La Serena School for Data Science

Manifold learning and dimensionality reduction

2024

Federica Bianco

University of Delaware

Rubin Observatory

Special thanks to A. Bayo

access this presentation at

https://slides.com/federicabianco/lsdss24_AEC

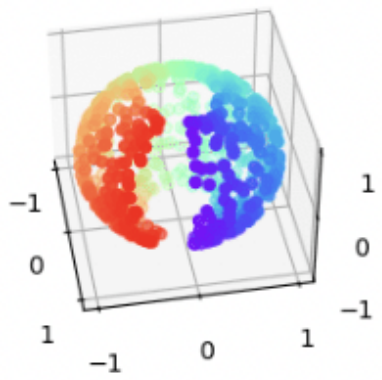
1/6

dimensionality reduction - why

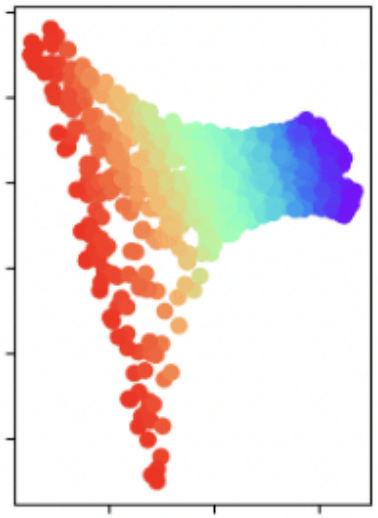
- Visualization
- Computational cost
- Course of dimensionality

2/6 manifold lerning

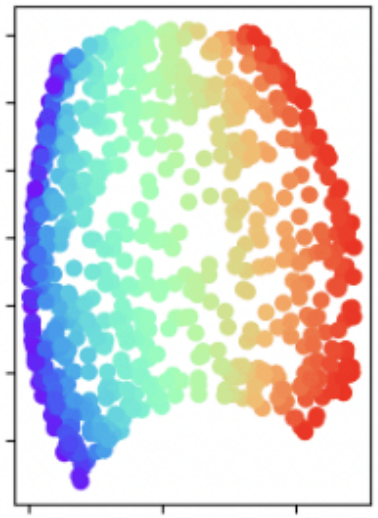
Manifold Learning with 1000 points, 10 neighbors



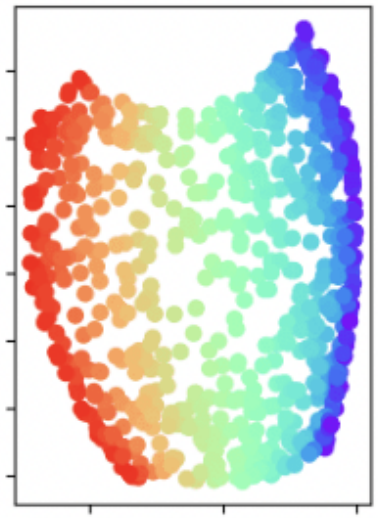
LLE (0.057 sec)



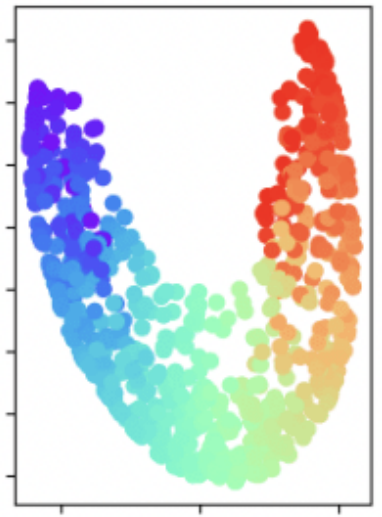
LTSA (0.79 sec)



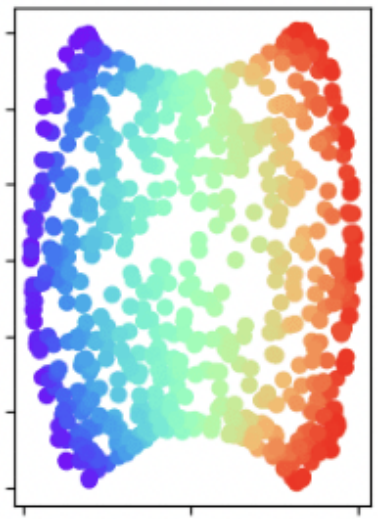
Hessian LLE (0.66 sec)



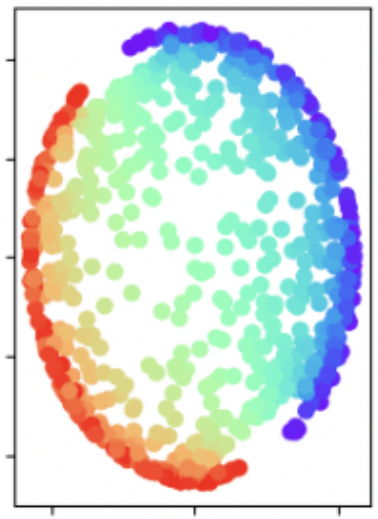
Modified LLE (1.2 sec)



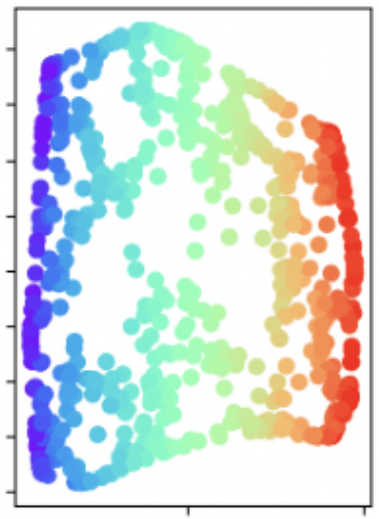
Isomap (0.19 sec)



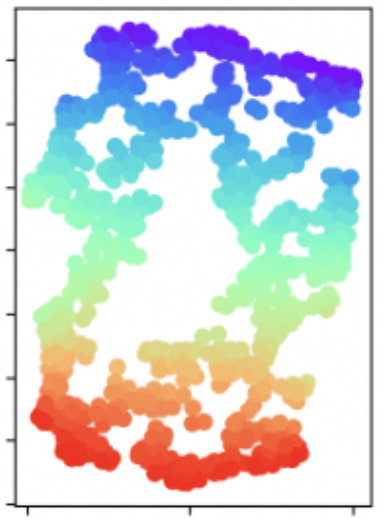
MDS (0.7 sec)



Spectral Embedding (0.041 sec)

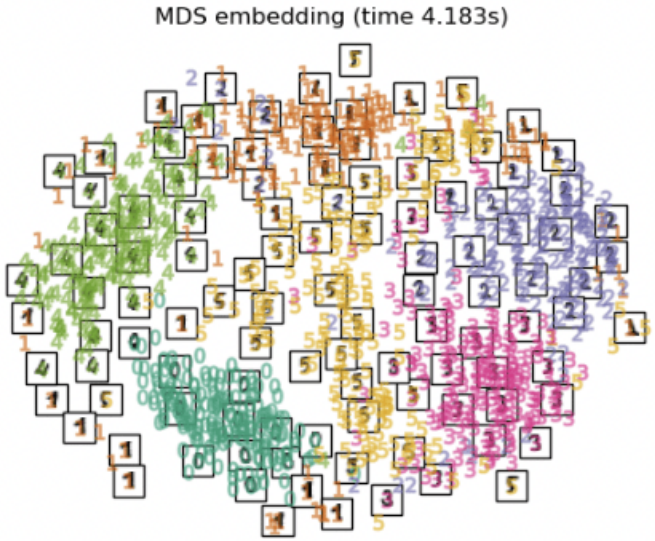


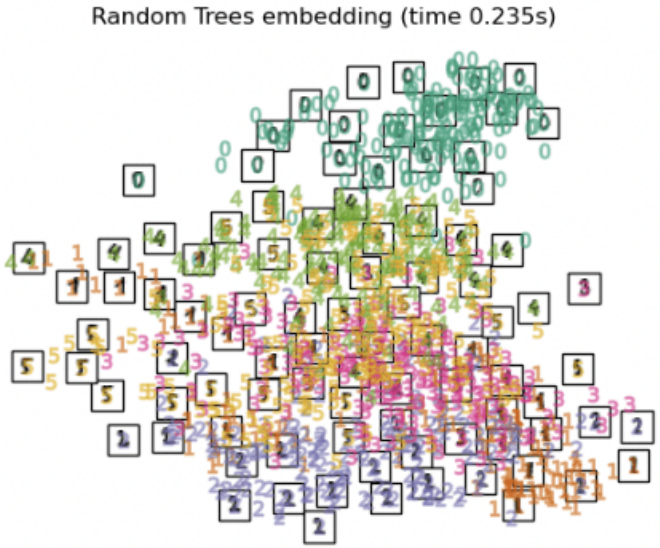
t-SNE (3.6 sec)

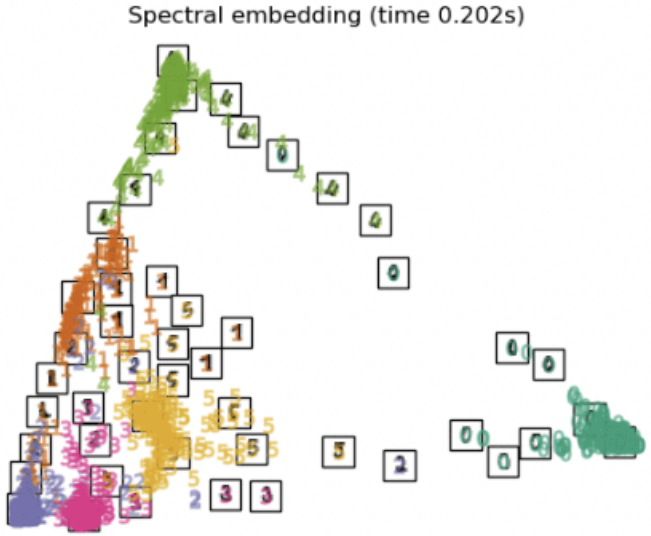


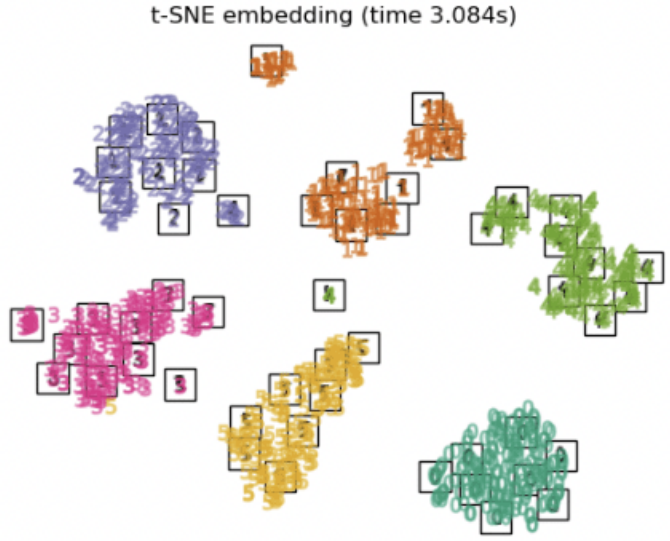
MANIFOLD LEARNING ON MNIST

https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html



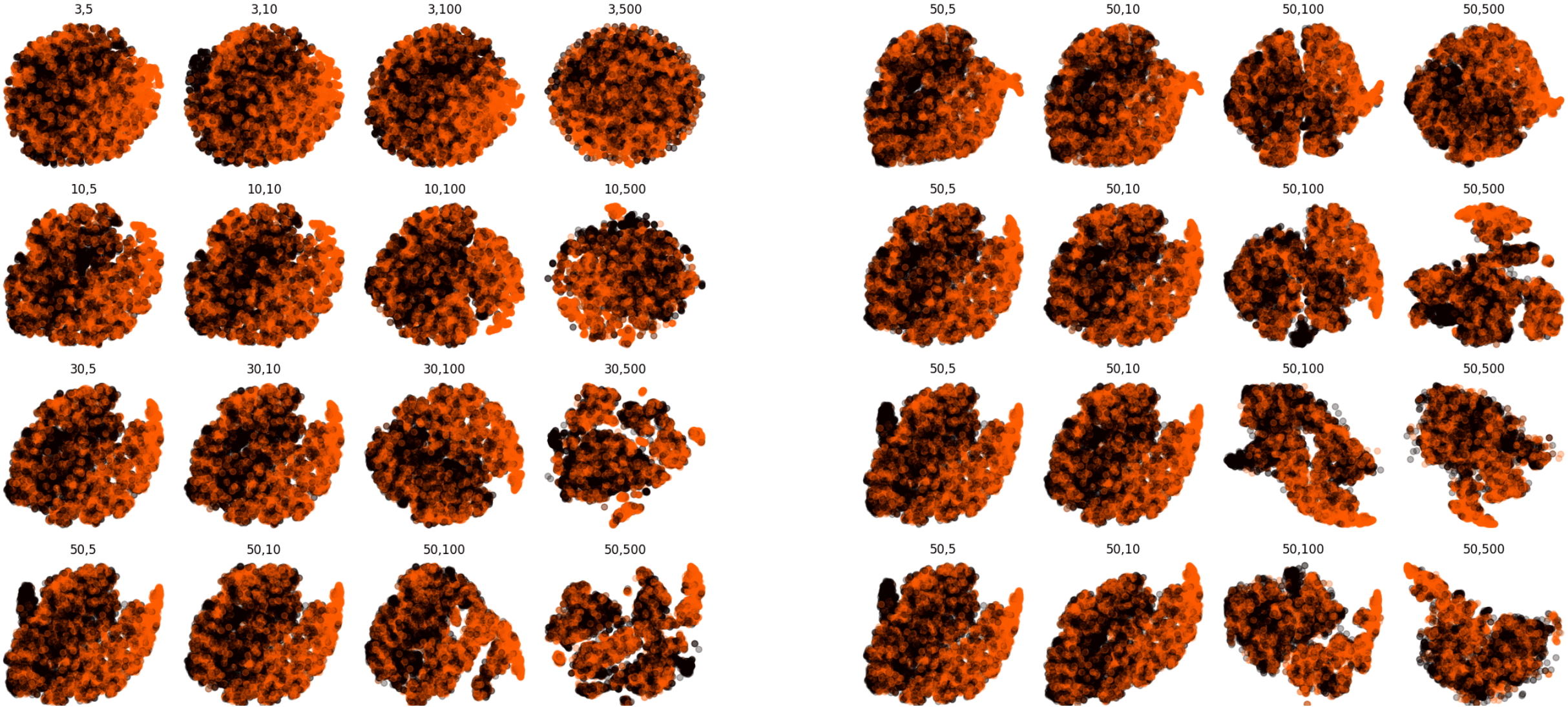






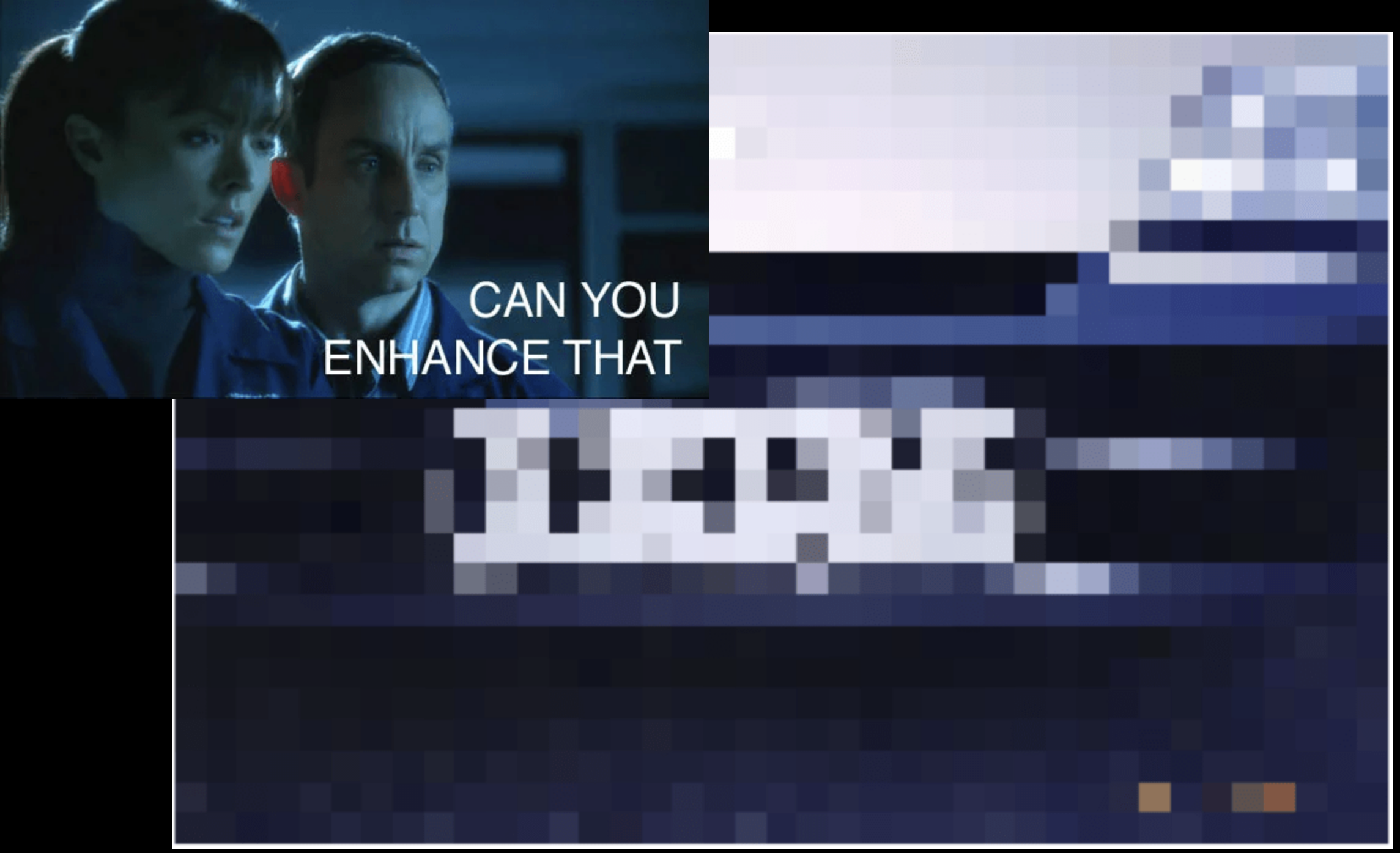
Higgs boson Kaggle dataset Training set of 250000 events, with an ID column, 30 feature columns, a weight column and a label column.

