance on features



Cutout



Mixup



Mixup

- 1 Car
- 0 Bus
- 0 CNG

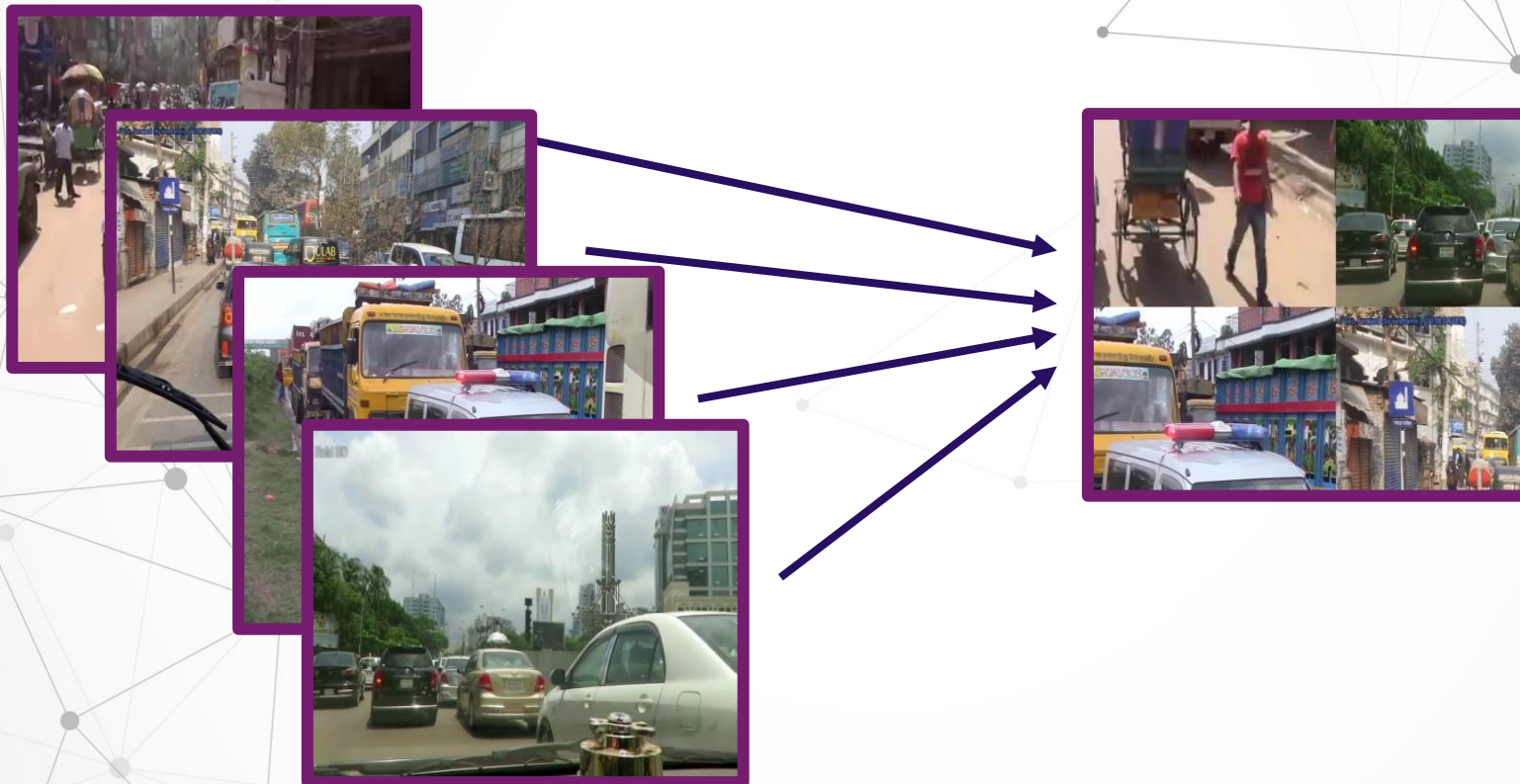


- 0
- 1
- 0

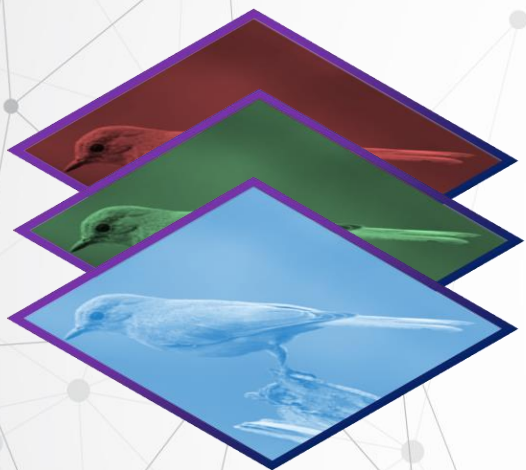


- 0.5
- 0.5
- 0

Mosaic

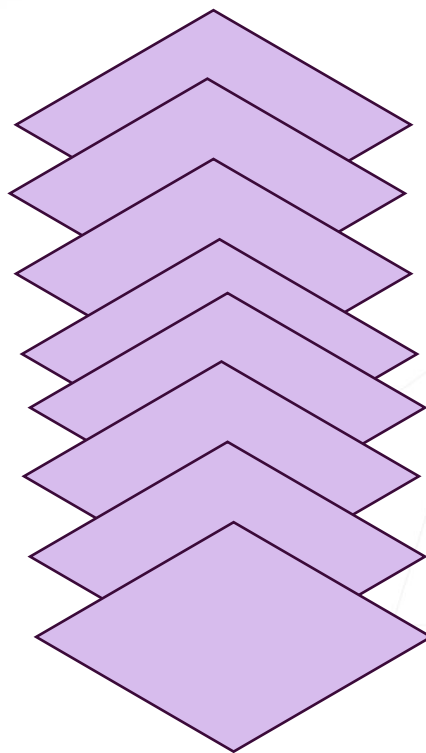


Dropout



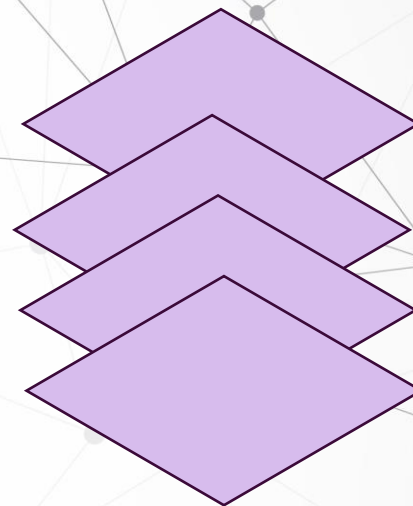
$256 \times 256 \times 3$

Conv2D



