Interpreting a trained CNN

-- What did my convolutional network learn?

Columbus Machine Learners Meetup

Waylon Chen

June 5, 2019

Outline

Interpreting a trained CNN

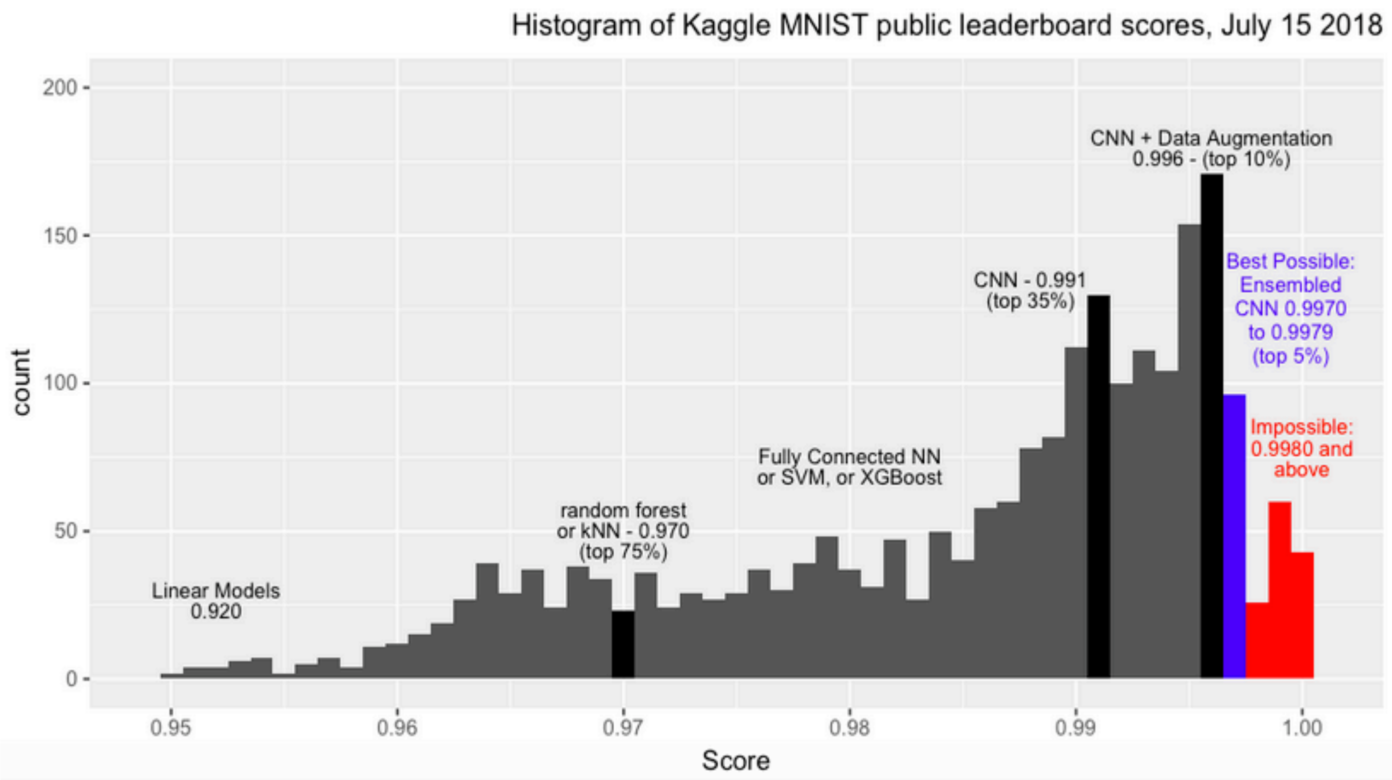
I. Motivation and introduction to CNN
   1. Neural network
   2. Convolution
   3. Convolutional network
II. The approaches
   1. Visualize intermediate activations
   2. My CNN encoder and interpreters – some hybrid models
   3. Did my CNN learn translational invariance?
III. Conclusion

I. Motivation and Introduction to CNN

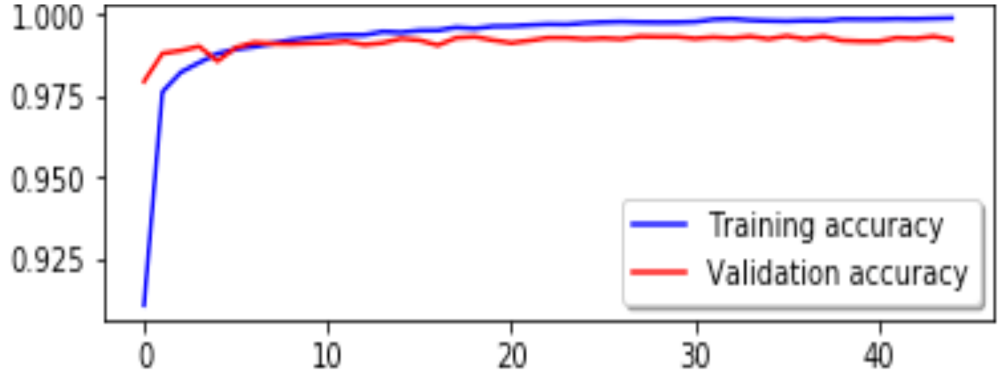
1. Motivation
2. Neural network
3. Convolution
4. Convolutional network

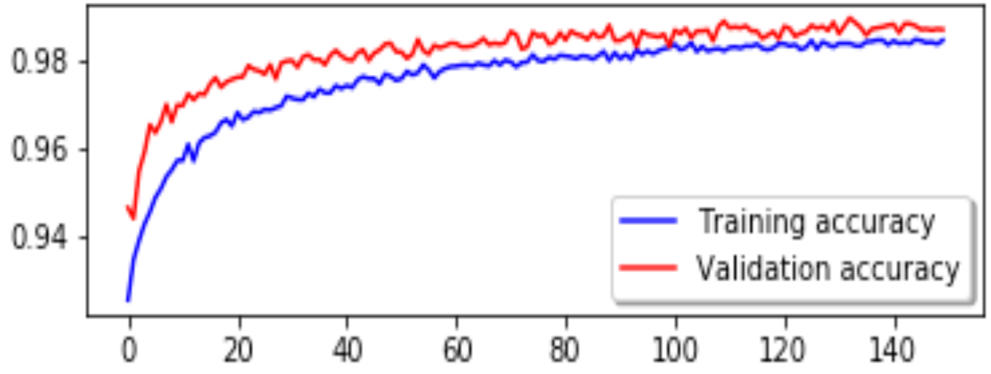
Motivation

1. My CNN classifier achieved 99.3% accuracy on MNIST hand-written digit dataset, but I cannot interpret it!
2. Knowing the information it did/did not learn can be the key to improvements.
3. *Validation accuracy exceeds training accuracy after data augmentation!*