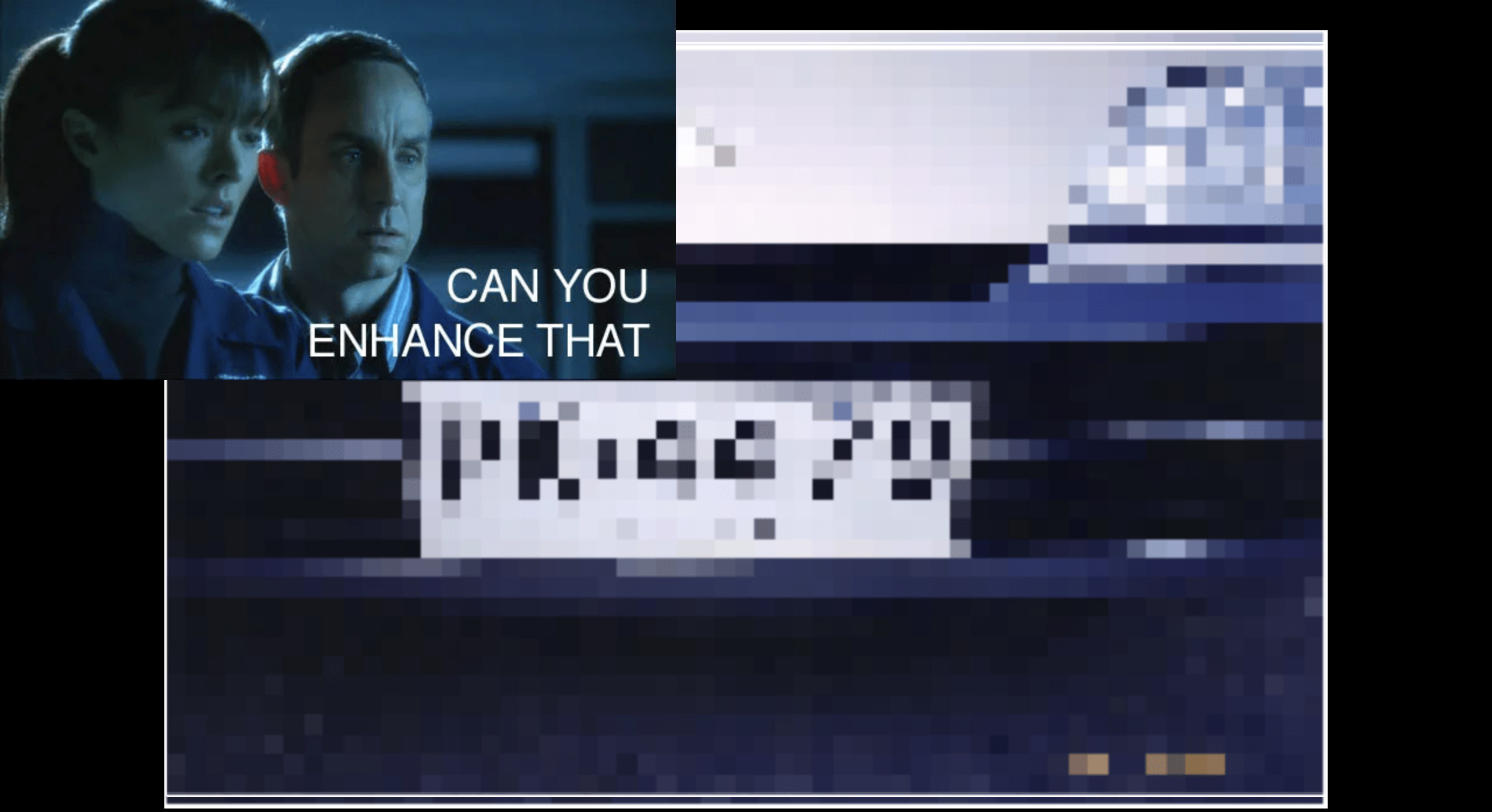
perplexity, early exaggeration

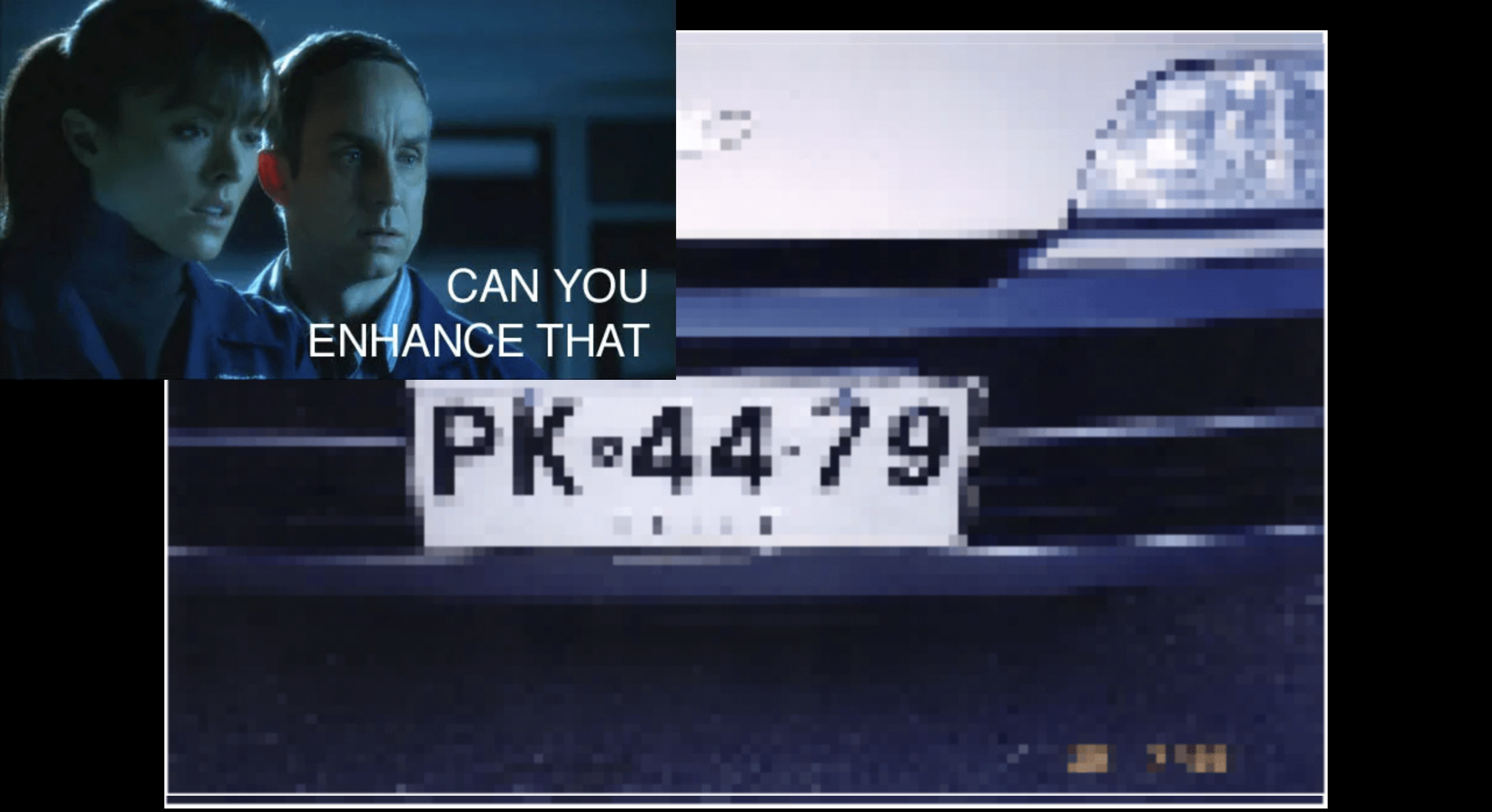


3/6

Super resolution in society and astronomy


CAN YOU
ENHANCE THAT


CAN YOU
ENHANCE THAT


CAN YOU
ENHANCE THAT
PK-44-79

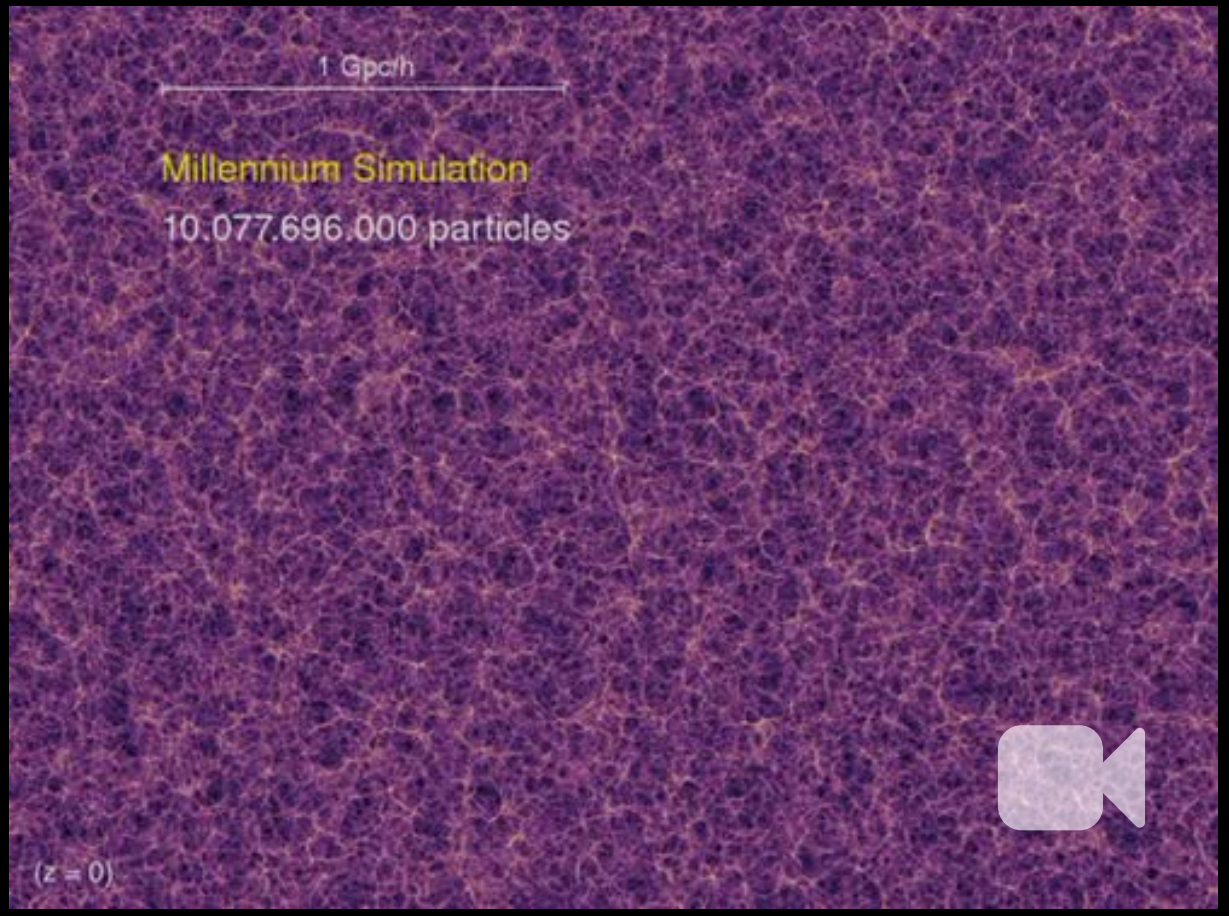

CAN YOU
ENHANCE THAT
PK·44·79
CHILE
25 7 '99

remember the timae when simulations drove astronomy...

Experiment driven

Theory driven | Falsifiability

Simulations | Probabilistic inference | Computation



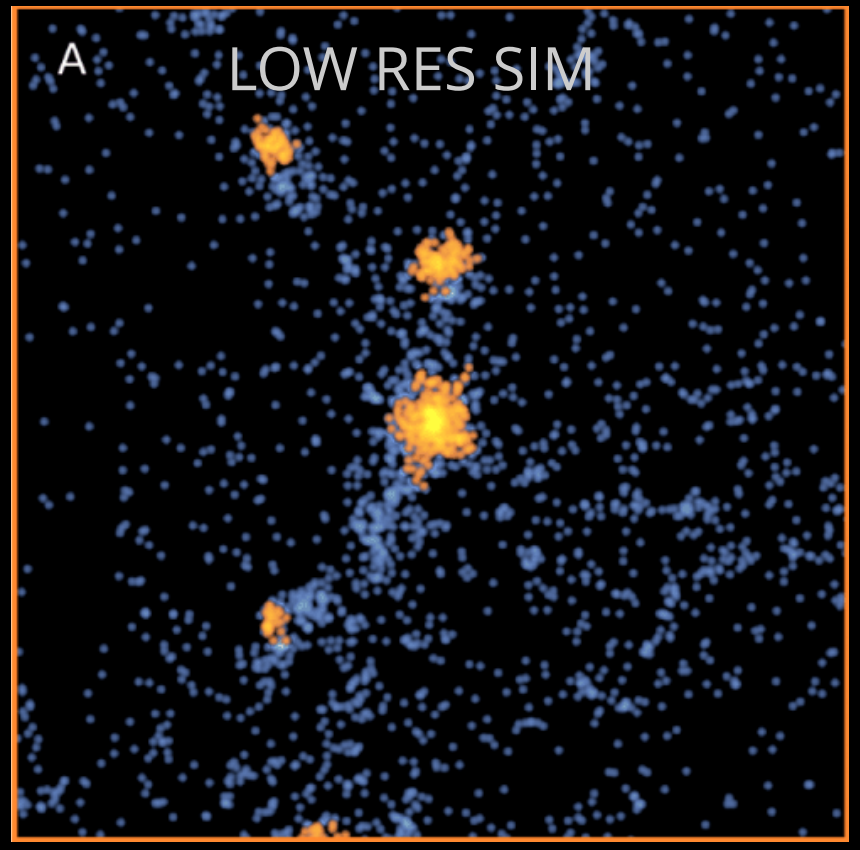
The Millennium Run used more than 10^10 particles to trace the evolution of the matter distribution in a cubic region of the Universe 500/h Mpc on a side (~over 2 billion light-years on a side), and has a spatial resolution of 5/h kpc. ~20M galaxies.

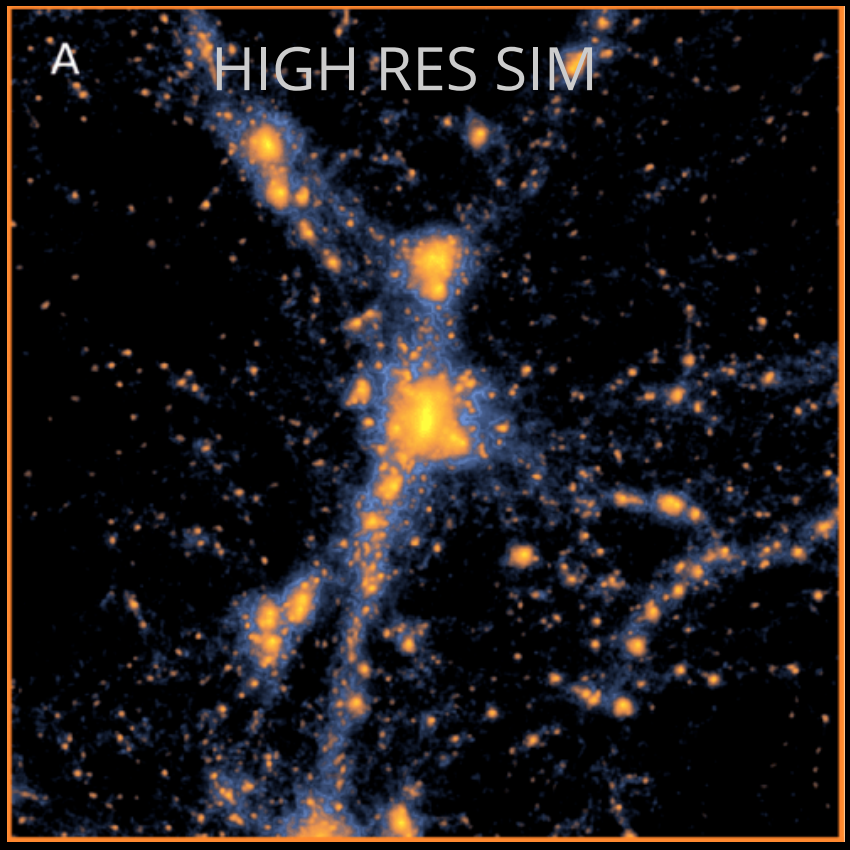
350 000 processor hours of CPU time, or 28 days of wall-clock time. Springel+2005

https://wwwmpa.mpa-garching.mpg.de/galform/virgo/millennium

AI-assisted superresolution cosmological simulations

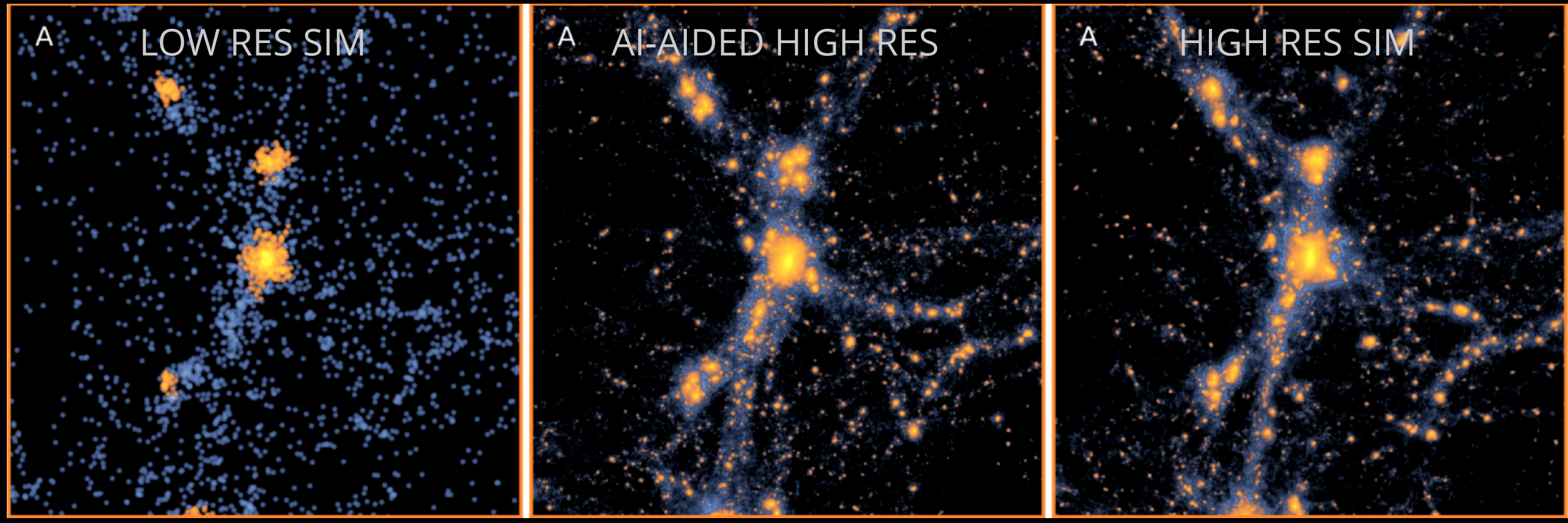
Yin Li+2021





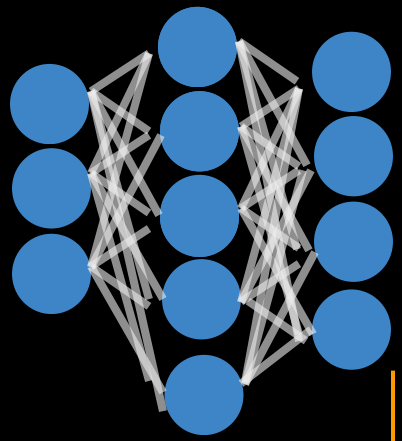
AI-assisted superresolution cosmological simulations