$254 \times 254 \times 8$

Conv2D



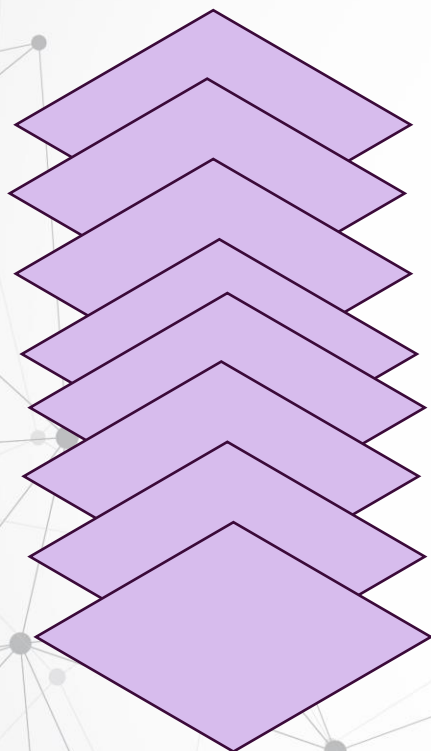
$252 \times 252 \times 4$

Flatten

Dropout

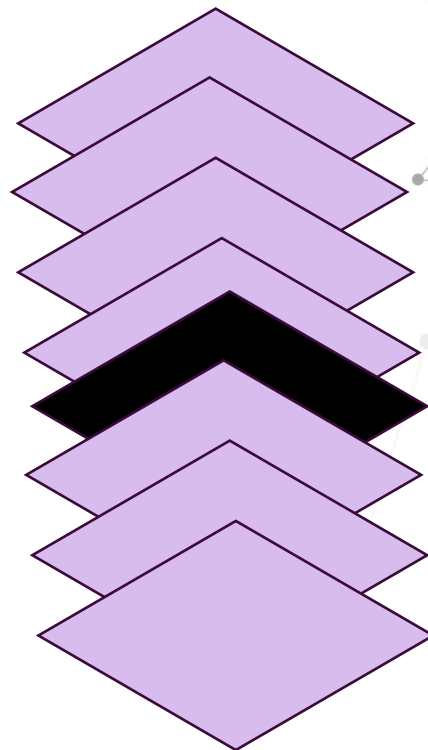
```
tf.keras.layers.Dropout(rate=.2)
```

Conv2D



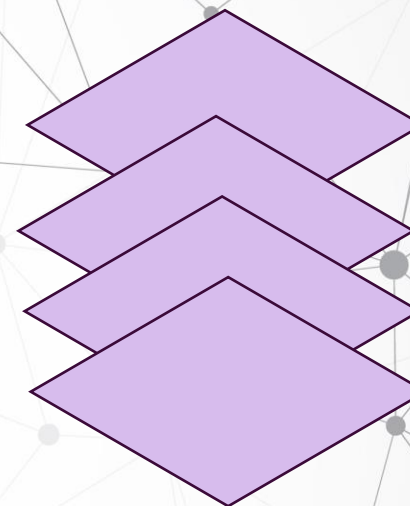
254 x 254 x 8

Dropout



254 x 254 x 8

Conv2D



252 x 252 x 4

Flatten



Any Question?