https://www.kaggle.com/cdeotte/25-million-images-0-99757-mnist

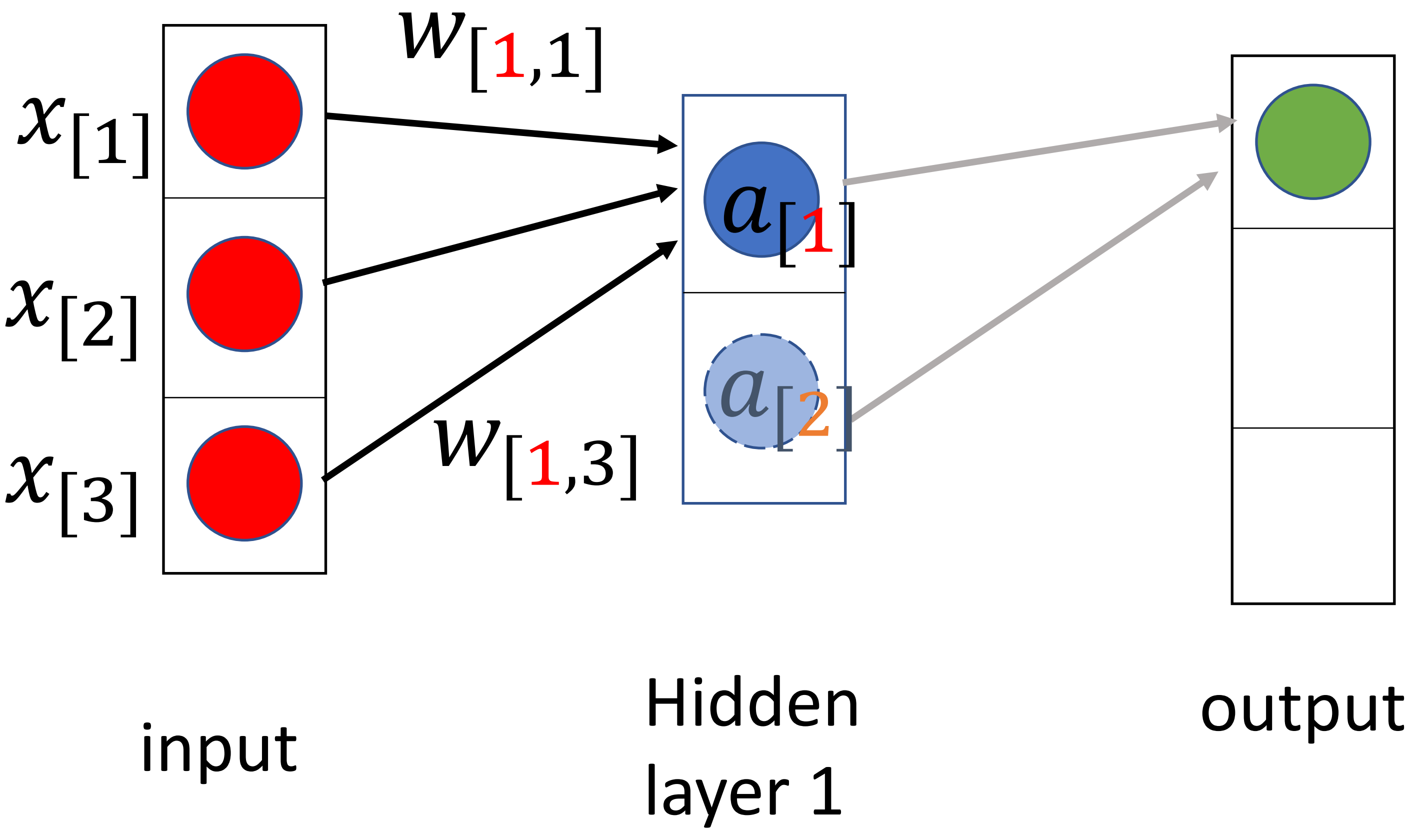




Neural network

$$z_{[1]} = w_{[1,1]}x_{[1]} + w_{[1,2]}x_{[2]} + w_{[1,3]}x_{[3]} + b_{[1]} \cdot 1$$

$$a_{[1]} = \sigma\left(z_{[1]}\right) \qquad \sigma: nonlinear\ activation\ function$$



TL;DR

1. Weigh
2. Bias
3. Activate

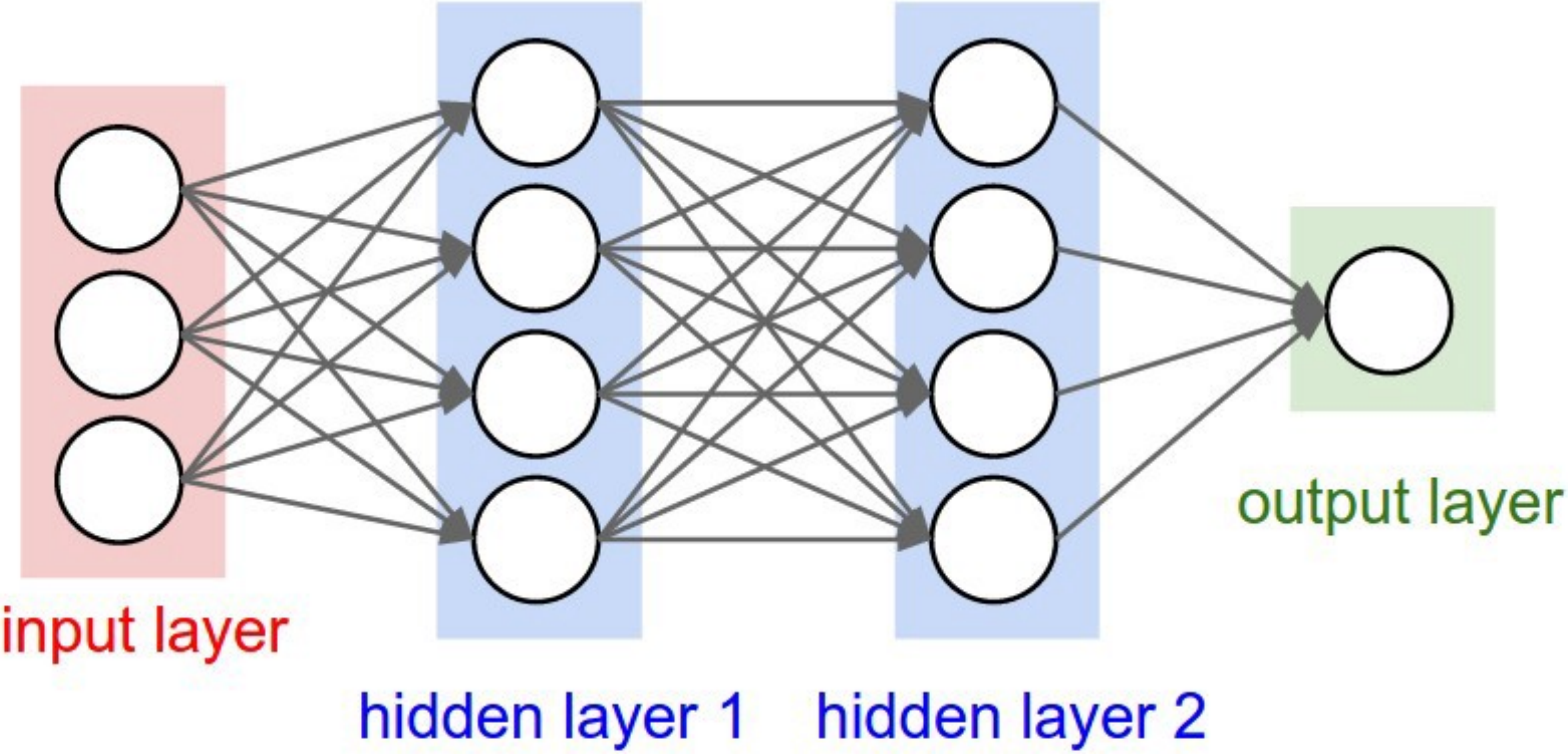
$$\vec{a} = \sigma\left(\overleftrightarrow{w} \cdot \vec{x} + \vec{b}\right)$$

$$\vec{a}^{(1)} = \sigma\left(\overleftrightarrow{w}^{(1)} \cdot \vec{x} + \vec{b}^{(1)}\right)$$

$$\vec{a}^{(2)} = \sigma\left(\overleftrightarrow{w}^{(2)} \cdot \vec{a}^{(1)} + \vec{b}^{(2)}\right)$$

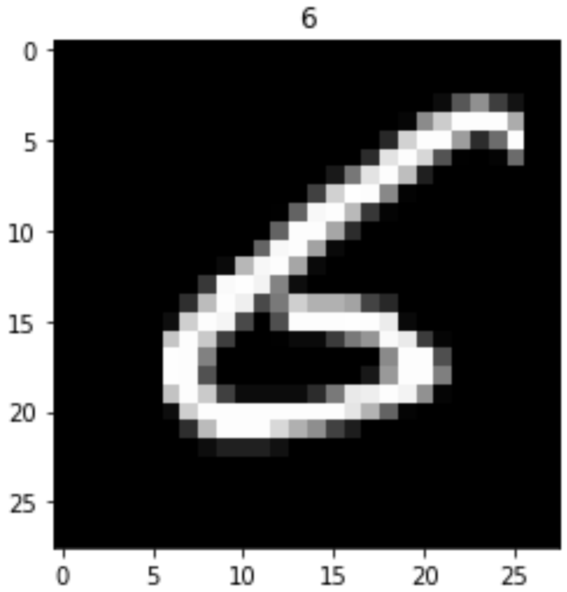
$$\vec{a}^{(h)} = \sigma\left(\overleftrightarrow{w}^{(h)} \cdot \vec{a}^{(h-1)} + \vec{b}^{(h)}\right)$$