Yin Li+2021



AI-assisted superresolution cosmological simulations

Yin Li+2021

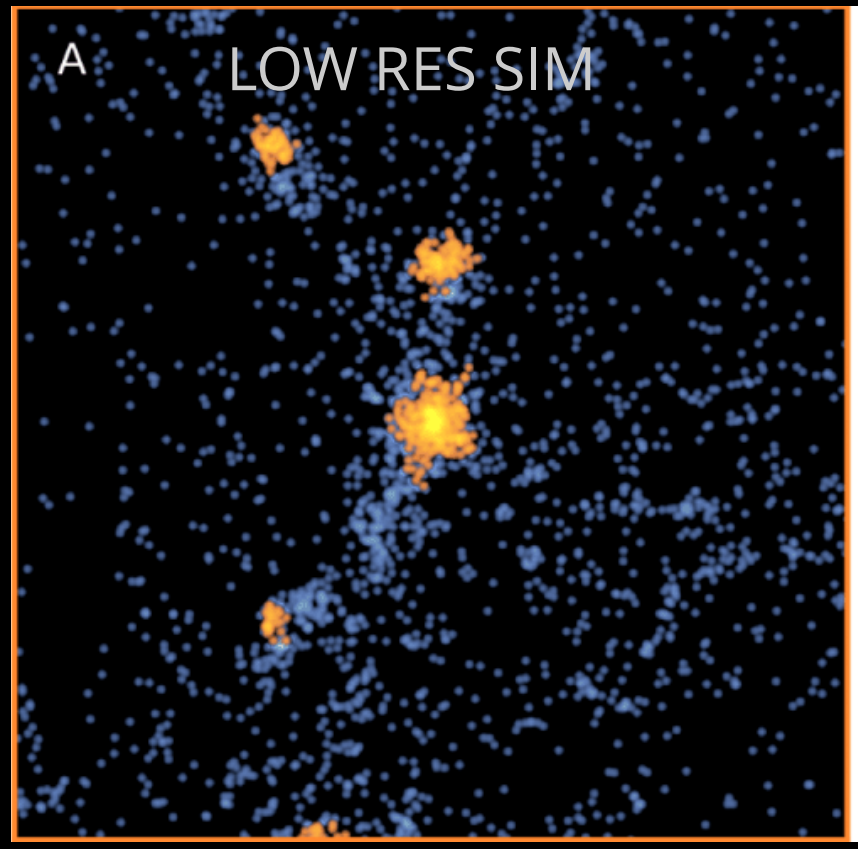


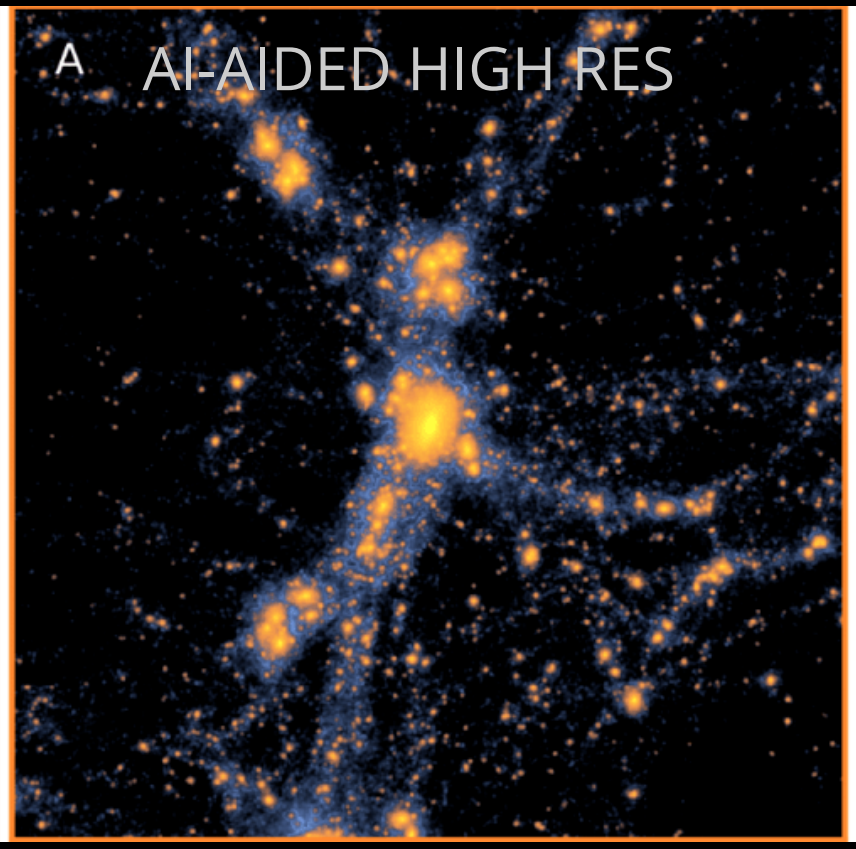
loss = *D*(OUTPUT-TARGET)

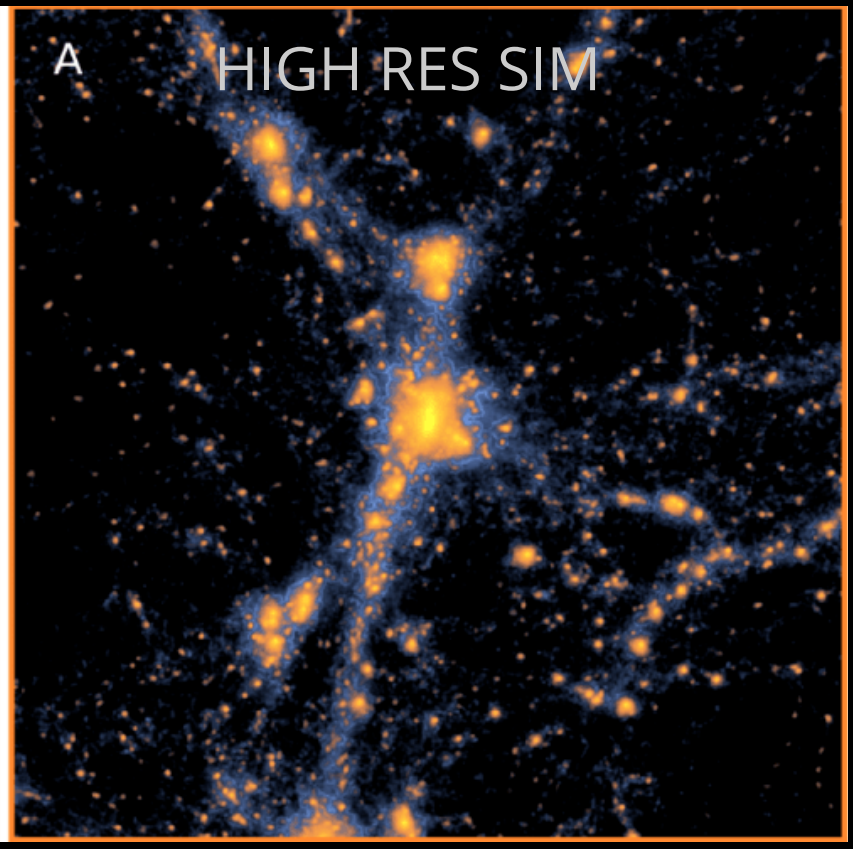
INPUT

OUTPUT

TARGET



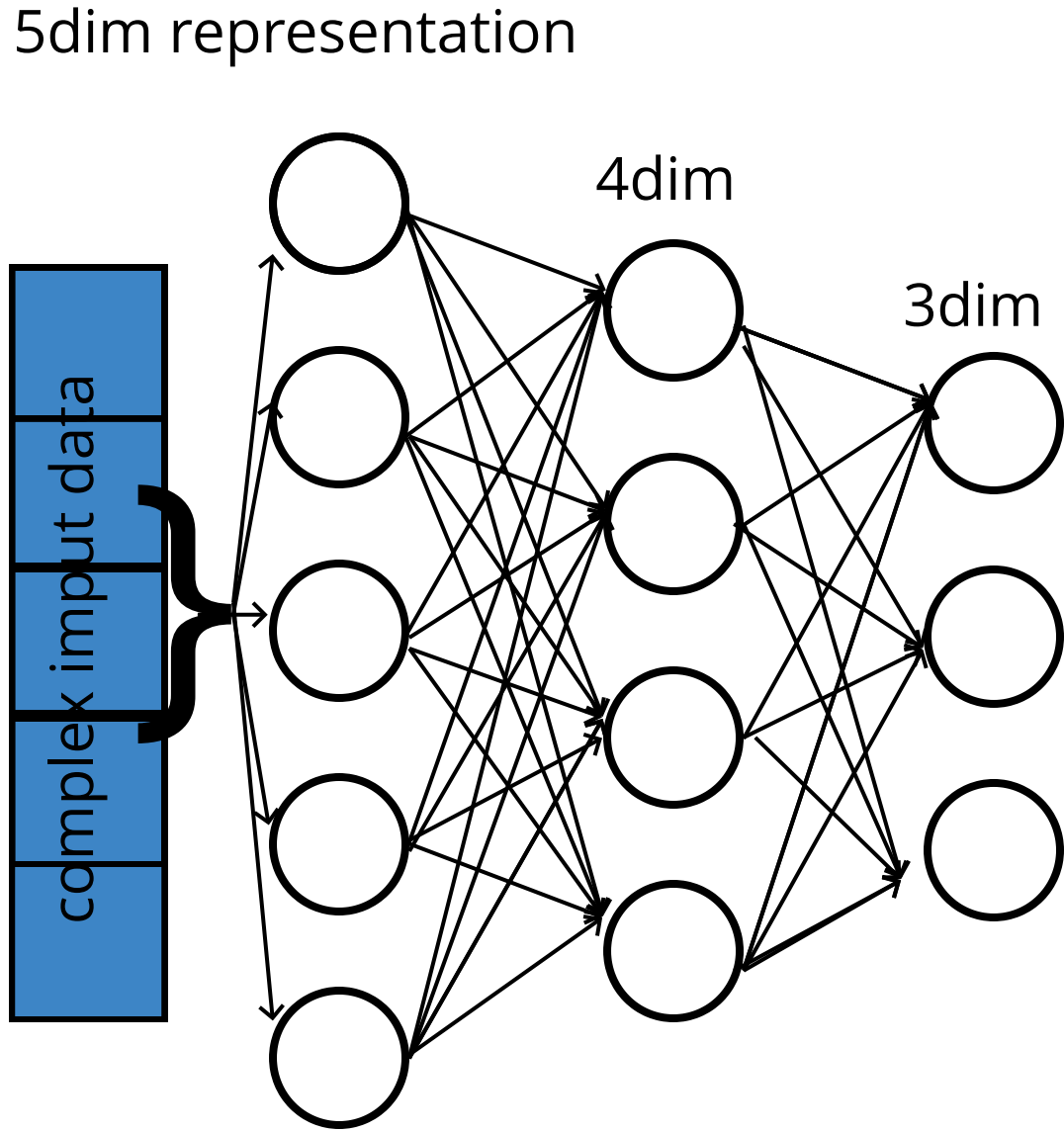




4/6 AUTOENCODERS

Unsupervised learning with Neural Networks

What do NN do? approximate complex functions with series of modified linear functions



perceptrons

Perceptrons are ***linear classifiers:*** makes its predictions based on a linear predictor function

combining a set of weights (=parameters) with the feature vector.

$$y = wx + b$$

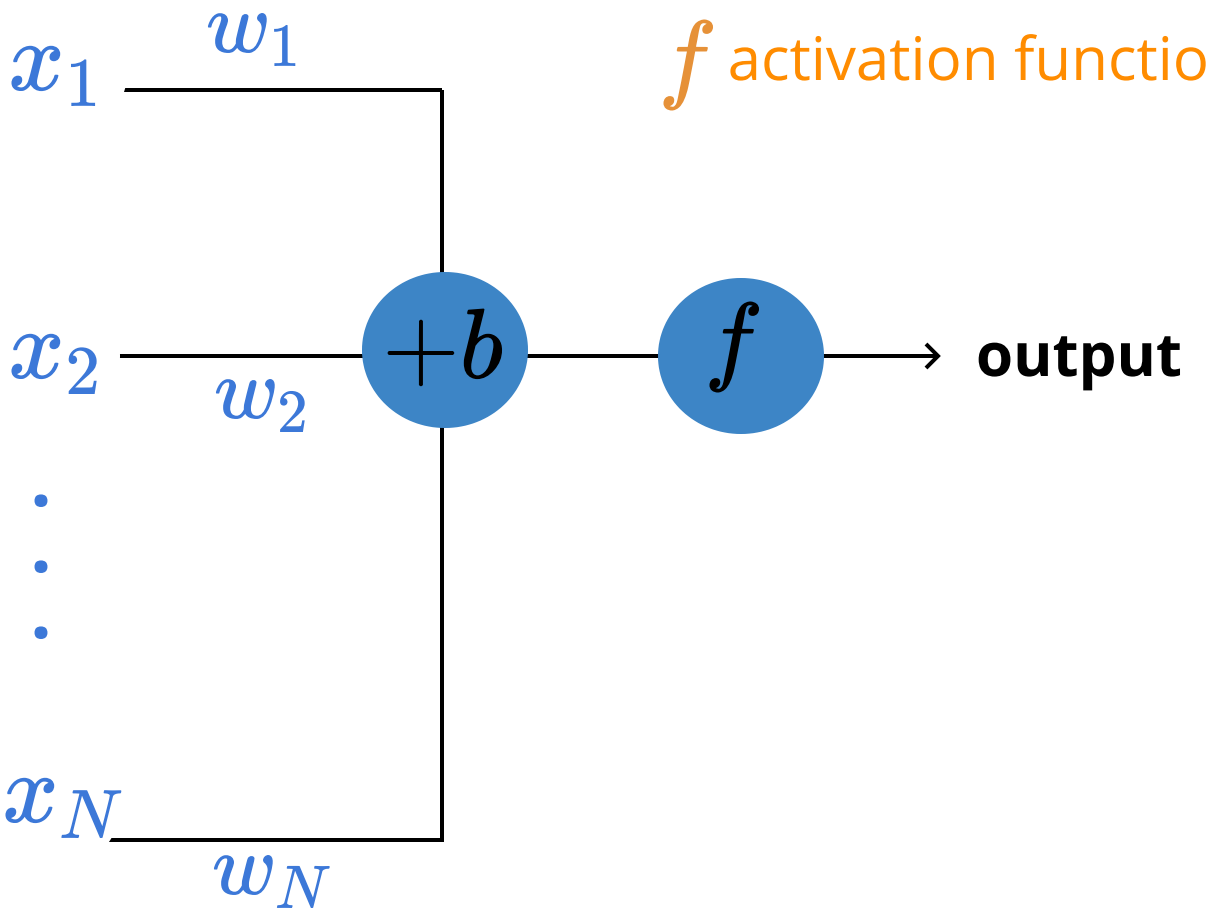
$$y = \sum_i w_i x_i + b$$

$$y = f(\sum_i w_i x_i + b)$$

w_i weights

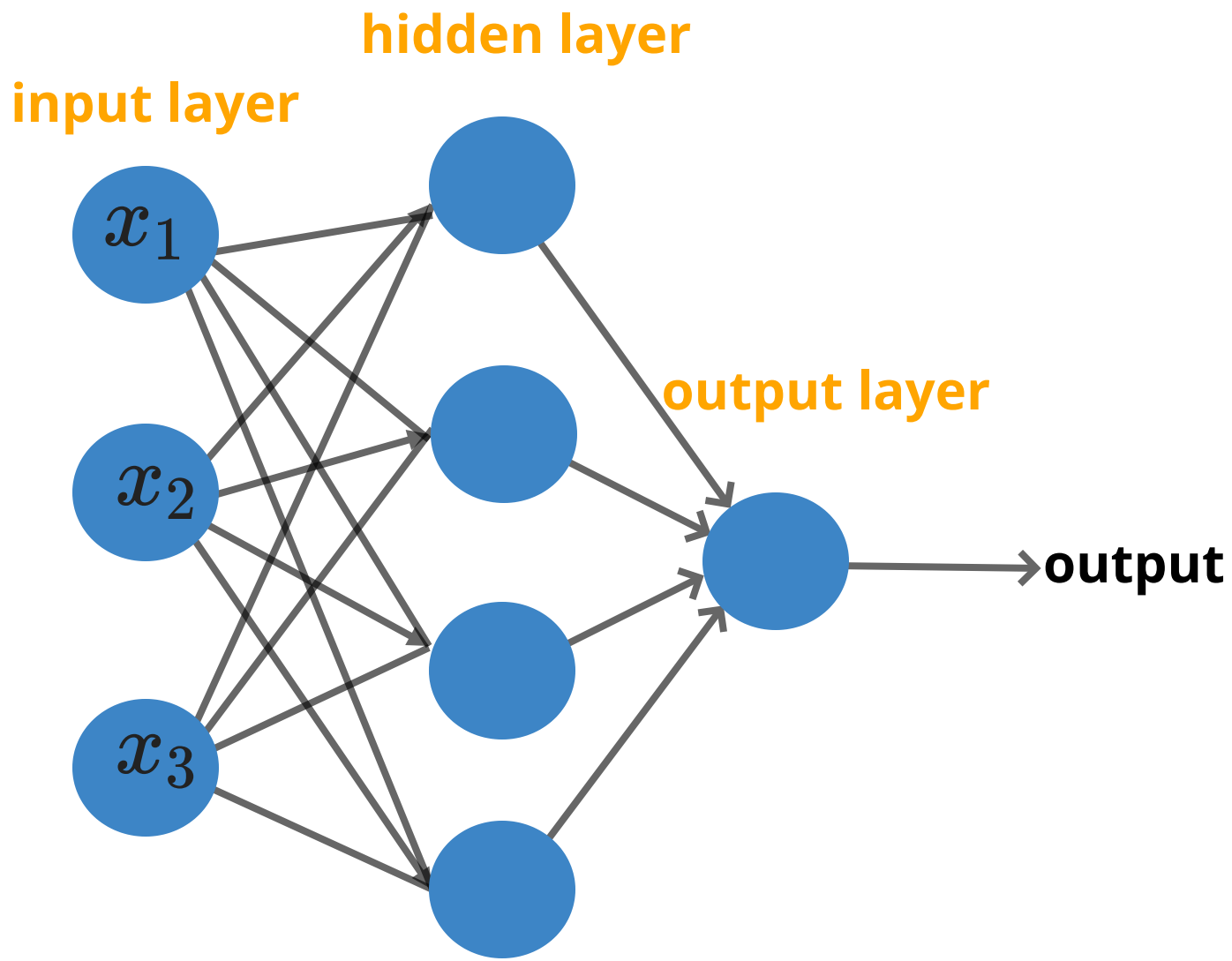
b bias

f activation function



multilayer perceptron

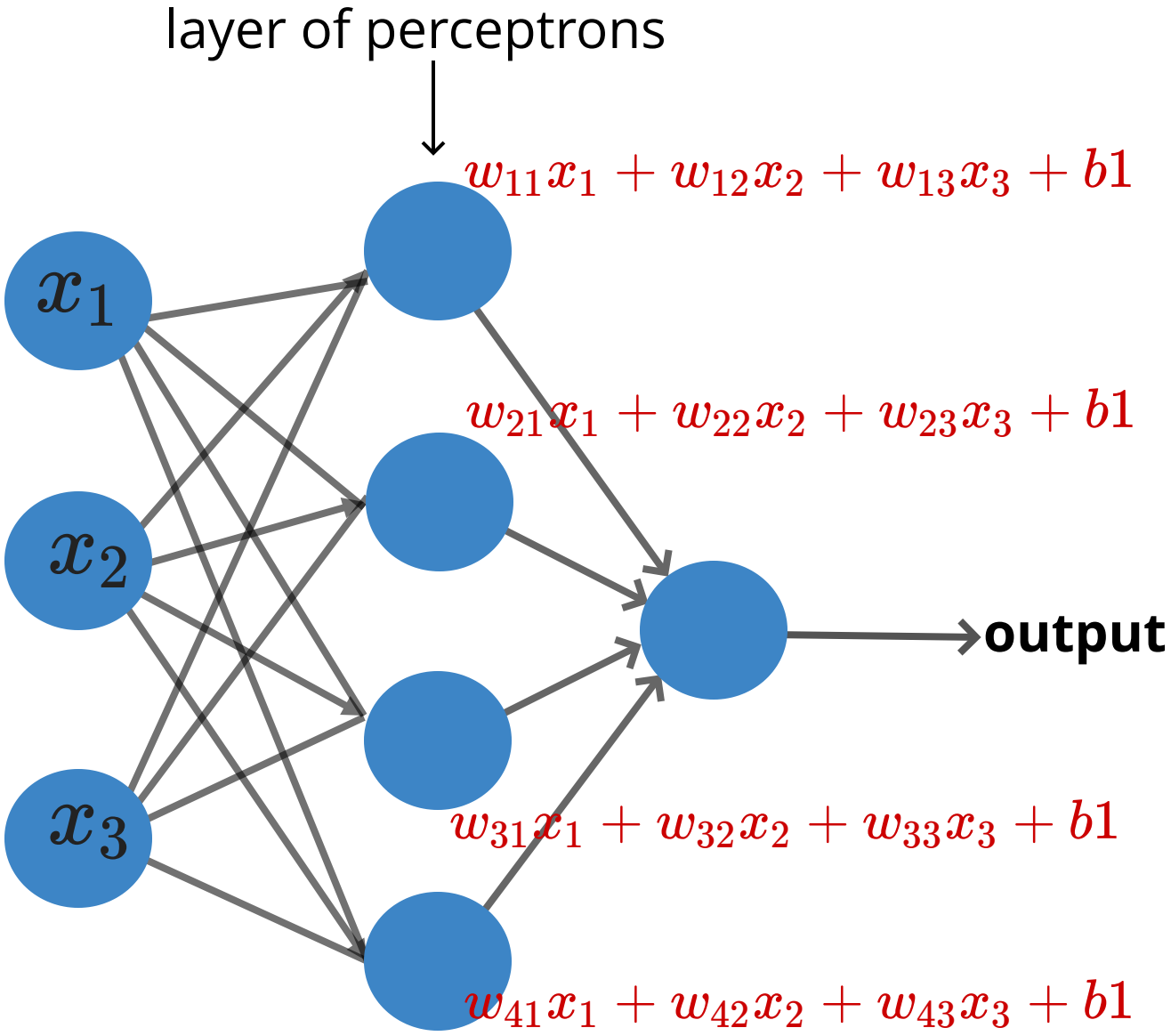
1970: multilayer perceptron architecture