Neural Network – fully connected



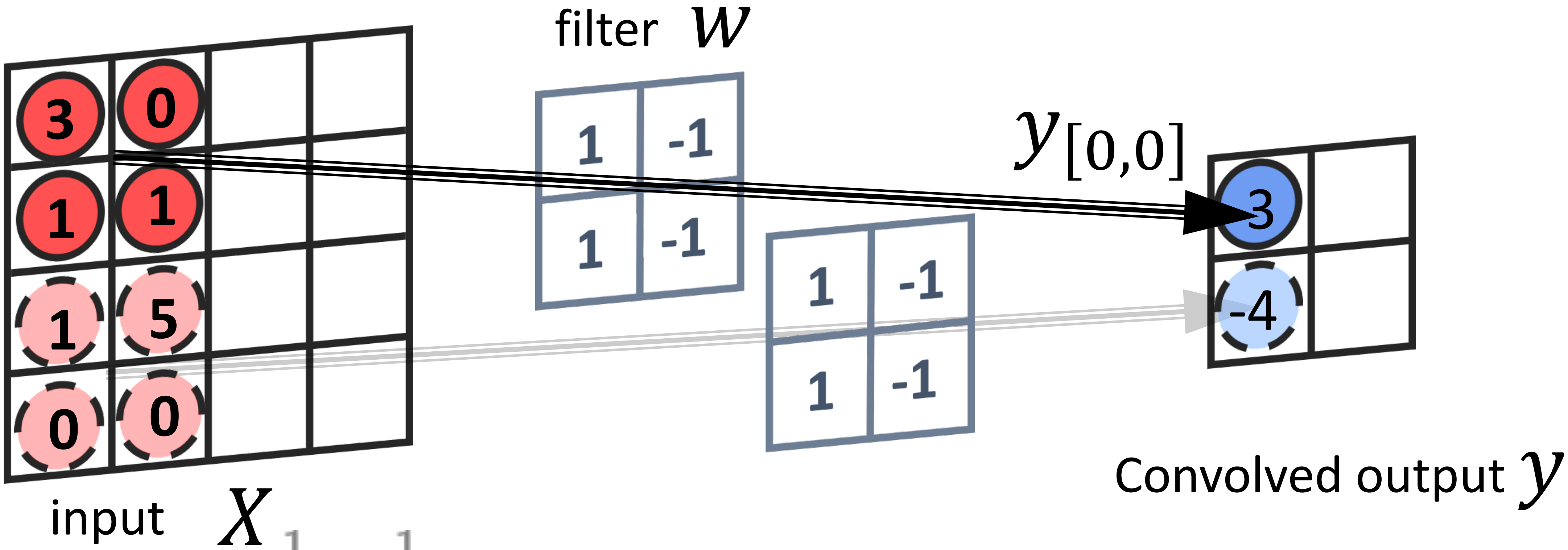
Pixels

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	12	87	144	62	19	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	139	204	253	253	253	172	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	210	253	253	157	44	108	253	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	228	253	215	83	1	0	4	107	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	119	227	253	190	31	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	66	209	255	253	136	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	99	248	253	185	55	4	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	99	253	253	166	44	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	100	248	253	162	9	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	10	183	247	253	166	9	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	57	250	254	243	131	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	47	188	253	239	74	122	207	177	177	163	67	41	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	14	207	253	245	73	0	79	253	253	253	254	253	236	9	0	0	0	0	0	0	0	0
0	0	0	0	0	0	197	253	201	62	0	0	1	10	10	59	122	149	250	234	61	5	0	0	0	0	0	0
0	0	0	0	0	0	254	253	129	0	0	0	0	0	0	0	0	0	138	253	253	79	0	0	0	0	0	0
0	0	0	0	0	0	254	253	82	0	0	0	0	0	0	0	0	24	153	253	253	127	0	0	0	0	0	0
0	0	0	0	0	0	196	253	202	64	12	12	12	12	45	122	233	238	253	243	145	5	0	0	0	0	0	0
0	0	0	0	0	0	13	207	253	253	253	253	253	253	253	253	229	180	99	4	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	46	187	253	253	253	216	176	142	66	29	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	13	33	33	33	17	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	0	0	0	0	0	0	0	0	0	0	0



Convolution

$$y_{[0,0]} = \sum_{t=0}^{1}\sum_{s=0}^{1} w_{[s,t]} X_{[0+s,0+t]}$$



$$y_{[i,j]} = \sum_{t=0}^{1}\sum_{s=0}^{1} w_{[s,t]} X_{[2i+s,2j+t]}$$

$$y = w * X$$

Convolution

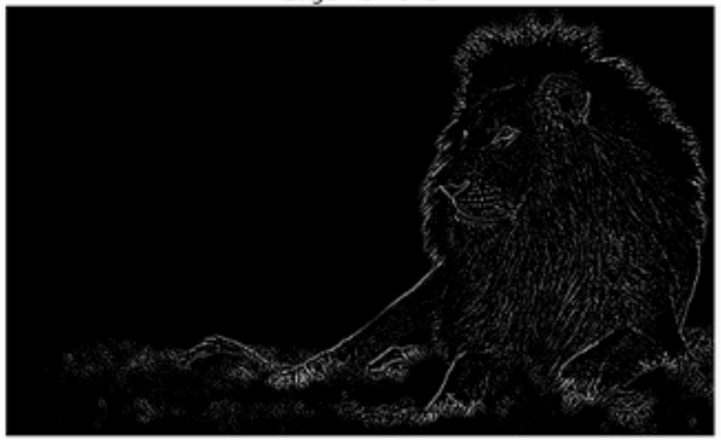
- Convolution can do edge detection, blurring, sharpening and etc.

```
edge_kernel_1 = np.array([[1, 0, -1],
                          [0, 0, 0],
                          [-1, 0, 1]])

edge_kernel_2 = np.array([[0, 1, 0],
                          [1, -4, 1],
                          [0, 1, 0]])

edge_kernel_3 = np.array([[-1, -1, -1],
                          [-1, 8, -1],
                          [-1, -1, -1]])
```

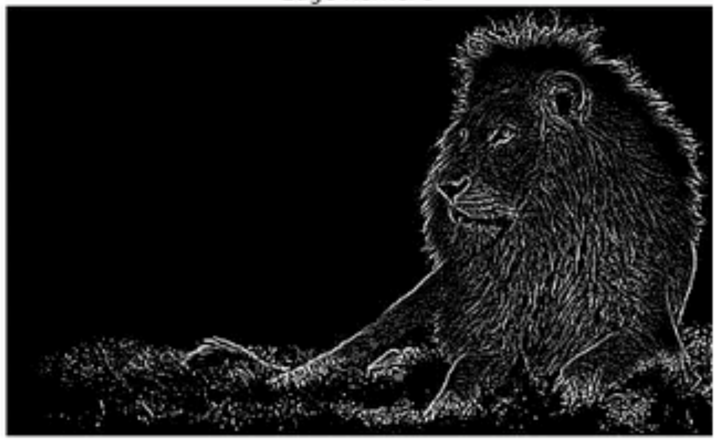
Original Image

Edge Kernel 1

Edge Kernel 2

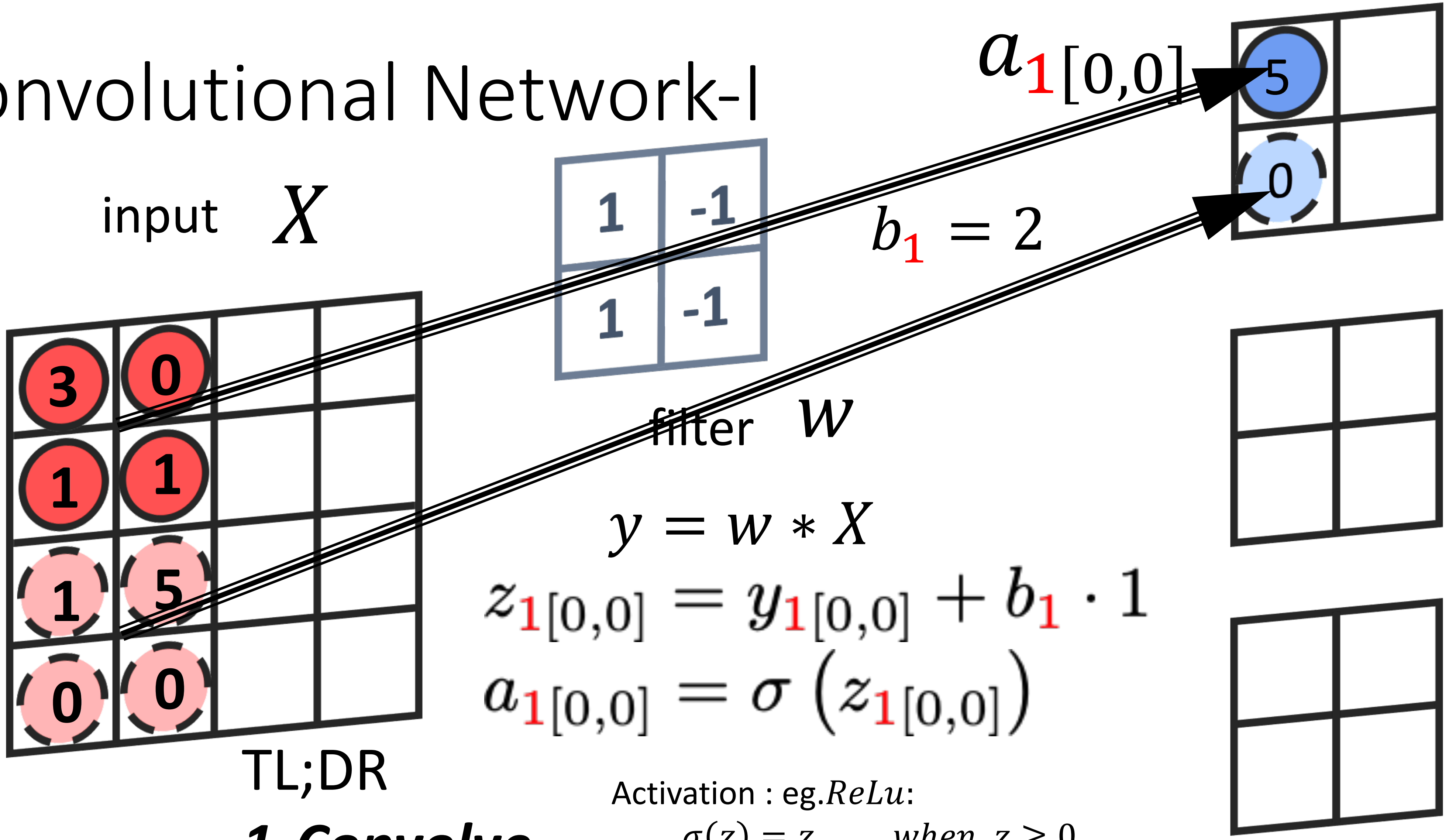


Edge Kernel 3



https://beckernick.github.io/convolutions/

Convolutional Network-I



$$y = w * X$$

$$z_{1[0,0]} = y_{1[0,0]} + b_1 \cdot 1$$

$$a_{1[0,0]} = \sigma\left(z_{1[0,0]}\right)$$

Activation : eg.$ReLu$:

$$\sigma(z) = z, \quad when\ \ z \geq 0$$
$$= 0, \quad when\ \ z < 0$$

TL;DR

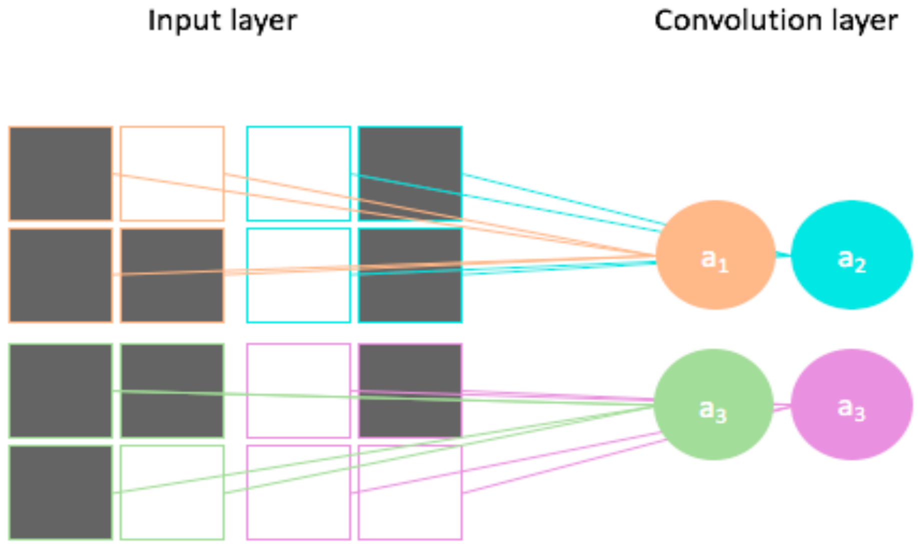
1. Convolve

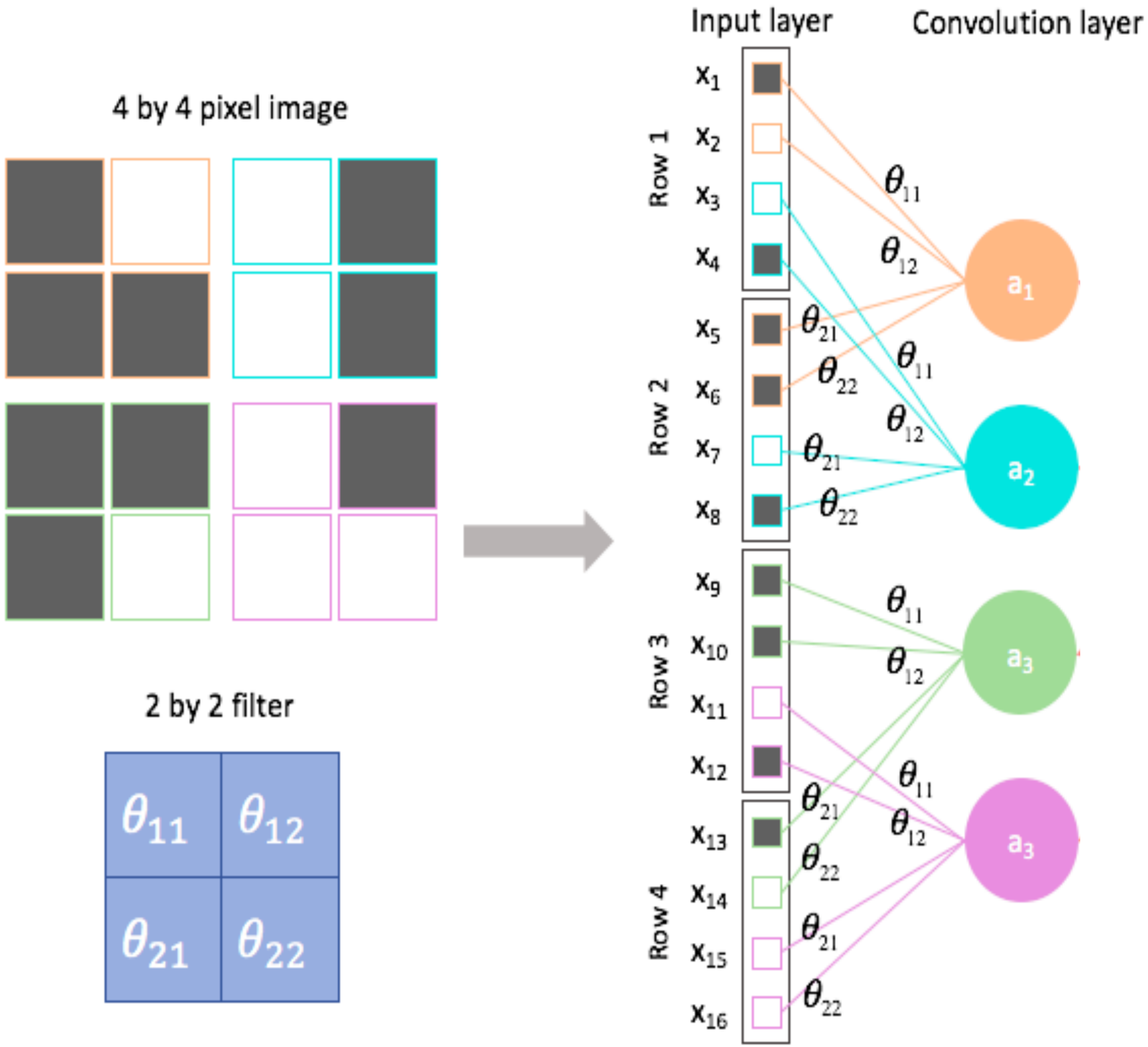
2. Bias

3. Activate

Conv Net-II

1. Sparsely connected network
2. Locality is preserved with convolution





https://www.jeremyjordan.me/convolutional-neural-networks/

The approaches

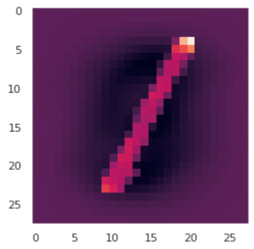
1. Visualize the intermediate activations
2. CNN encoder and interpreters
3. Did my CNN learn translational invariance?

Visualize the intermediate activations-I

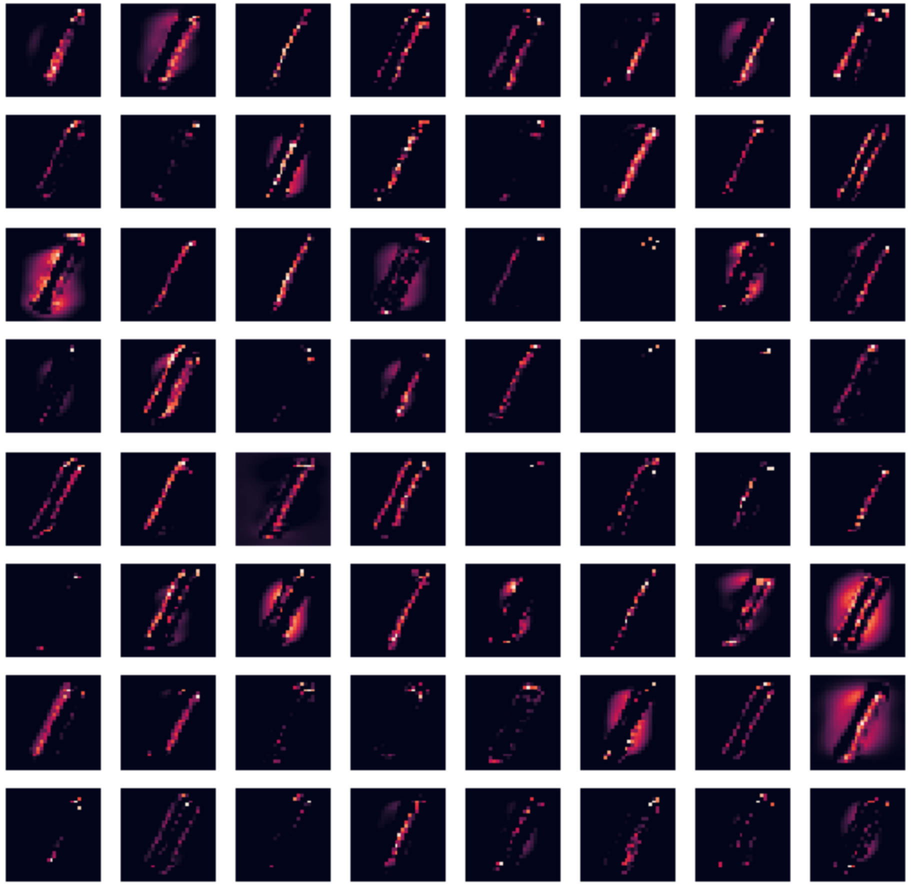
h:hiddne layer index

conv2d_1 $h = 1$

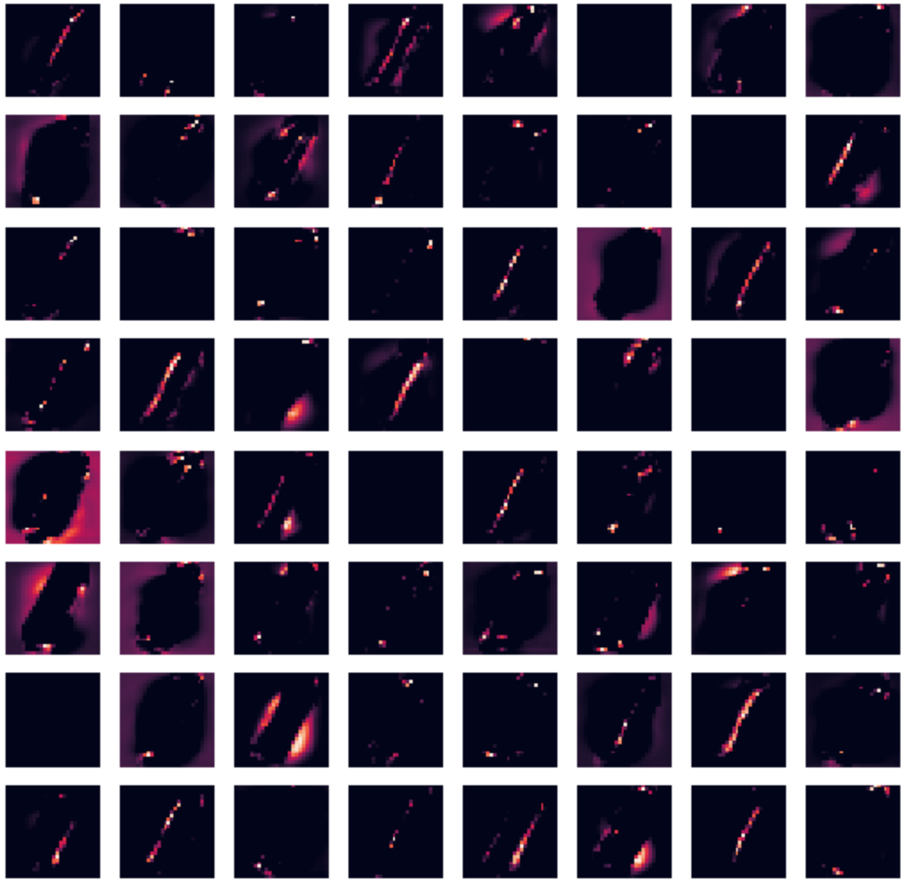
conv2d_2 $h = 2$



28*28=784



(28*28)*64=50,176



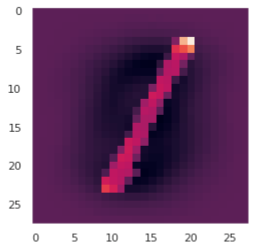
(28*28)*64=50,176

Visualize the intermediate activations-II

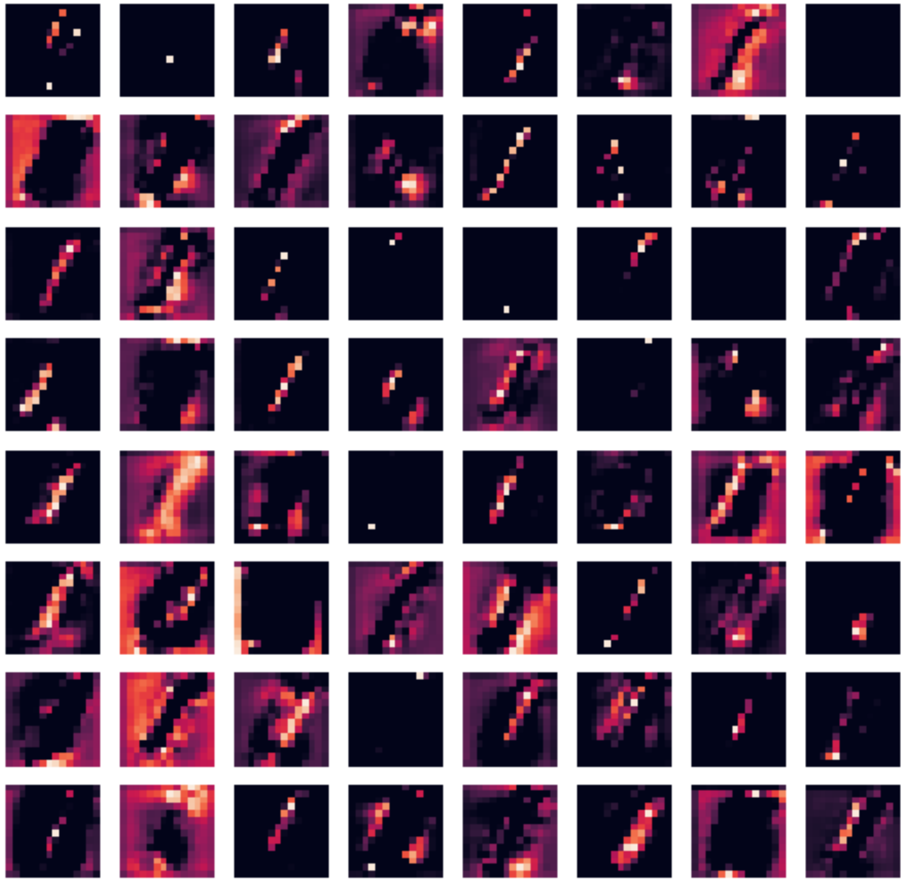
h:hiddne layer index

conv2d_3 $h = 5$

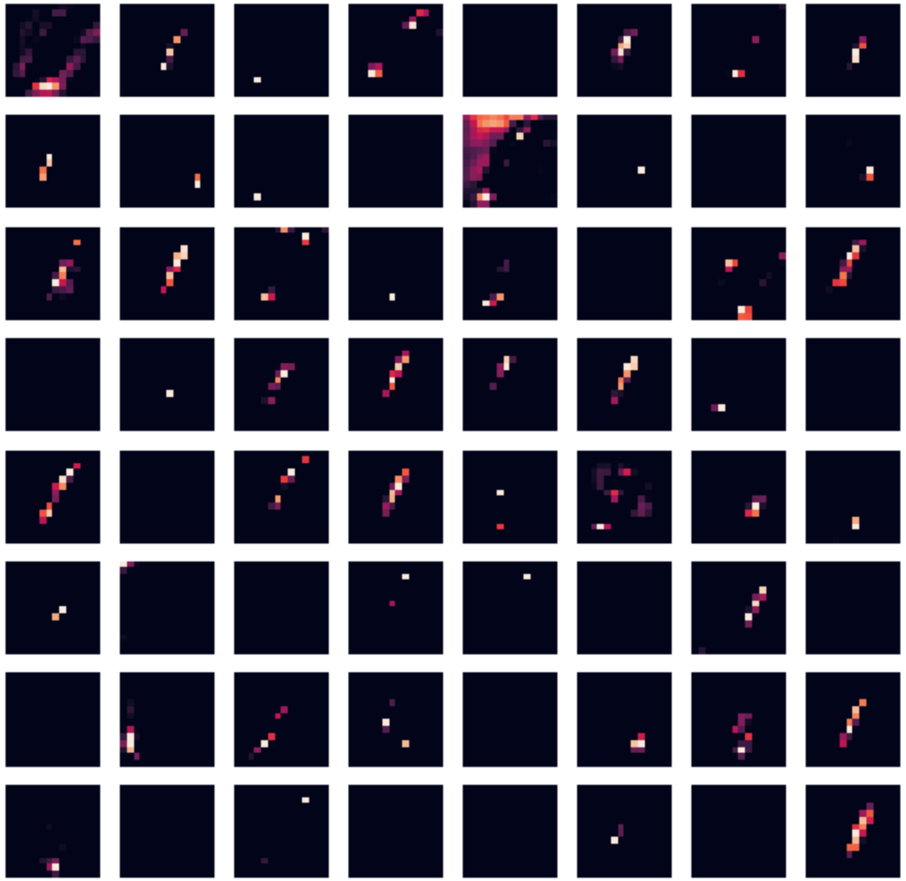
conv2d_4 $h = 6$



28*28=784



(14*14)*64=12,544

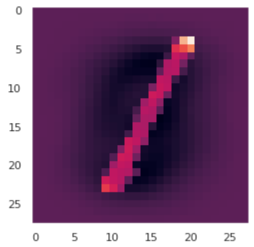


(14*14)*64=12,544

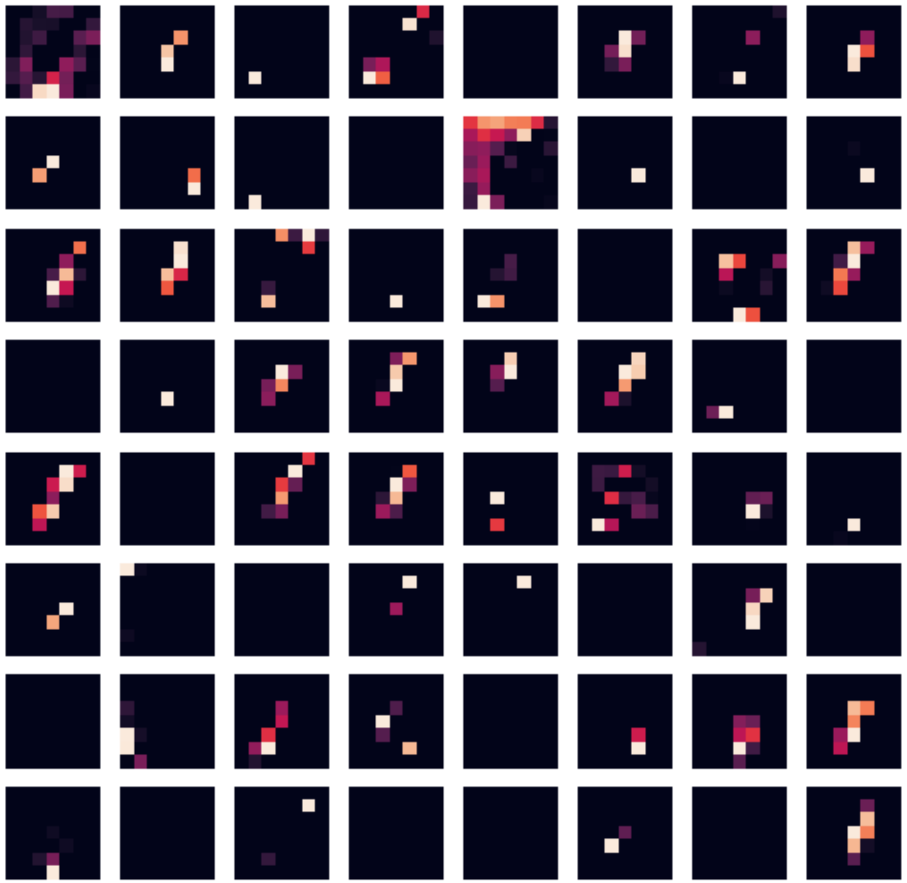
Visualize the intermediate activations-III

h:hiddne layer index

dropout_2 $h = 8$

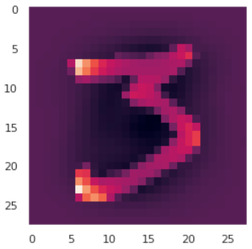


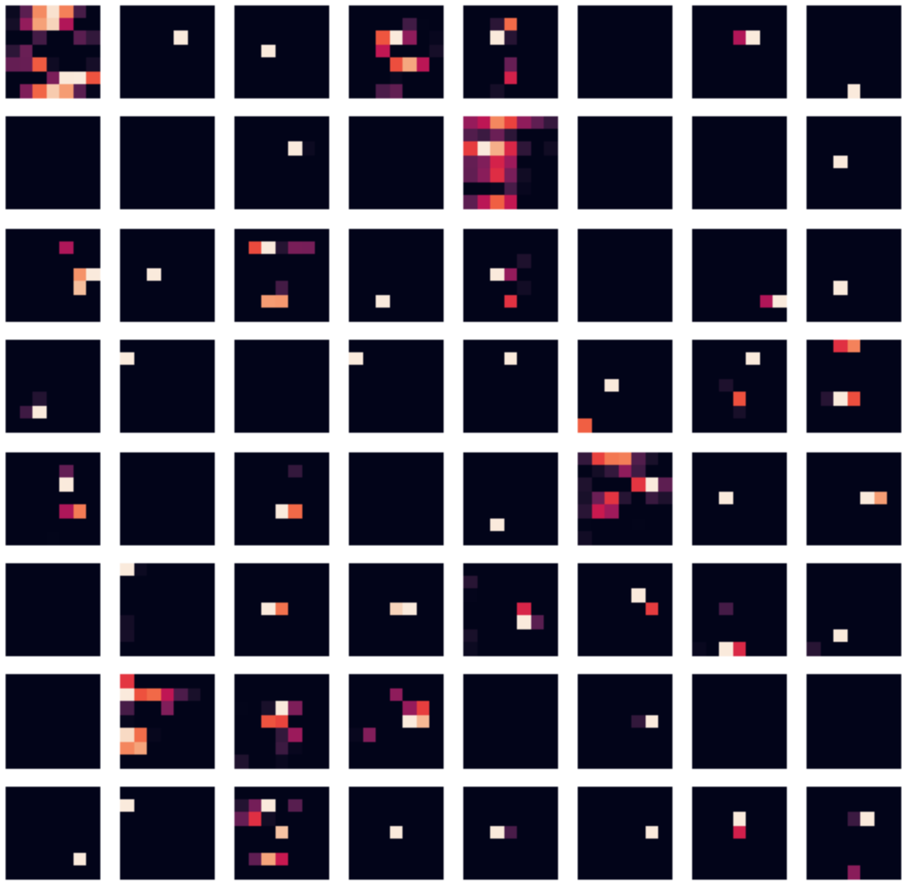
28*28=784



(7*7)*64=3,136

dropout_2 $h = 8$





(7*7)*64=3,136

CNN encoder and interpreters

the architecture

```
#The model structure
my_CNN_clf=Sequential([
    ############################## CNN-encoder
    Conv2D(filters = 64, activation = 'relu',kernel_size=(5,5),padding='Same',input_shape = (28,28,1)),