Fully connected: all nodes go to all nodes of the next layer.

multilayer perceptron

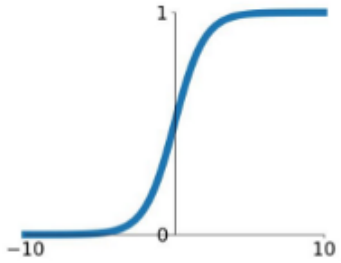
Fully connected: all nodes go to all nodes of the next layer.



activation functions

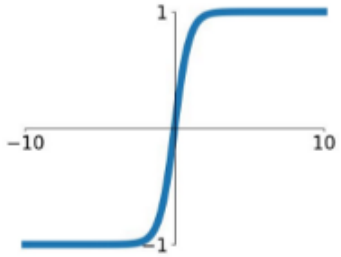
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



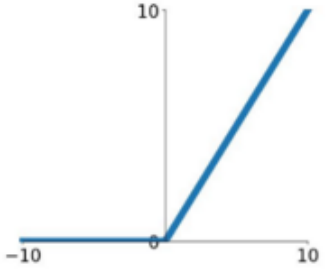
tanh

$$\tanh(x)$$



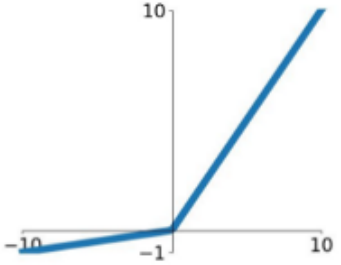
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

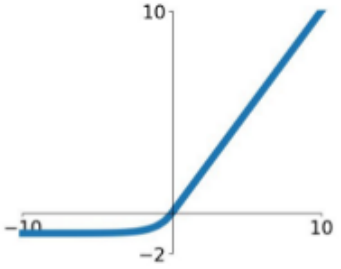


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

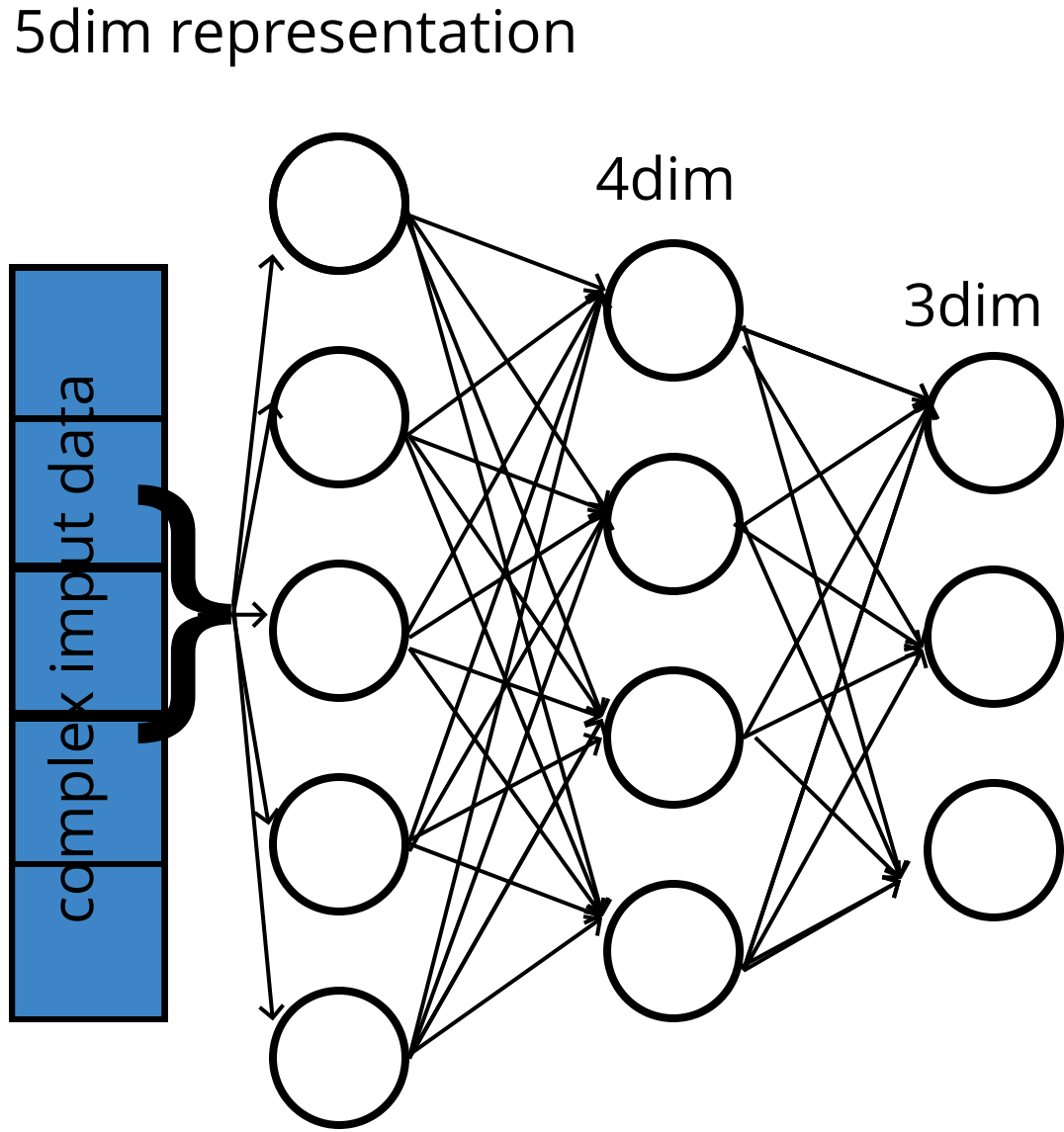
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Unsupervised learning with Neural Networks

What do NN do? approximate complex functions with series of modified linear functions

To do that they extract information from the data: **each layer of the DNN produces a representation of the data a "latent representation"**



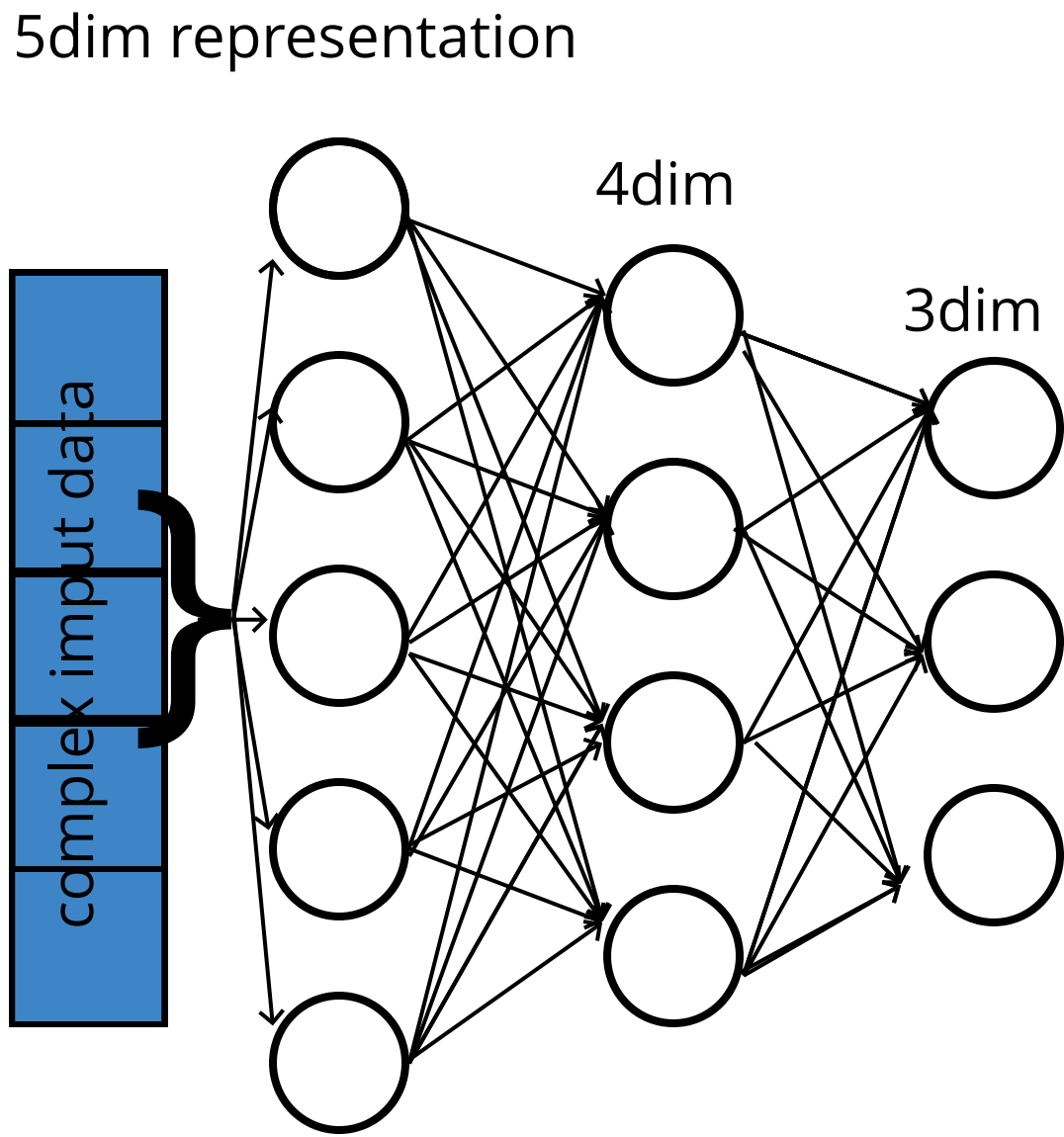
Unsupervised learning with Neural Networks

What do NN do? approximate complex functions with series of modified linear functions

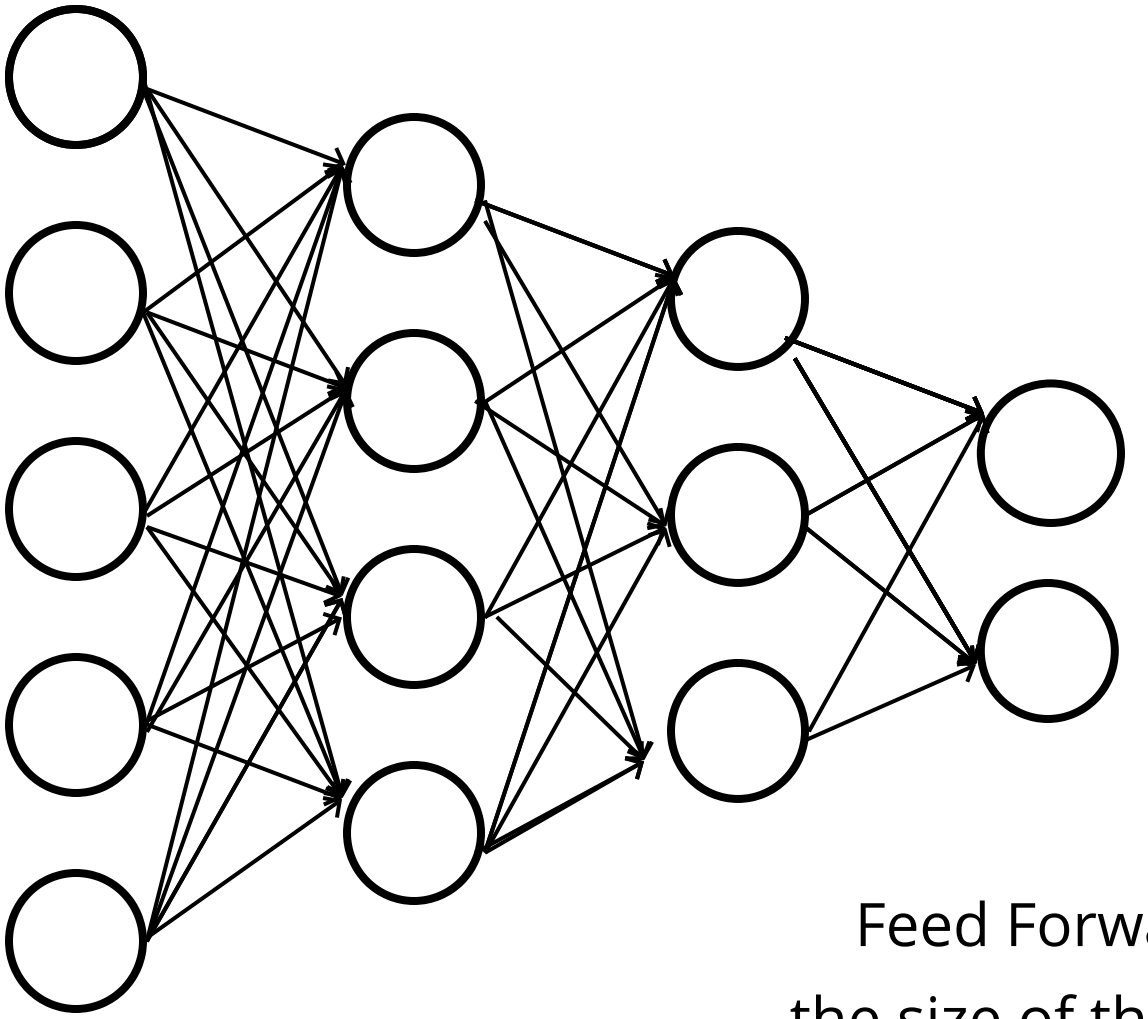
To do that they extract information from the data: **each layer of the DNN produces a representation of the data a "latent representation"**

The dimensionality of that latent representation is determined by the size of the layer

.... so if my layers are smaller what I have is a compact representation of the data



Autoencoder Architecture

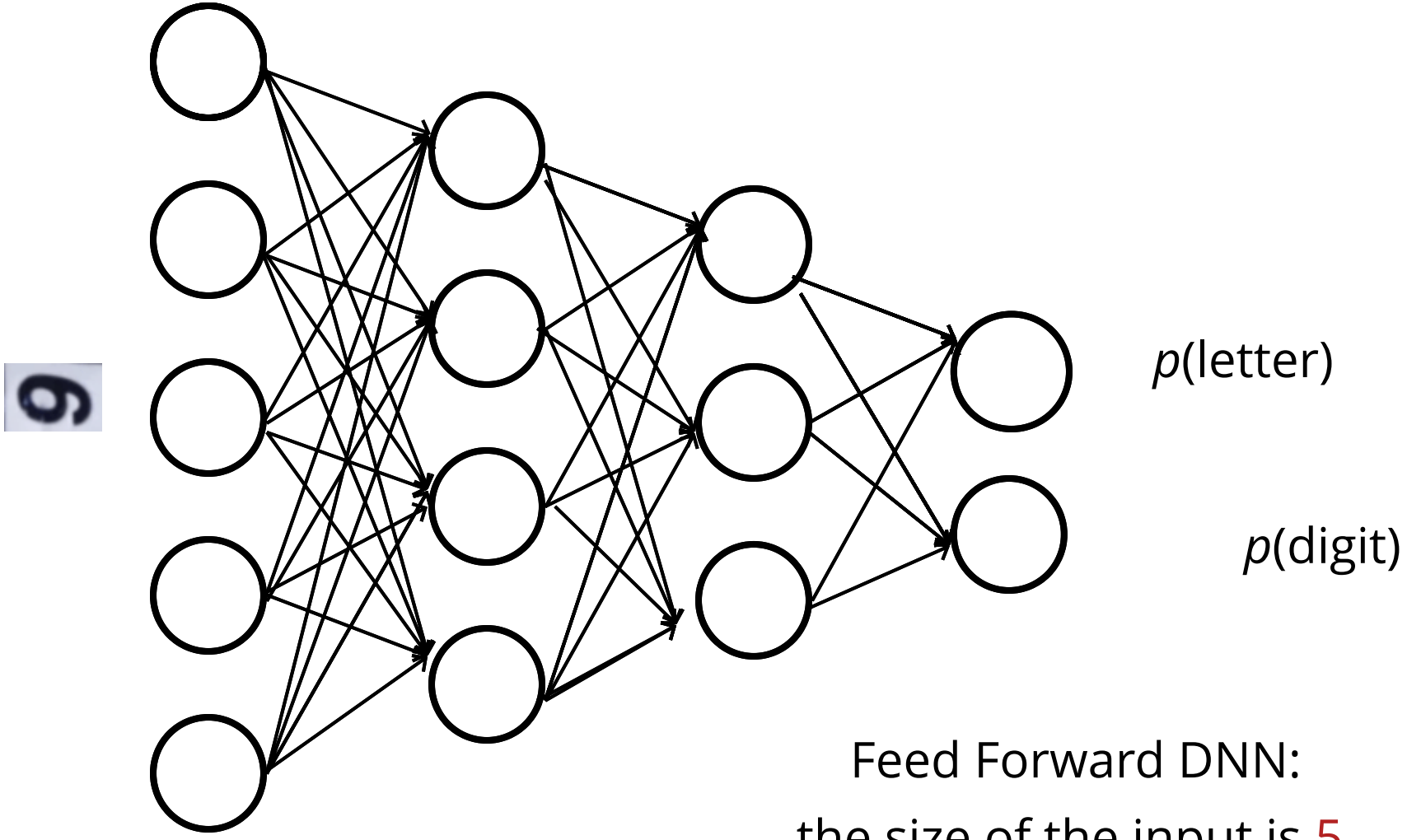


Feed Forward DNN:

the size of the input is 5,

the size of the last layer is 2

Autoencoder Architecture



p(letter)