    Conv2D(filters = 64, activation = 'relu',kernel_size=(5,5),padding='Same'),
    MaxPool2D(pool_size=(2,2),strides=(2,2)),
    Dropout(0.25),
    Conv2D(filters = 64, activation = 'relu',kernel_size=(3,3),padding='Same'),
    Conv2D(filters = 64, activation = 'relu',kernel_size=(3,3),padding='Same'),
    MaxPool2D(pool_size=(2,2),strides=(2,2)),
    Dropout(0.25),
    Flatten(), # d=64*7*7=3136
    ############################## CNN-encoder

    ##################            compressor
    Dense(256, activation='relu'),
    Dropout(0.5), #d = 256
    ##################            compressor

    ############################## interpretor
    Dense(10, activation='softmax'), #a normalized exponential functions for probability
    ############################## interpretor
    ])
```

$h = 1$

$h = 2$

$h = 5$

$h = 6$

$h = 8$

Encoder: input→ 3,136-dimensional vector

$h = 10$

Compressor: 3,136 → 256 dimensional

<u>The code vector</u>

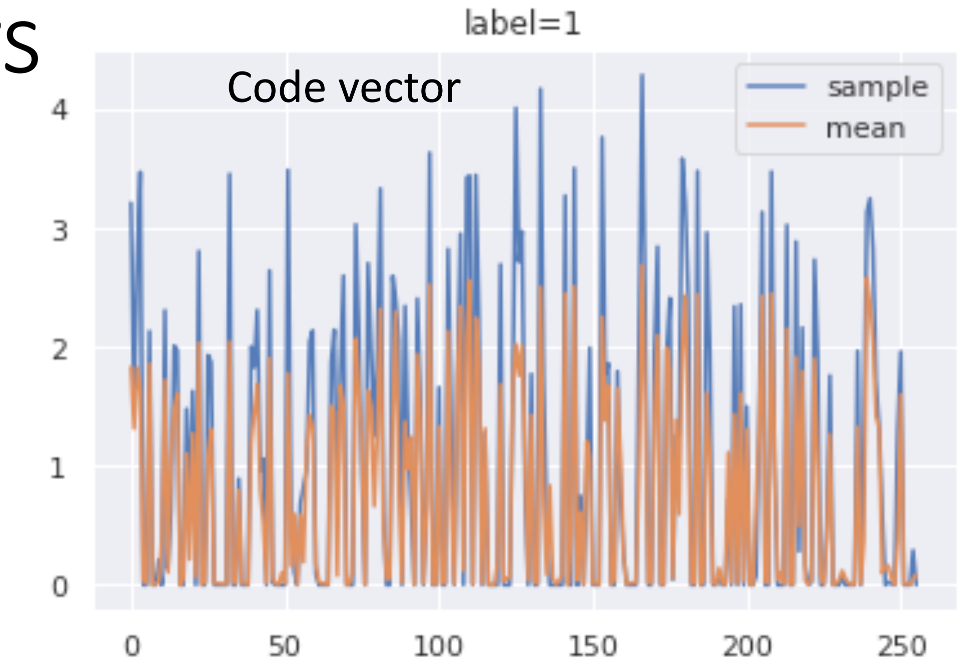
Interpreter: 256 → 10 dim. probability vector for predictions

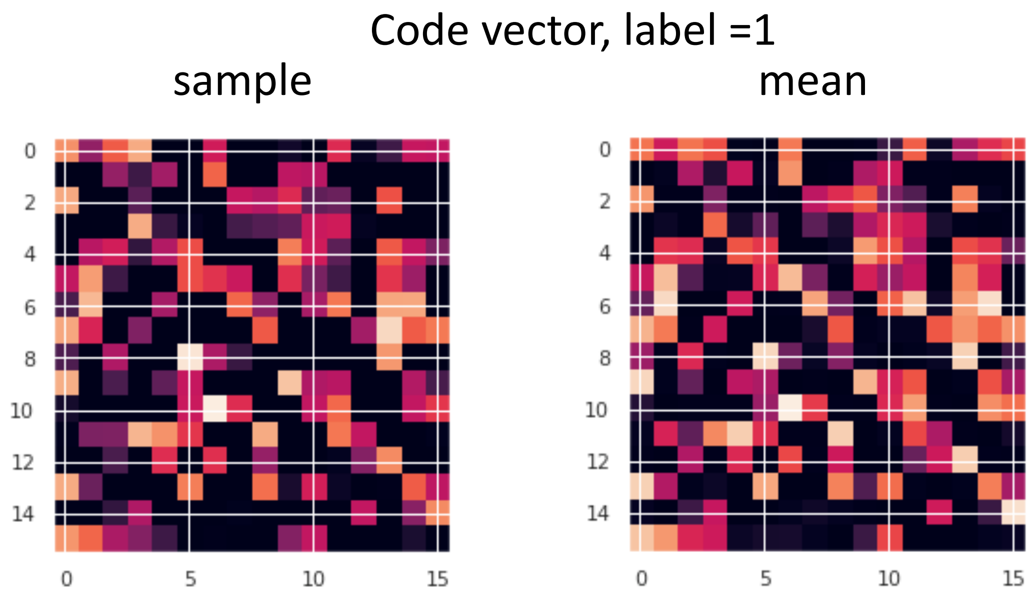
CNN encoder and interpreters

the code vectors

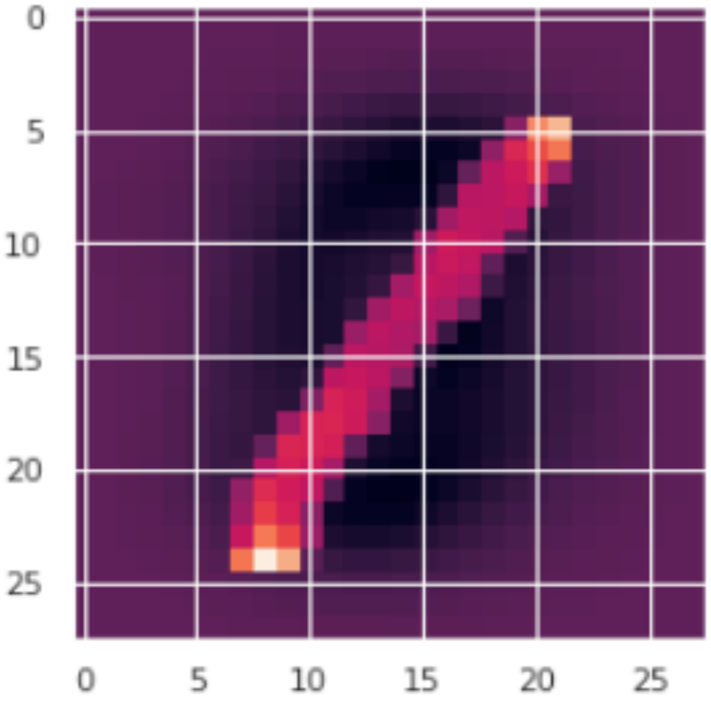
1. Consistent and self-similarity within a class
2. Different interpreters show similar accuracy
   => Conv. encoder is the key

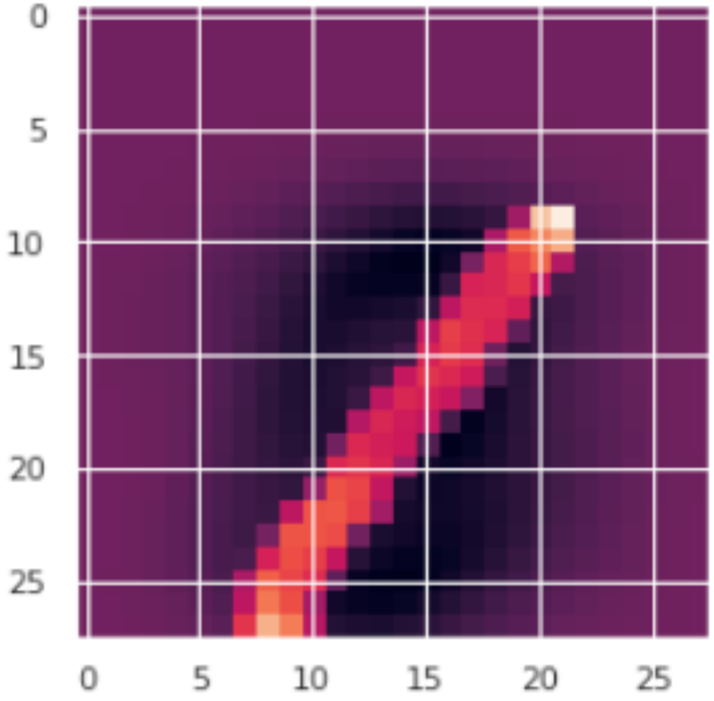
Interpreters	accuracy
1 Dense layer NN	99.44%
SVM	99.35%
Random Forests (100 trees)	99.32%
Gradient Boost (100 trees)	99.24%
Decision Tree	98.46%





Translational Invariance

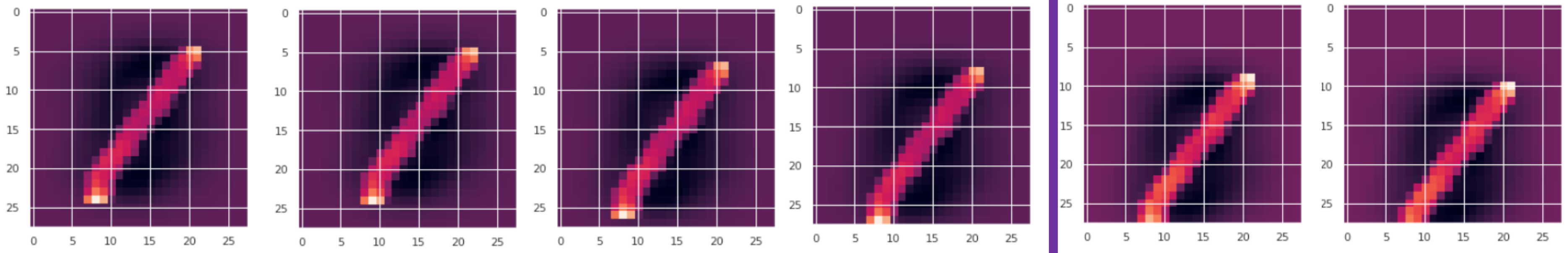




Did my CNN learn translational invariance?