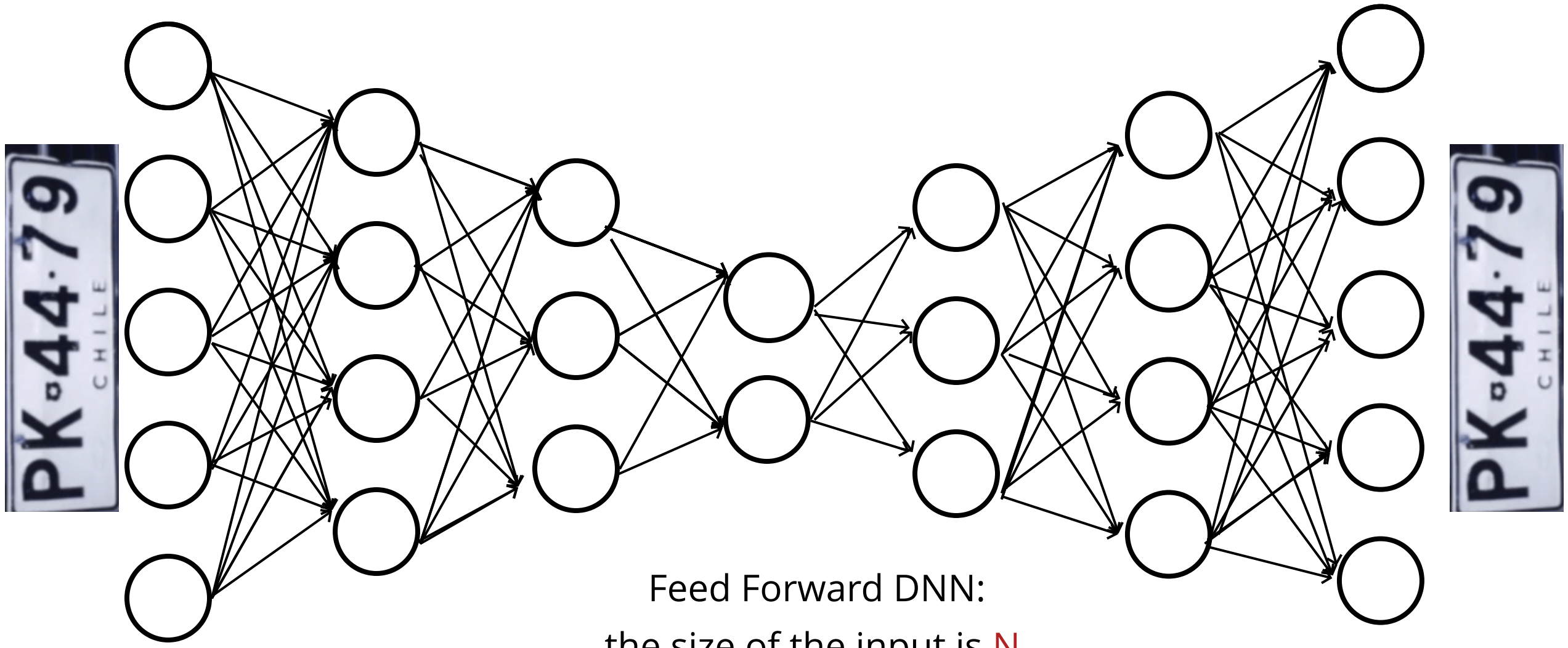
p(digit)

Feed Forward DNN:

the size of the input is 5,

the size of the last layer is 2

Autoencoder Architecture

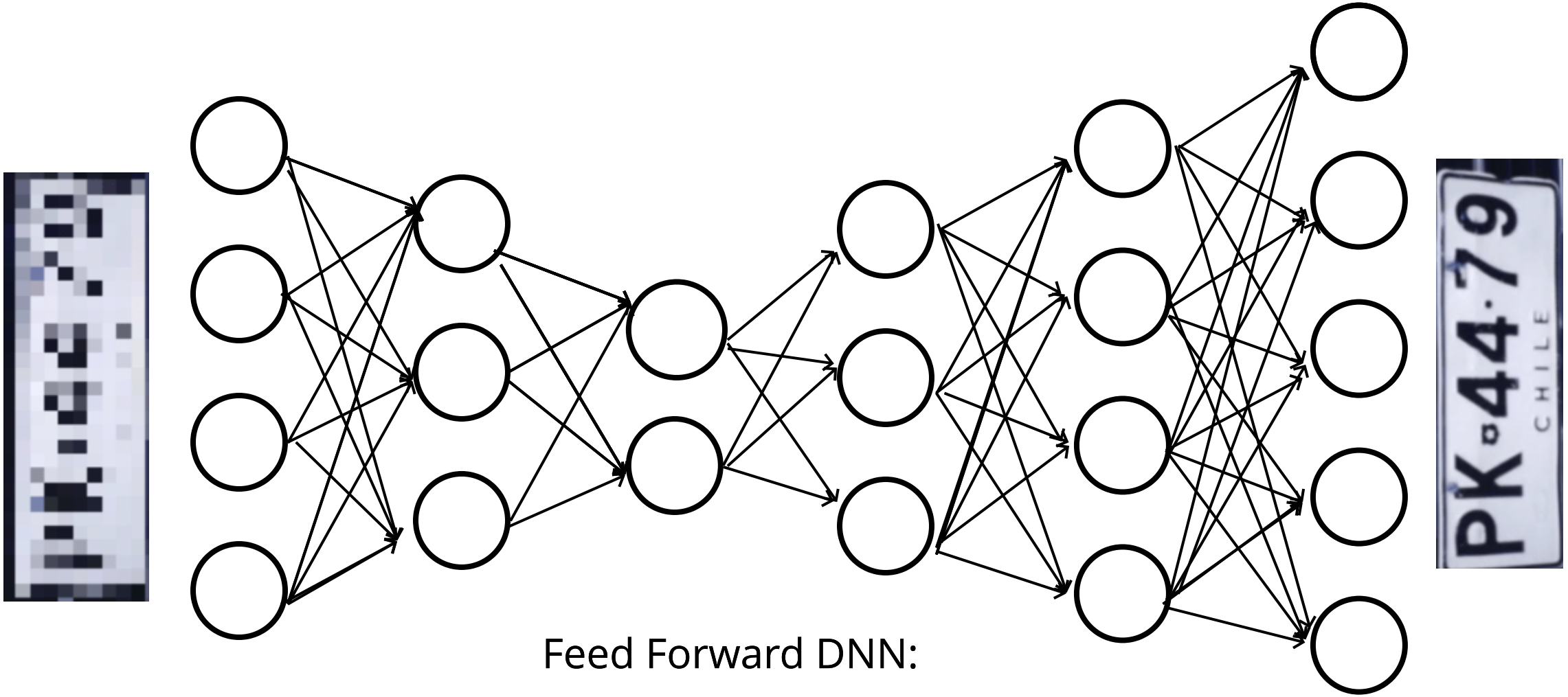


Feed Forward DNN:

the size of the input is N,

the size of the last layer is N

Autoencoder Architecture



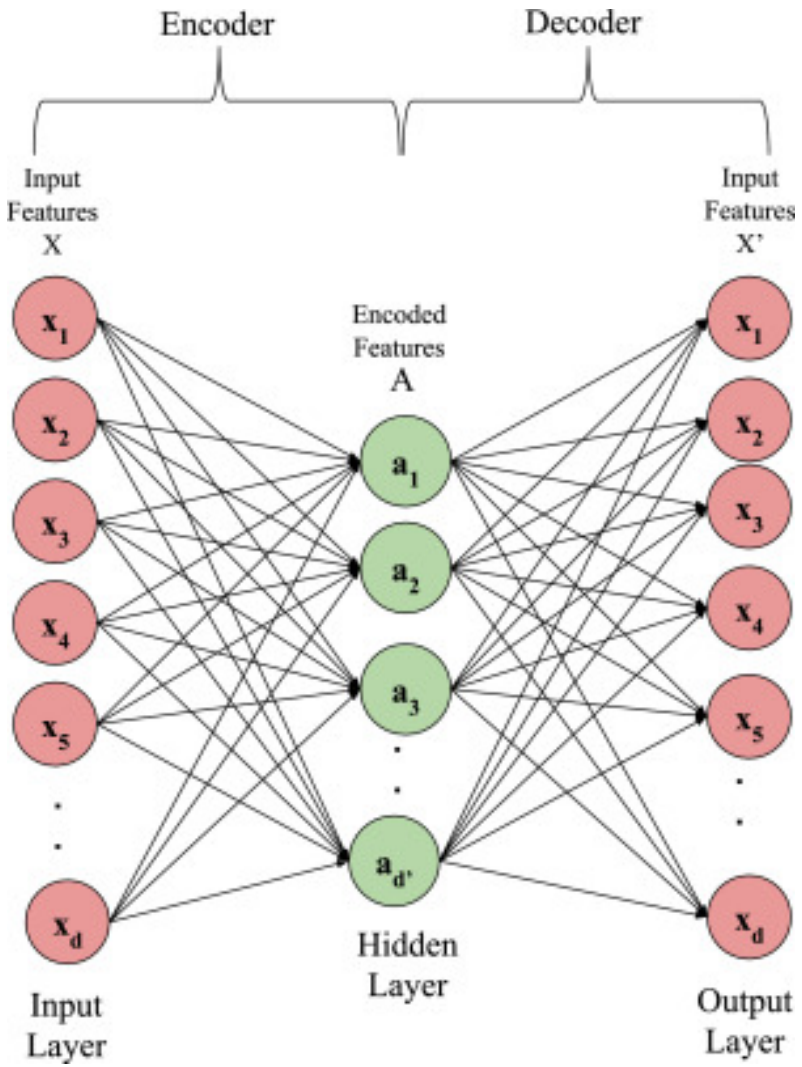
Feed Forward DNN:

the size of the input is N,

the size of the last layer is N

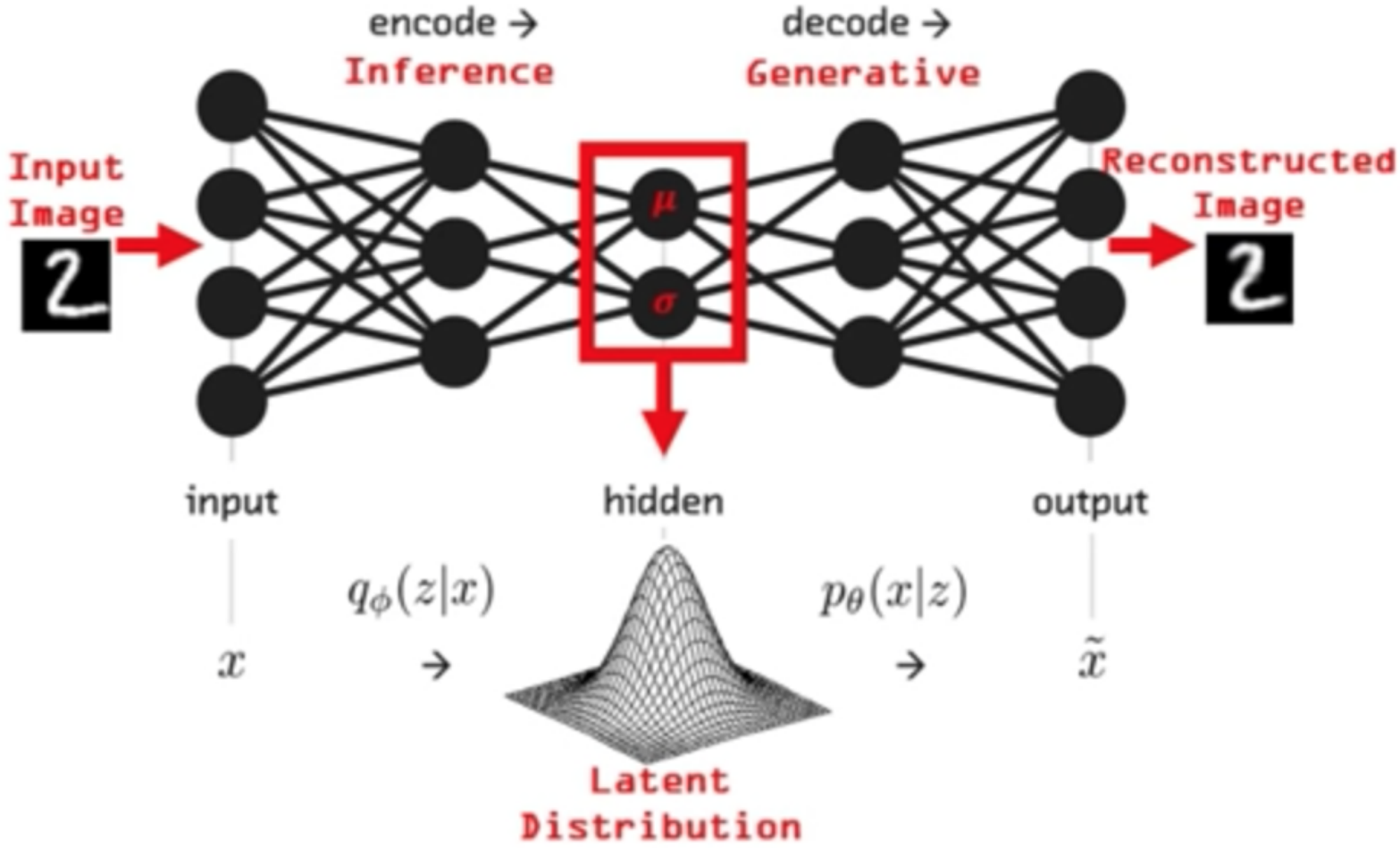
Autoencoder Architecture

https://link.springer.com/chapter/10.1007/978-981-13-6661-1_3



- **Encoder**: outputs a lower dimensional representation ***z*** of the data ***x*** (similar to PCA, tSNE...)
- **Decoder**: Learns how to reconstruct ***x*** given ***z:*** learns $p(x|\boldsymbol{z})$

Variational Autoencoder Architecture



The hidden representation is assumed (forced) to be Gaussian. The bottle-neck layer represents the mean and standard deviation of that distribution

Building a DNN
with keras and tensorflow

Trivial to build, but the devil is in the details!

Building a DNN with keras and tensorflow

Trivial to build, but the devil is in the details!

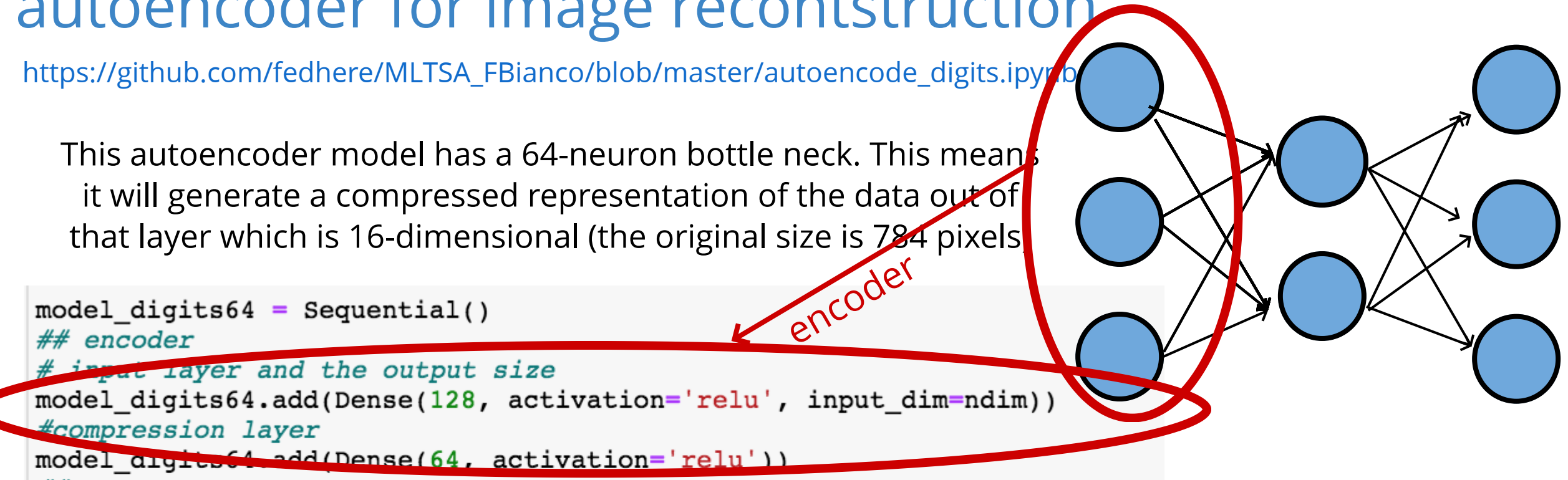
```
from keras.models import Sequential
#can upload pretrained models from keras.models
from keras.layers import Dense,  Conv2D, MaxPooling2D
#create model
model = Sequential()


#create the model architecture by adding model layers
model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))

#need to choose the loss function, metric, optimization scheme
model.compile(optimizer='adam', loss='mean_squared_error')

#need to learn what to look for - always plot the loss function!
model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=20, batch_size=100, verbose=1)
#note that the model allows to give a validation test,
#this is for a 3fold cross valiation: train-validate-test
#predict
```

Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```

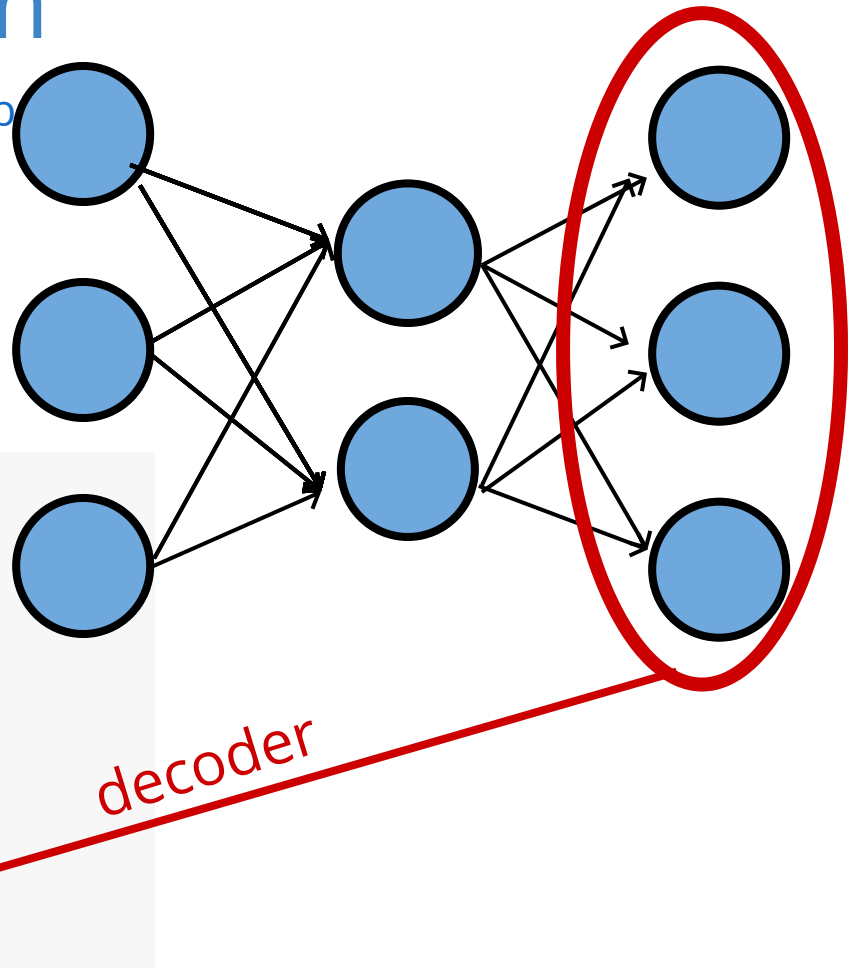


Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```



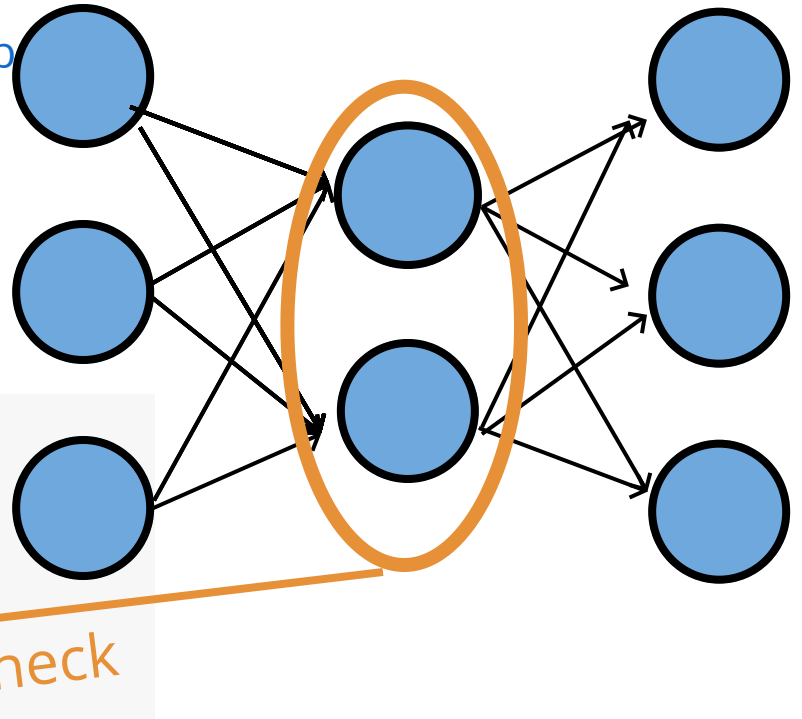
Building a DNN with keras and tensorflow

autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

This autoencoder model has a 64-neuron bottle neck. This means it will generate a compressed representation of the data out of that layer which is 16-dimensional (the original size is 784 pixels)

```
model_digits64 = Sequential()
## encoder
# input layer and the output size
model_digits64.add(Dense(128, activation='relu', input_dim=ndim))
#compression layer
model_digits64.add(Dense(64, activation='relu'))
## deencoder
#decompression layer, same size as in the encoder
model_digits64.add(Dense(128, activation='relu'))
#output layer, same size as input
model_digits64.add(Dense(ndim, activation='linear'))
```



Building a DNN with keras and tensorflow autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

```
# choose the optimizer and loss appropriately!
model_digits64.compile(optimizer="adadelta", loss="mean_squared_error")
```

```
print(model_digits64.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_4 (Dense)              (None, 784)               101136
=================================================================
Total params: 218,192
Trainable params: 218,192
Non-trainable params: 0
_________________________________________________________________
None
```

This simple model has 200K parameters!

My original choice is to train it with "adadelta" with a mean squared loss function, all activation functions are relu, appropriate for a linear regression

Building a DNN

with keras and tensorflow

autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

3.0.1 regression

- loss='mean_squared_error' L2: default loss to use for regression problems. => linear activation funct in output layer, one node out

alternatives: loss='mean_squared_logarithmic_error', 'mean_absolute_error' (which is L1 instead of L2)

3.0.2 binary classification

- loss='binary_crossentropy' => sigmoid activation function in output layer, one node out

alternatives: 'hinge'

3.0.3 multiclass classification

categorical encoded as numerical

- loss='categorical_crossentropy' => softmax n nodes out

onehot encoded categoridal

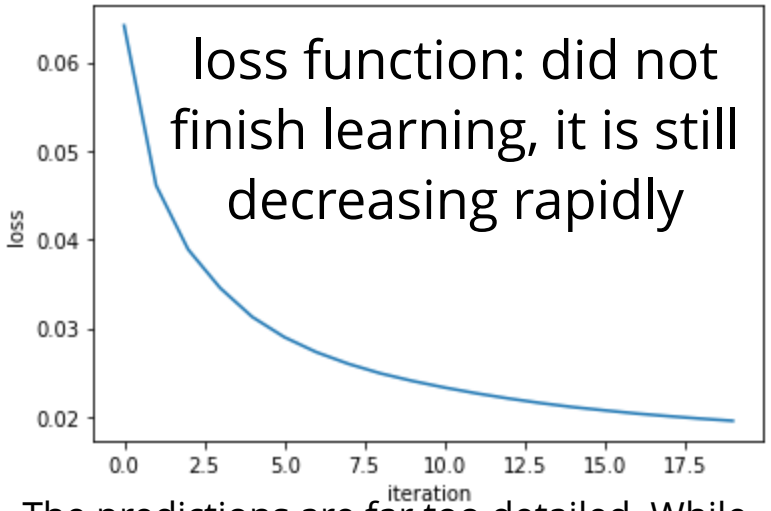
- 'parse_categorical_crossentropy' => softmax n nodes out
- 'kullback Leibler Divergence Loss' => probabilistic categorical classification; log(P/Q)

What should I choose for the loss function and how does that relate to the activation functiom and optimization?

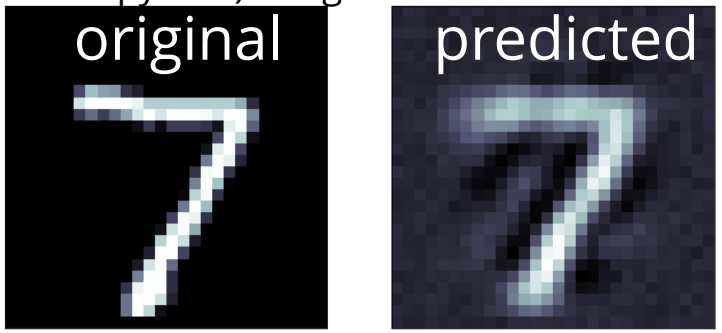
autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

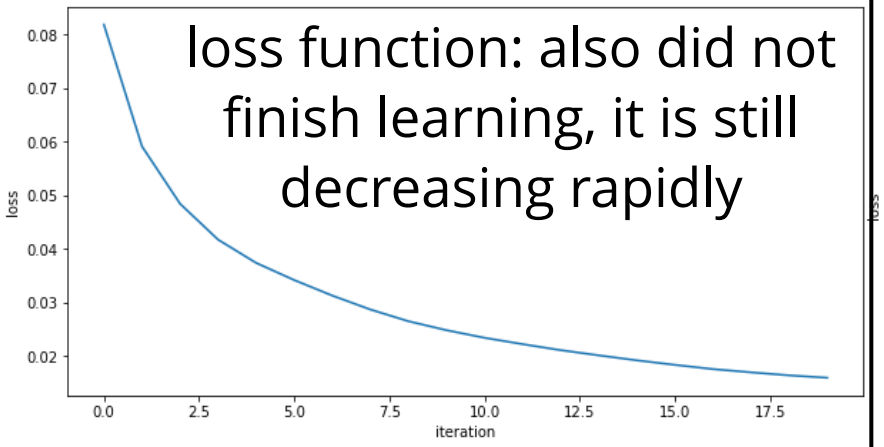
```
model_digits64.add(Dense(ndim,
                         activation='linear'))
model_digits64_sig.compile(optimizer="adadelta",
                    loss="mean_squared_error")
```



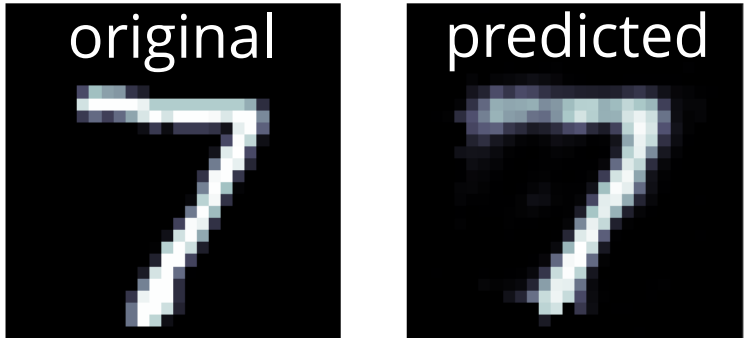
The predictions are far too detailed. While the input is not binary, it does not have a lot of details. Maybe approaching it as a binary problem (with a sigmoid and a binary cross entropy loss) will give better results



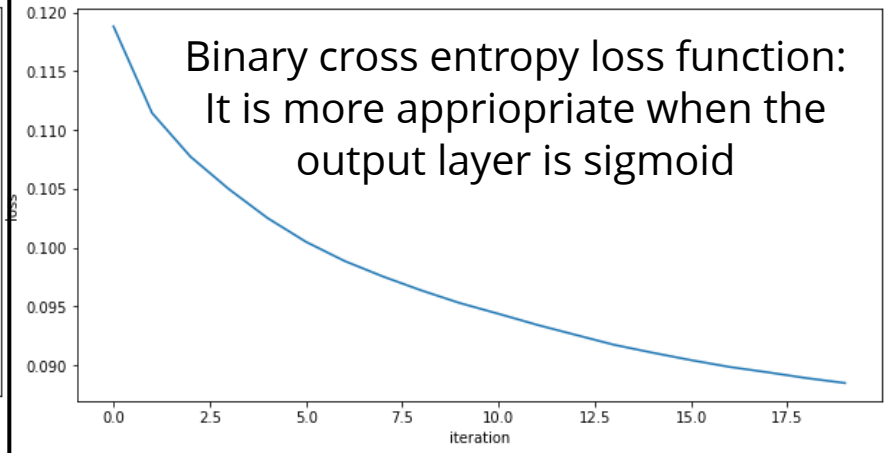

```
model_digits64_sig.add(Dense(ndim,
                             activation='sigmoid'))
model_digits64_sig.compile(optimizer="adadelta",
                           loss="mean_squared_error")
```



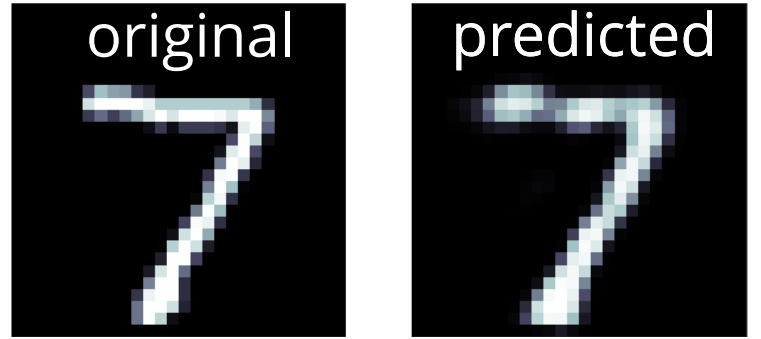
A sigmoid gives activation gives a much better result!




```
model_digits64_sig.add(Dense(ndim,
                             activation='sigmoid'))
model_digits64_bce.compile(optimizer="adadelta",
                           loss="binary_crossentropy")
```



Even better results!

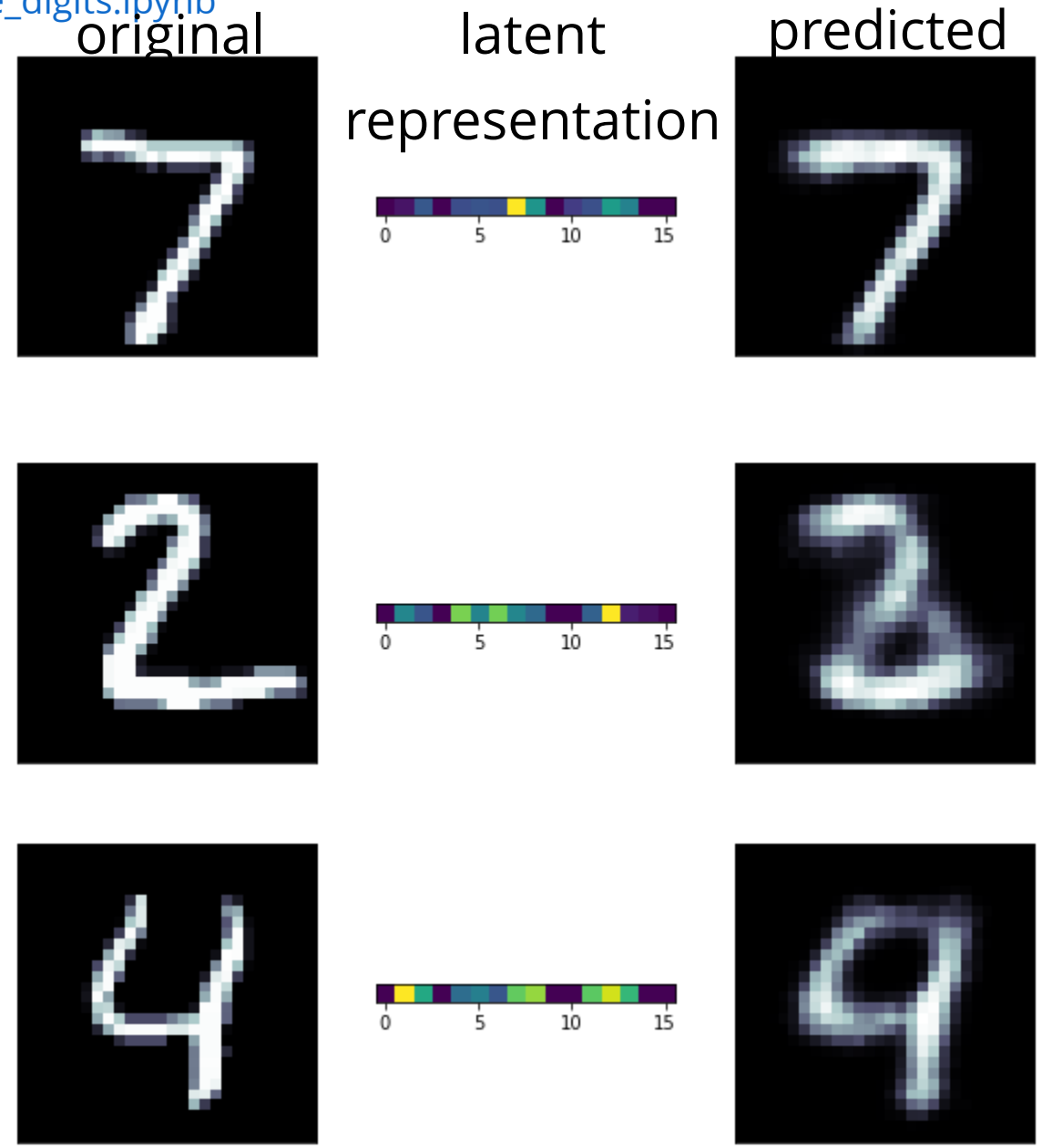


autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

A more ambitious model has a 16 neurons bottle neck: we are trying to extract 16 numbers to reconstruct the entire image! its pretty remarcable! those 16 number are ***extracted features*** from the data