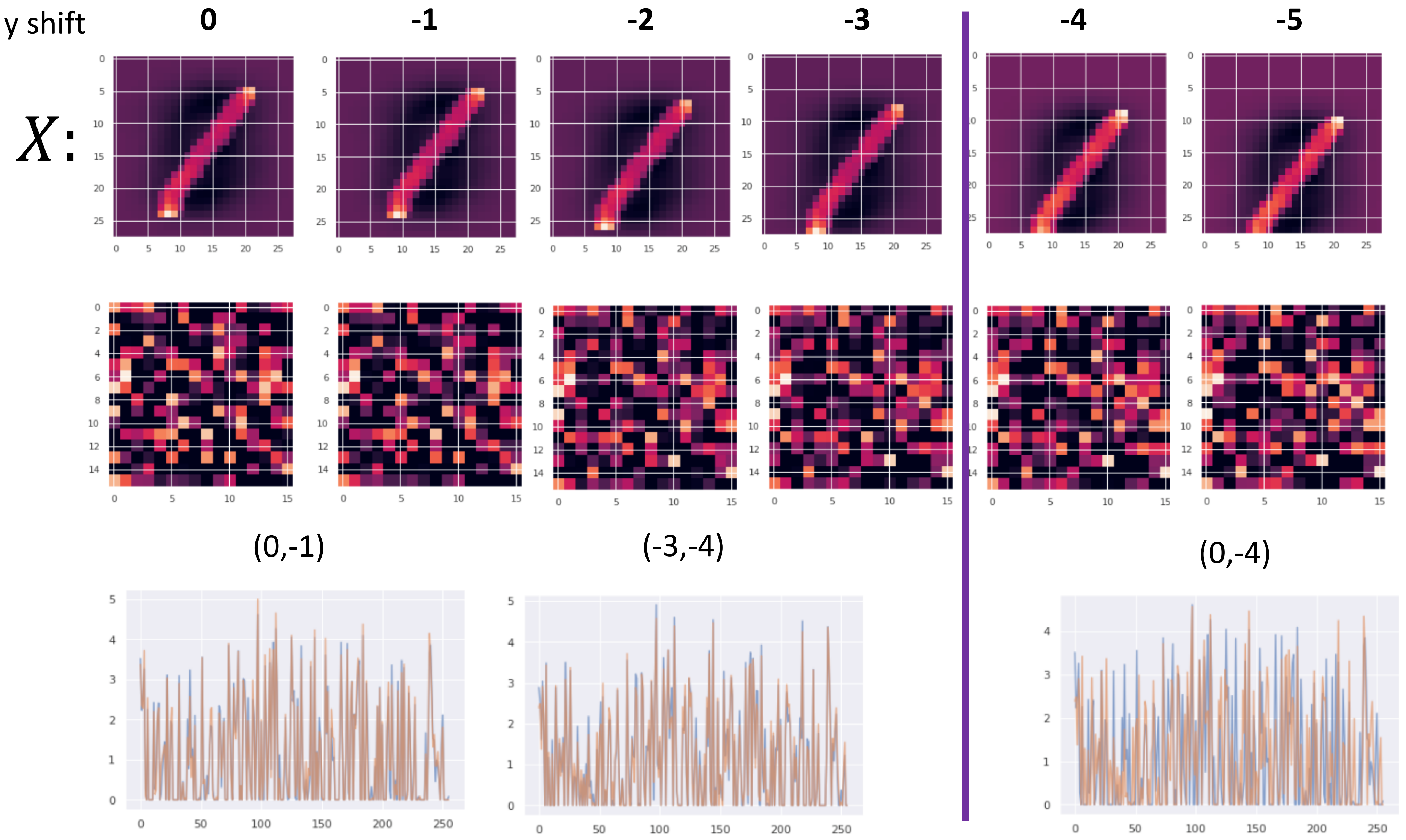
$Prob(y = 1|X)$

X:



y shift (pixels)	0	-1	-2	-3	-4	-5
CNN (CNN+NN) (acc = 99.3%)	100%	100%	99.99%	99.4%	0.71% (7: 99.29%)	1.4E-4% (7: 99.99%)
CNN+RF, T=1000 (acc = 99.3%)	100%	96.6%	67.7% (7: 18.4)	52.8% (7: 28.3%)	44.4% (7: 35.5%)	34.8% (7: 47.6 %)
CNN+SVM (acc = 99.3%)	99.9%	99.8%	97.8%	79.9% (9: 18.8%)	21% (9: 77.76%)	1% (9: 98.9%)
Random Forest (RF) (acc = 96.6%)	99.5%	84.9% (7: 9.8%)	34.1% (7: 39.6%)	9.1% (7: 56.3%)	5.4% (7: 70.1%)	4.8% (7: 70.2 %)
SVM (acc = 96.1%)	99.6%	42.8% (7: 50.8%)	0.85% (7: 94.5%)	0.42% (7: 81.3 %)	0.43% (7: 79.5%)	0.42% (80.8%)

Did my CNN learn translational symmetry?

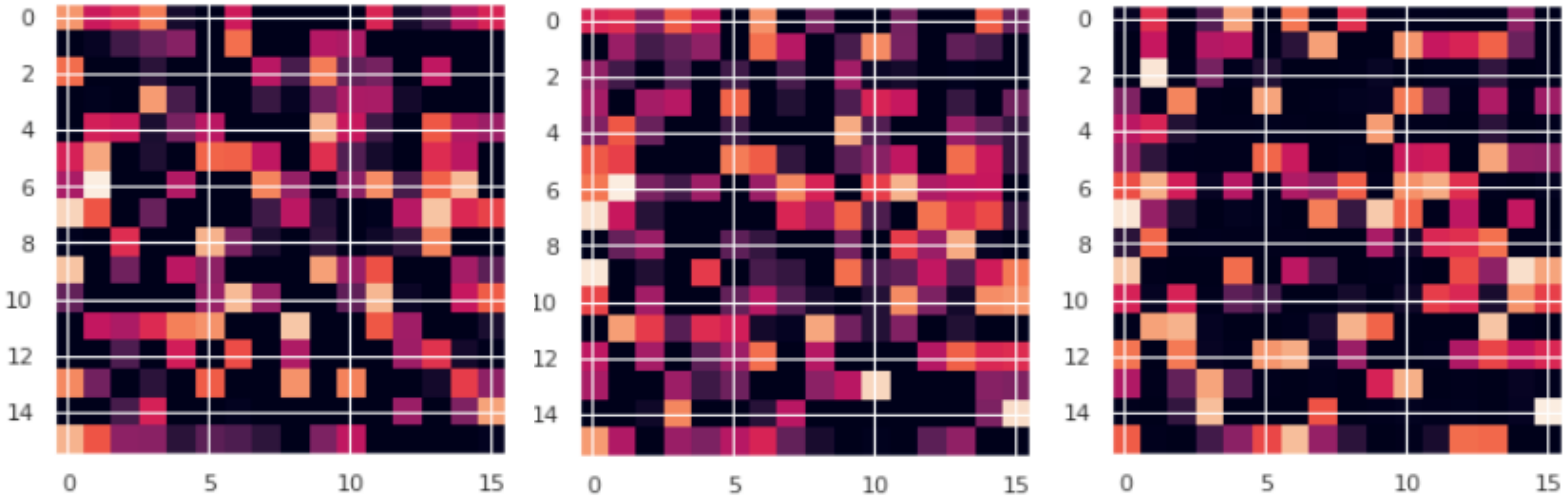


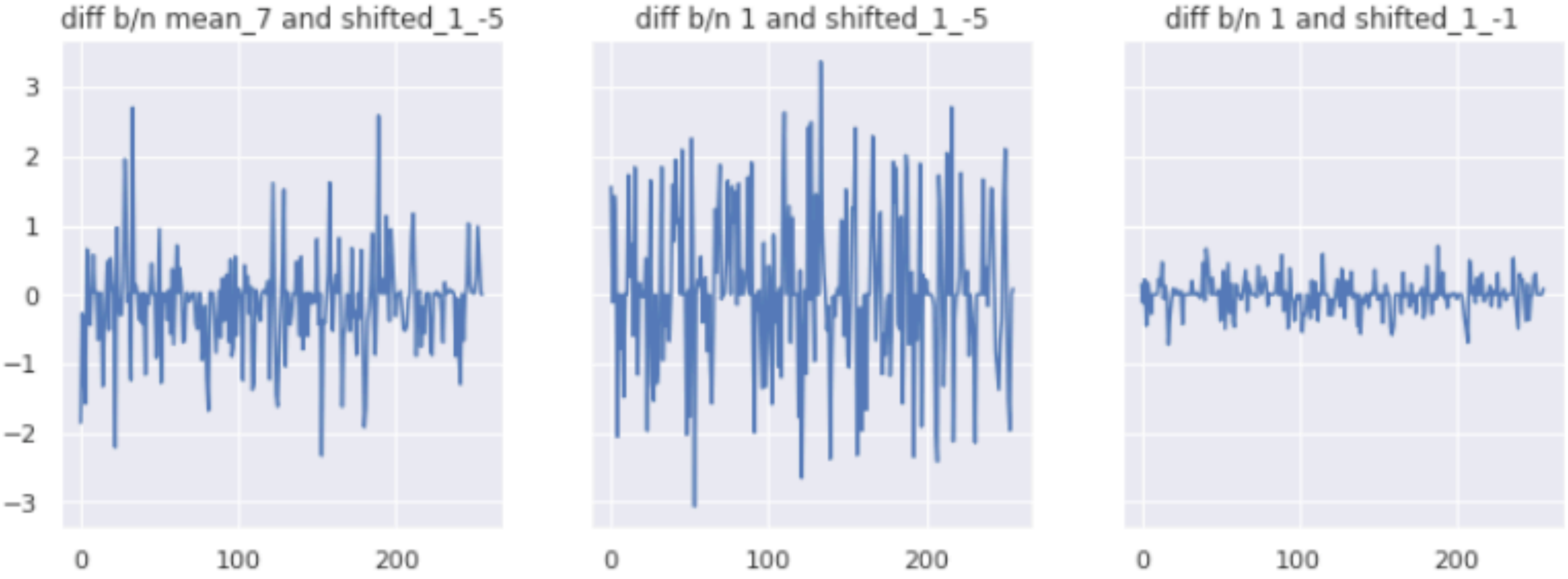
Conclusion

- Convolution reduces the neural network complexity and comprehend locality (relative positions) of the image (image pixels).
- The deep convolutional encoder is the key for the performance
- CNN encoding models learn translational invariance better than others, though not perfect.

Conclusion

- Code vectors in hidden layers of CNN reduce the dimensionality of images.
- Information learned in CNN is **position sensitive**. And convolution is the key for high accuracy performance.
- The predicted probabilities of CNN are not continuous with respect to translations.
- Translations deform CNN code vectors more softly.


0
2
4
6
8
10
12
14
0
5
10
15
0
2
4
6
8
10
12
14
0
5
10
15
0
2
4
6
8
10
12
14
0
5
10
15


diff b/n mean_7 and shifted_1_-5
diff b/n 1 and shifted_1_-5
diff b/n 1 and shifted_1_-1
3
2
1
0
−1
−2
−3
0
100
200