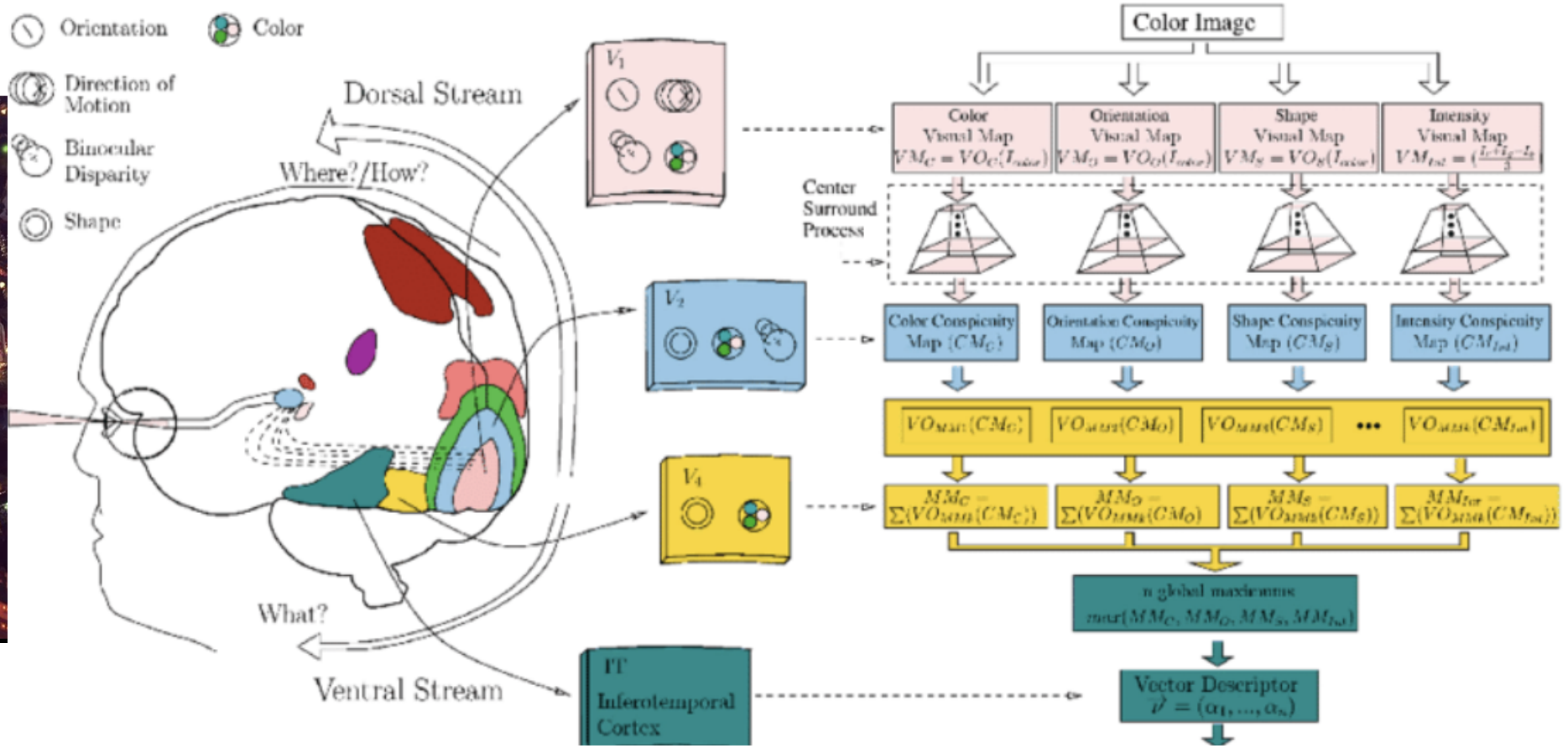
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_11 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_12 (Dense)             (None, 16)                528
_________________________________________________________________
dense_13 (Dense)             (None, 32)                544
_________________________________________________________________
dense_14 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_15 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_16 (Dense)             (None, 784)               101136
=================================================================
Total params: 223,456
Trainable params: 223,456
Non-trainable params: 0
```



5/6

convolution

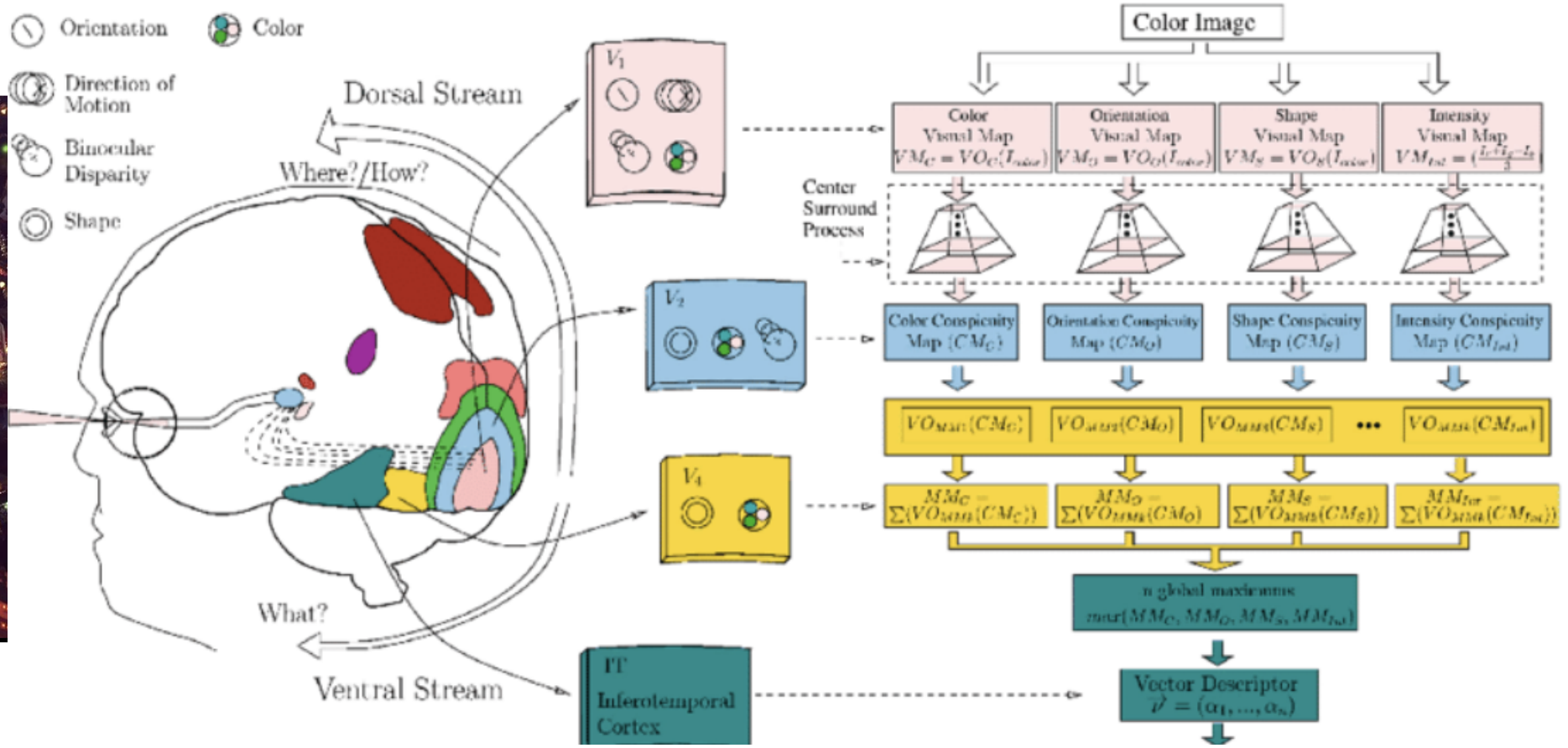
@akumadog



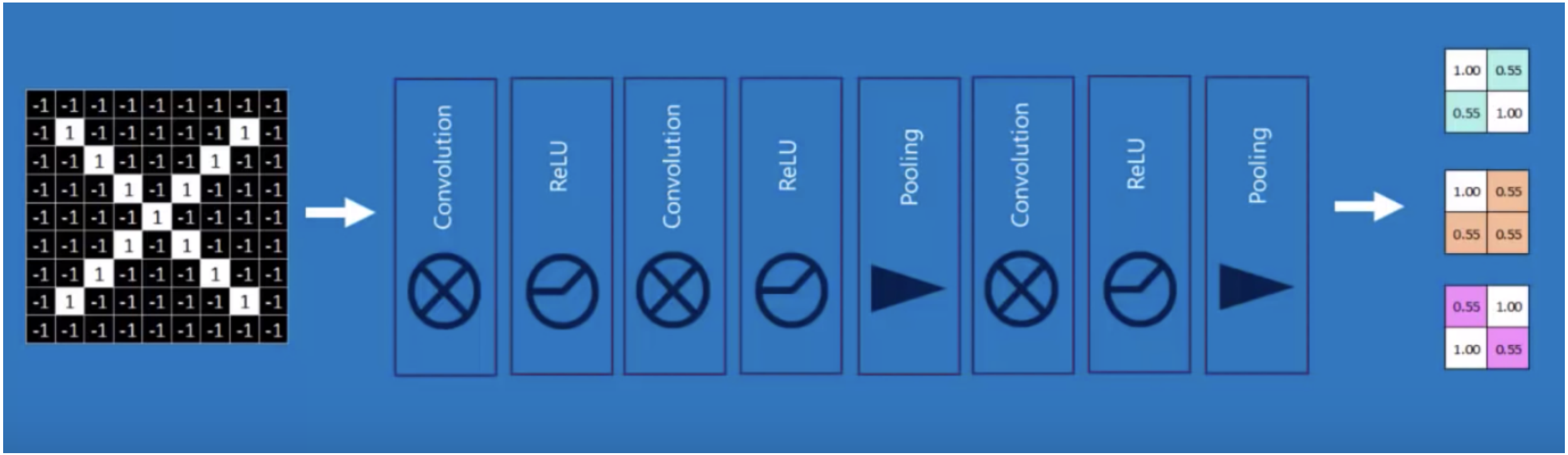
Brain Programming and the Random Search in Object Categorization

Olague et al 2017

The visual cortex learns hierarchically: first detects simple features, then more complex features and ensembles of features



CNN


-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
Convolution
ReLU
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
1.00 0.55
0.55 1.00
1.00 0.55
0.55 0.55
0.55 1.00
1.00 0.55

Convolution

Convolution

convolution is a mathematical operator on two functions

f and g

that produces a third function

$f \otimes g$

expressing *how the shape of one is modified by the other.*

Convolution Theorem

$$F(\nu) = \int_{\mathbb{R}^n} f(x) e^{-2\pi i x \cdot \nu}\,dx,$$

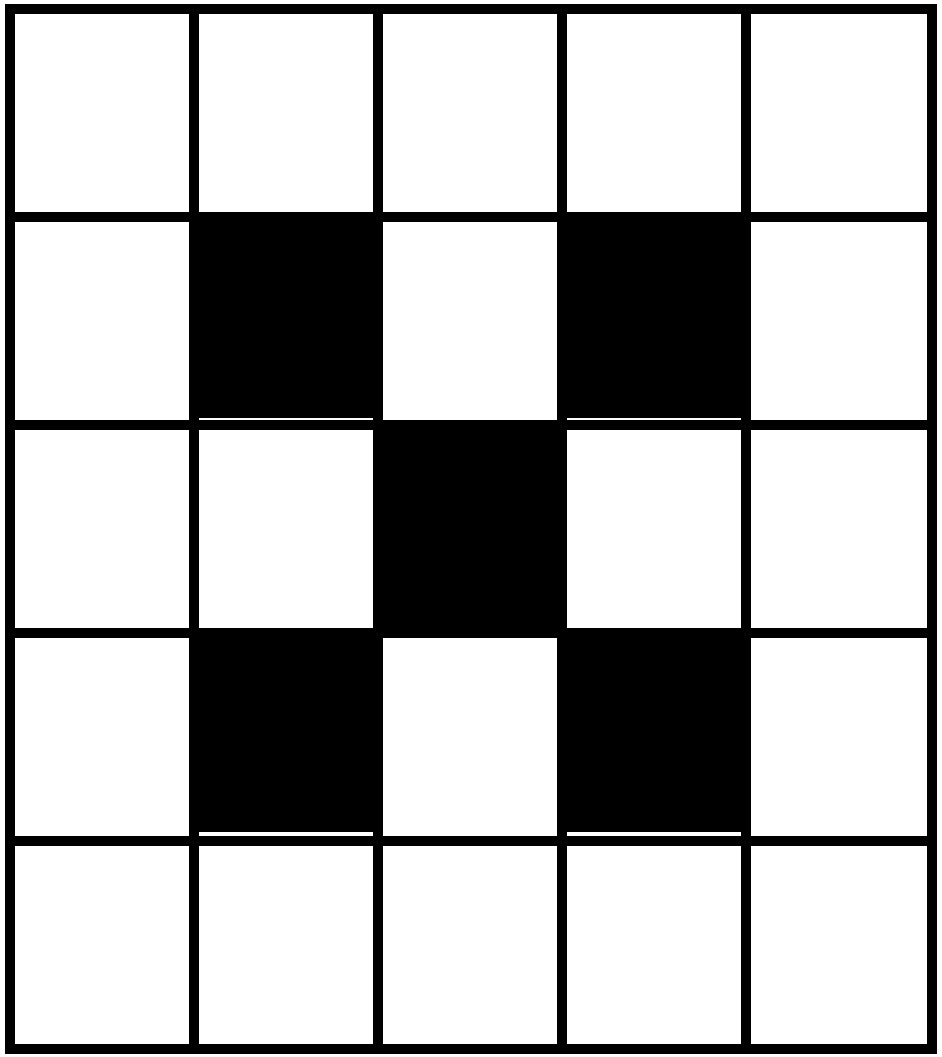
$$G(\nu) = \int_{\mathbb{R}^n} g(x) e^{-2\pi i x \cdot \nu}\,dx,$$

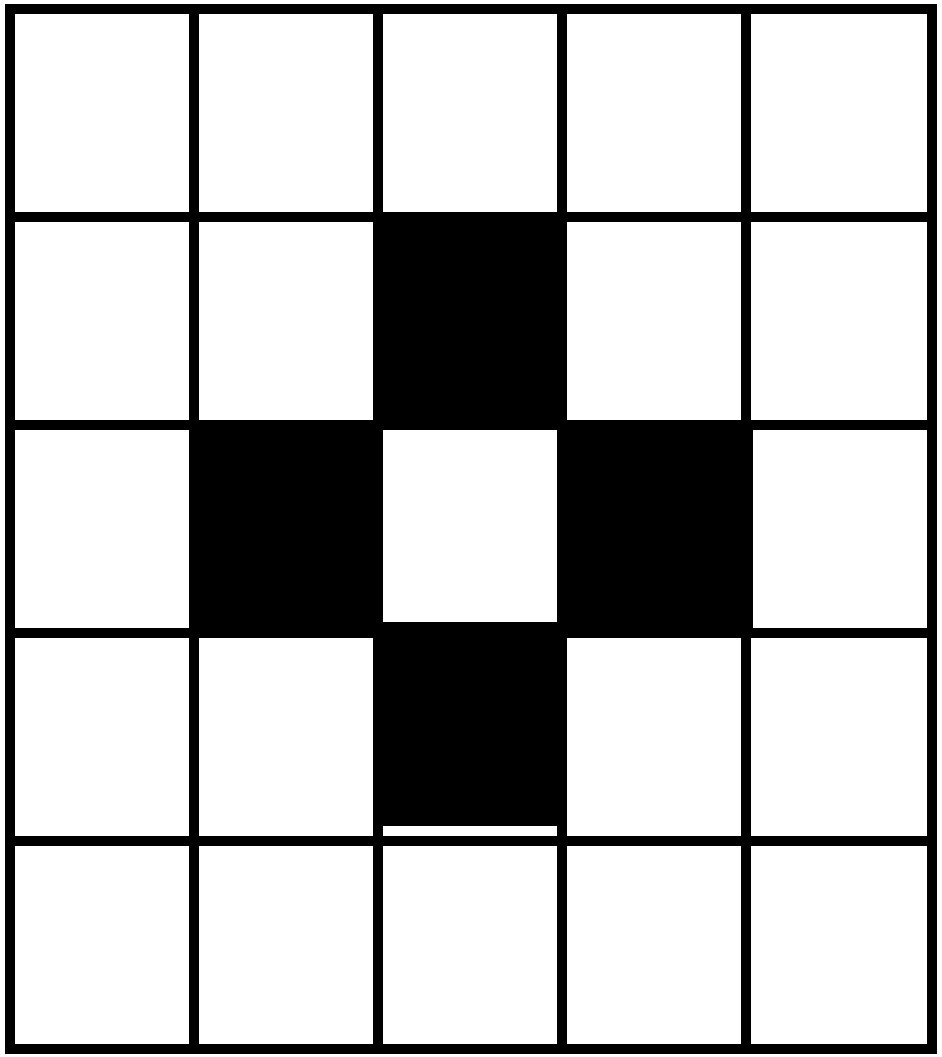
$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

$\mathcal{F}$ fourier transform

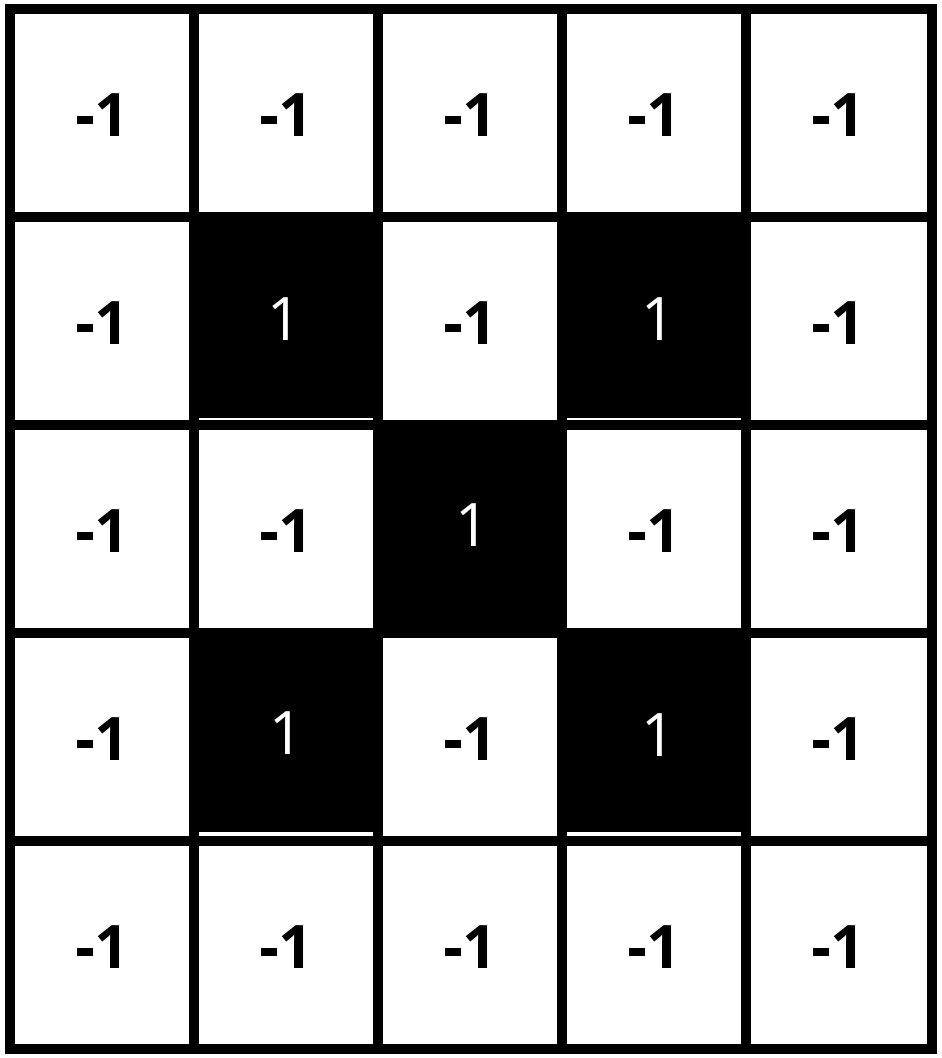
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

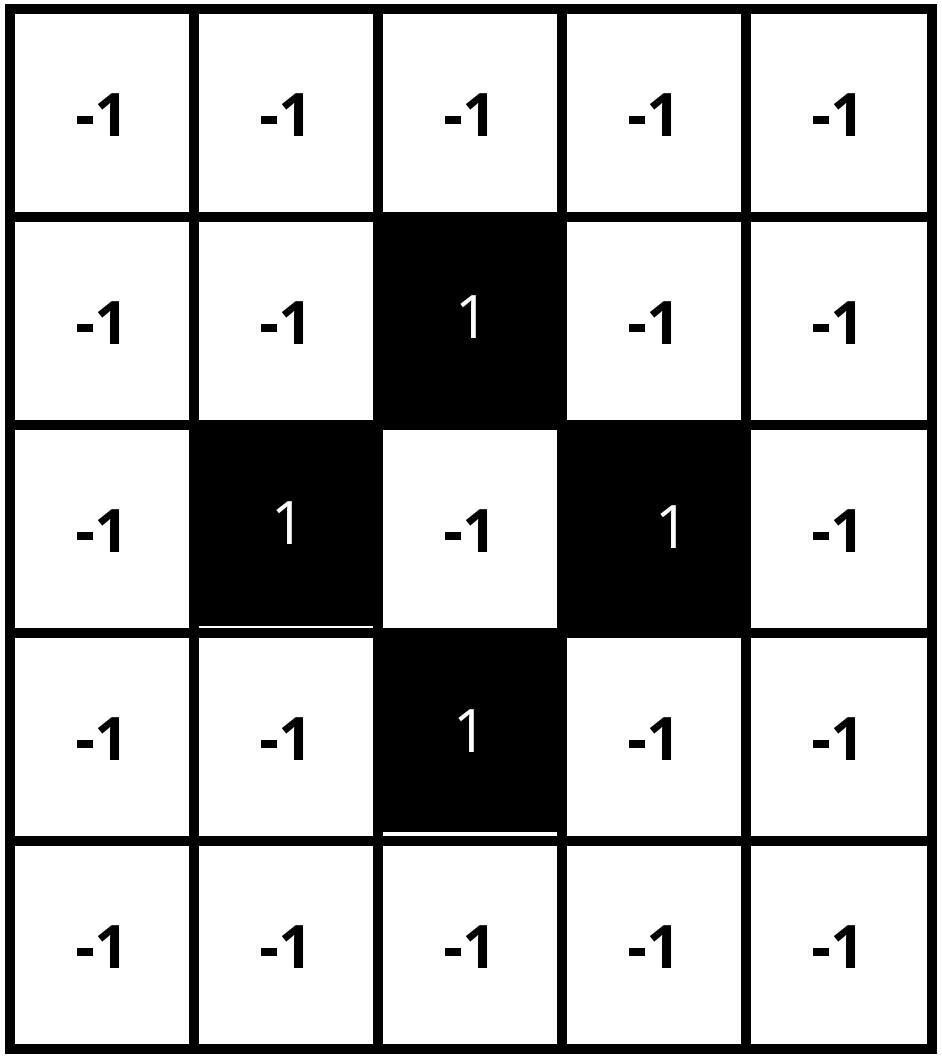
two images.



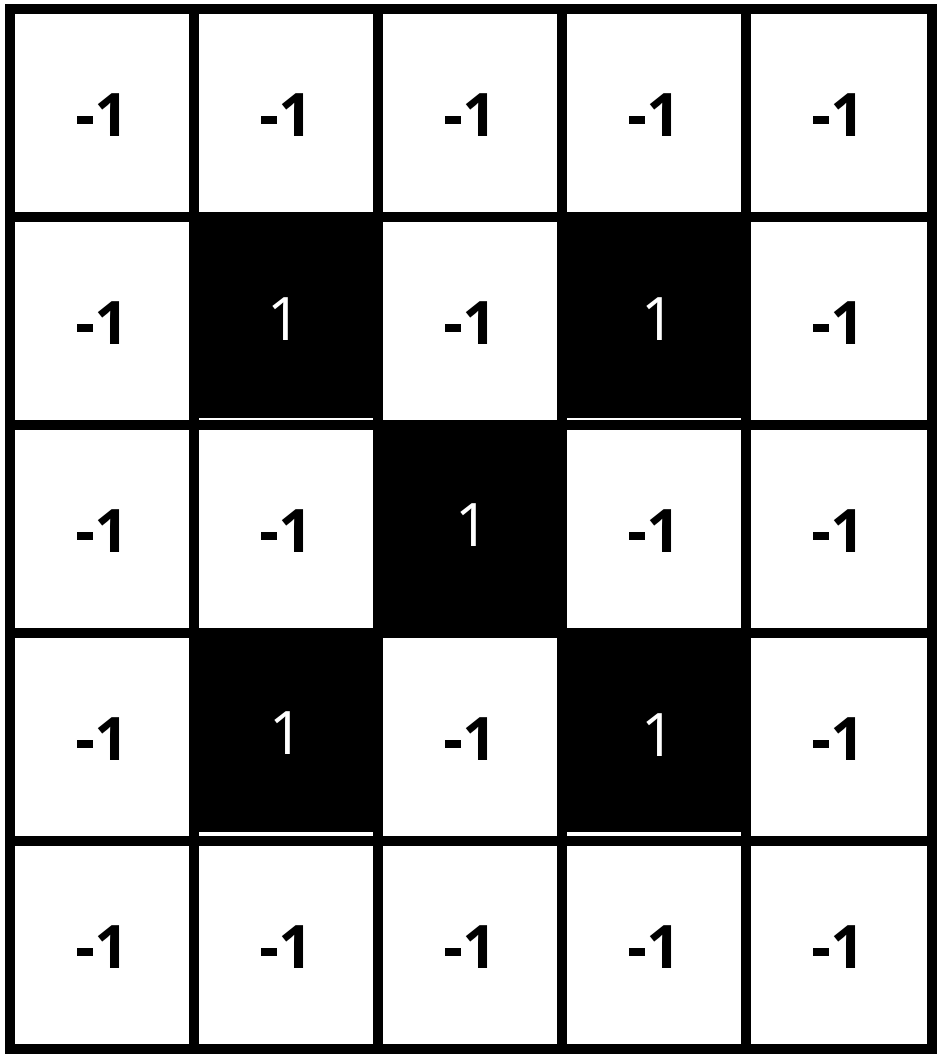


```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

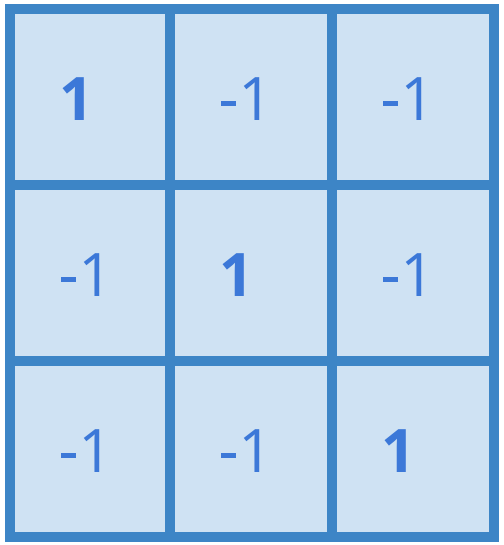


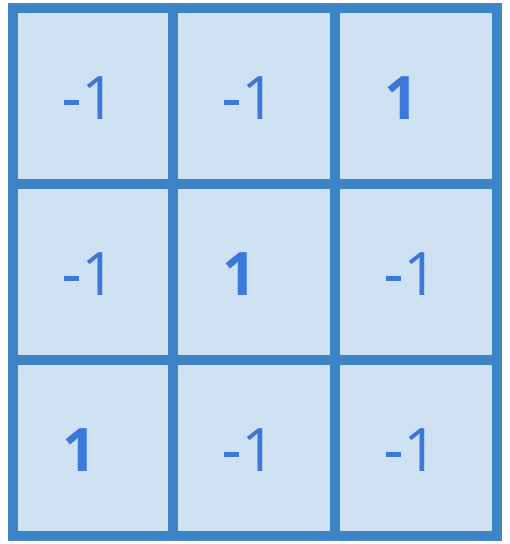


```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

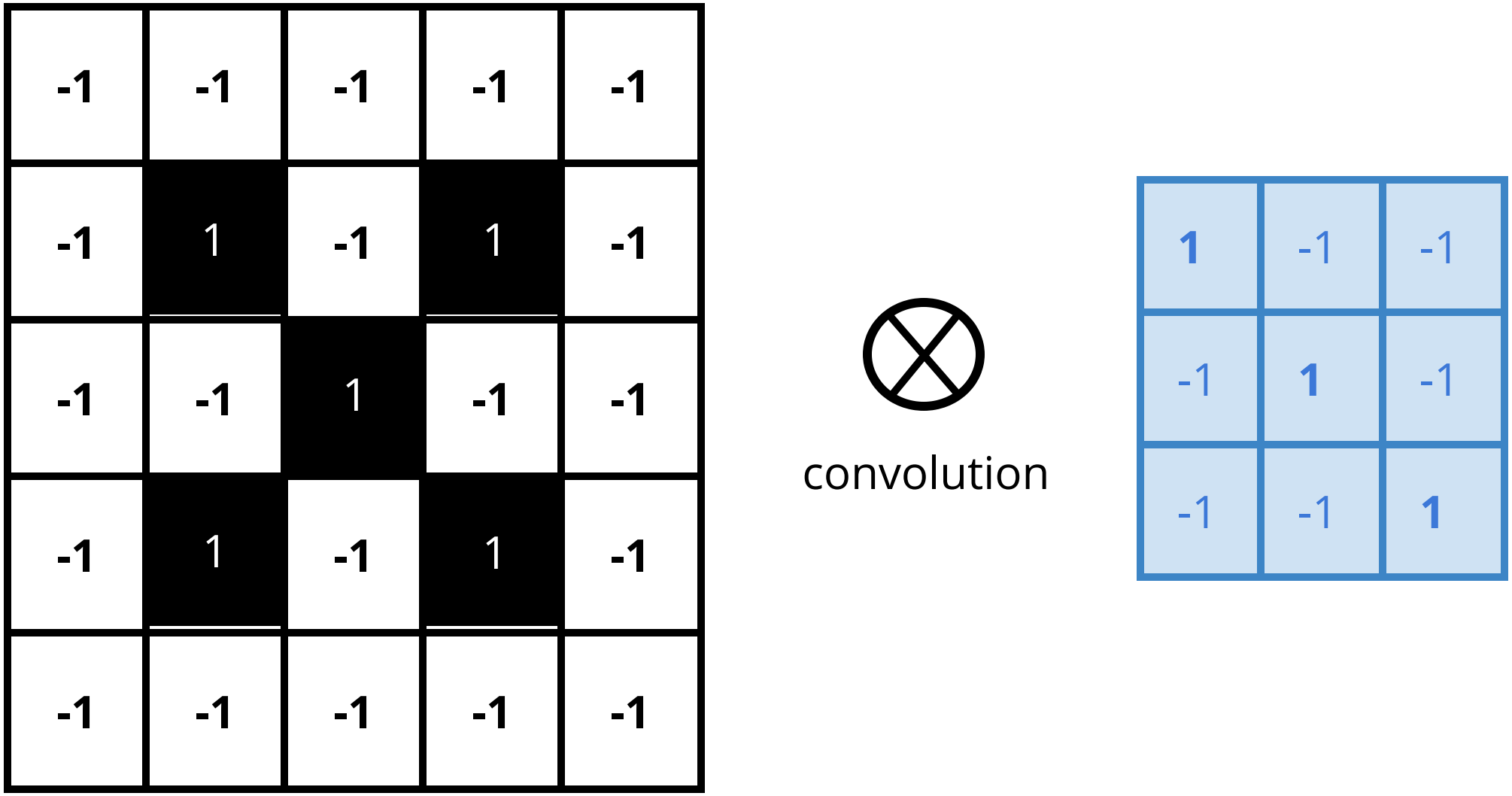


feature maps



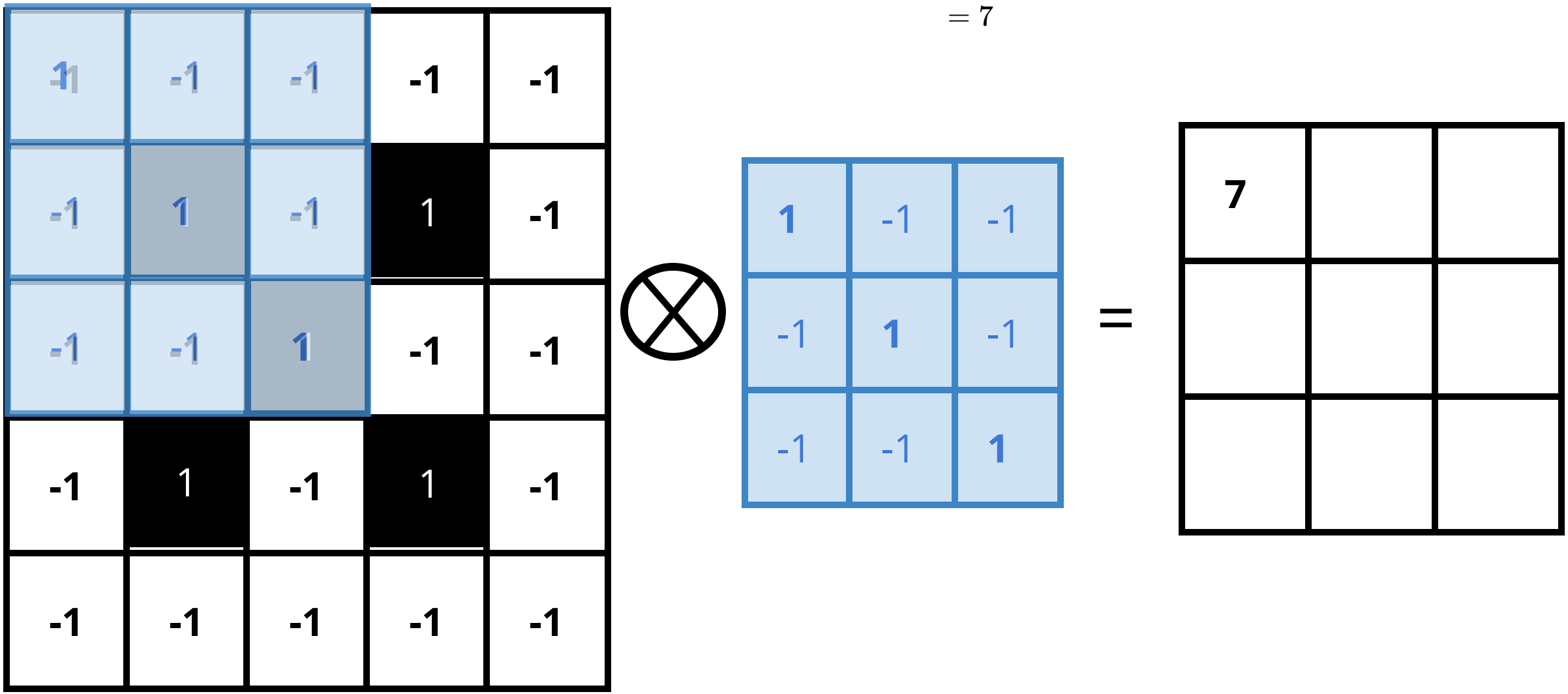


```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



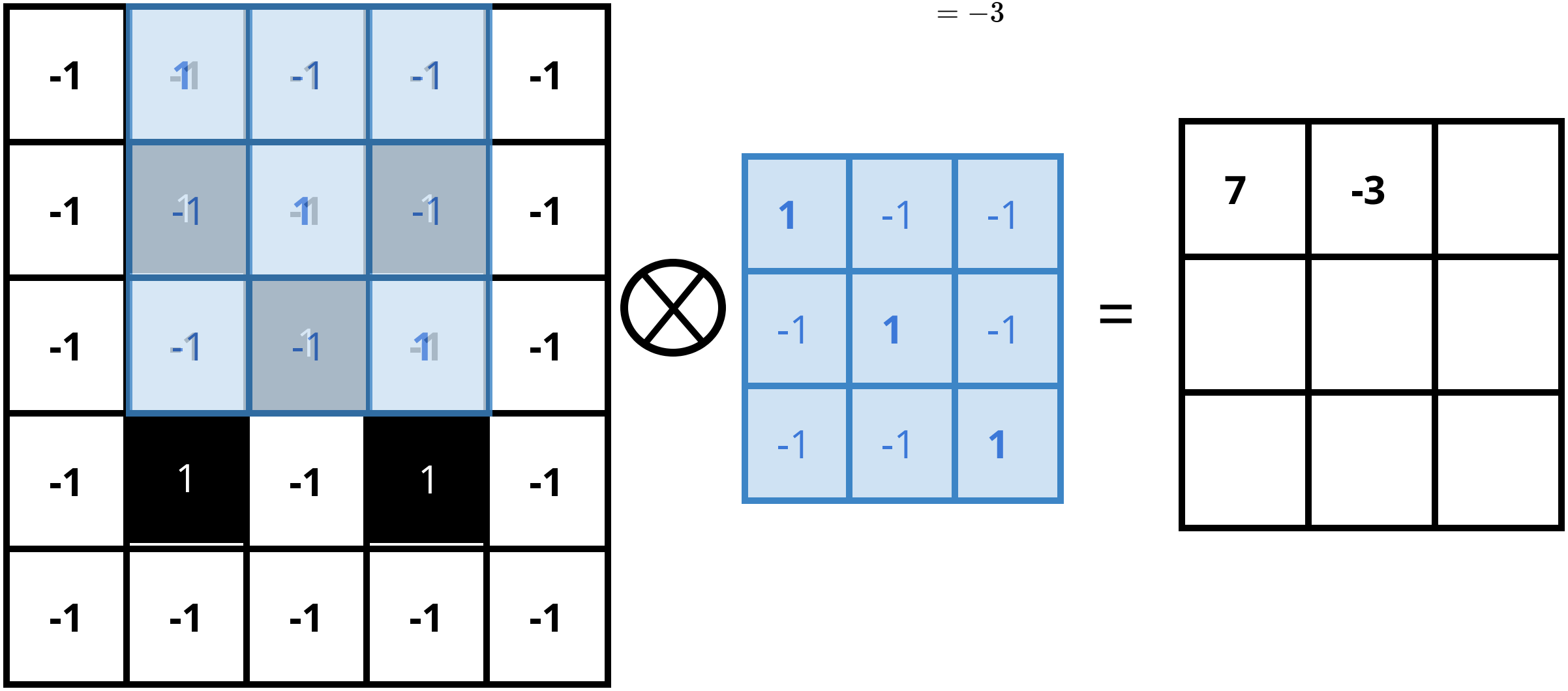
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

$$
\begin{gathered}
(-1 * 1) + (-1 * -1) + (-1 * -1) + \\
(-1 * -1) + (1 * 1) + (-1 * -1) \\
(-1 * -1) + (-1 * -1) + (1 * 1) \\
= 7
\end{gathered}
$$

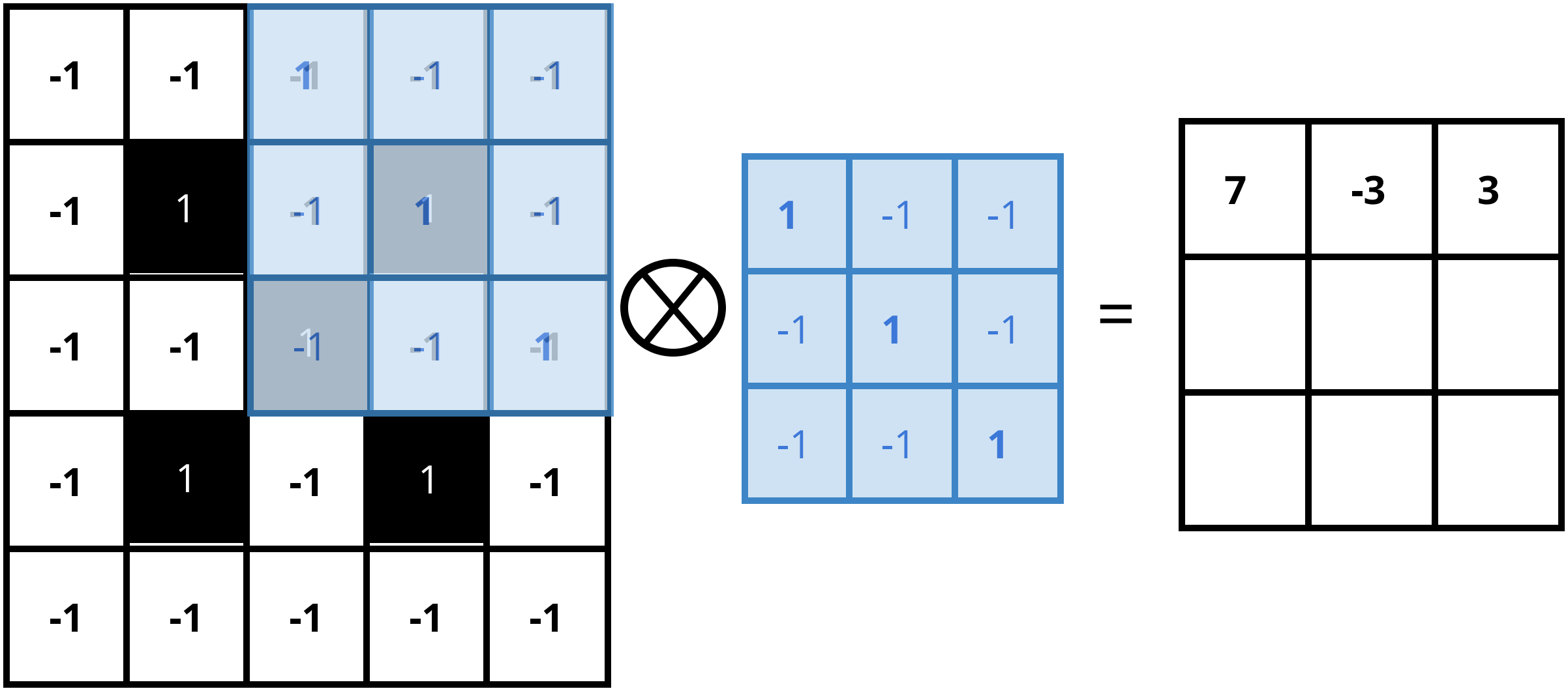


```
1 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

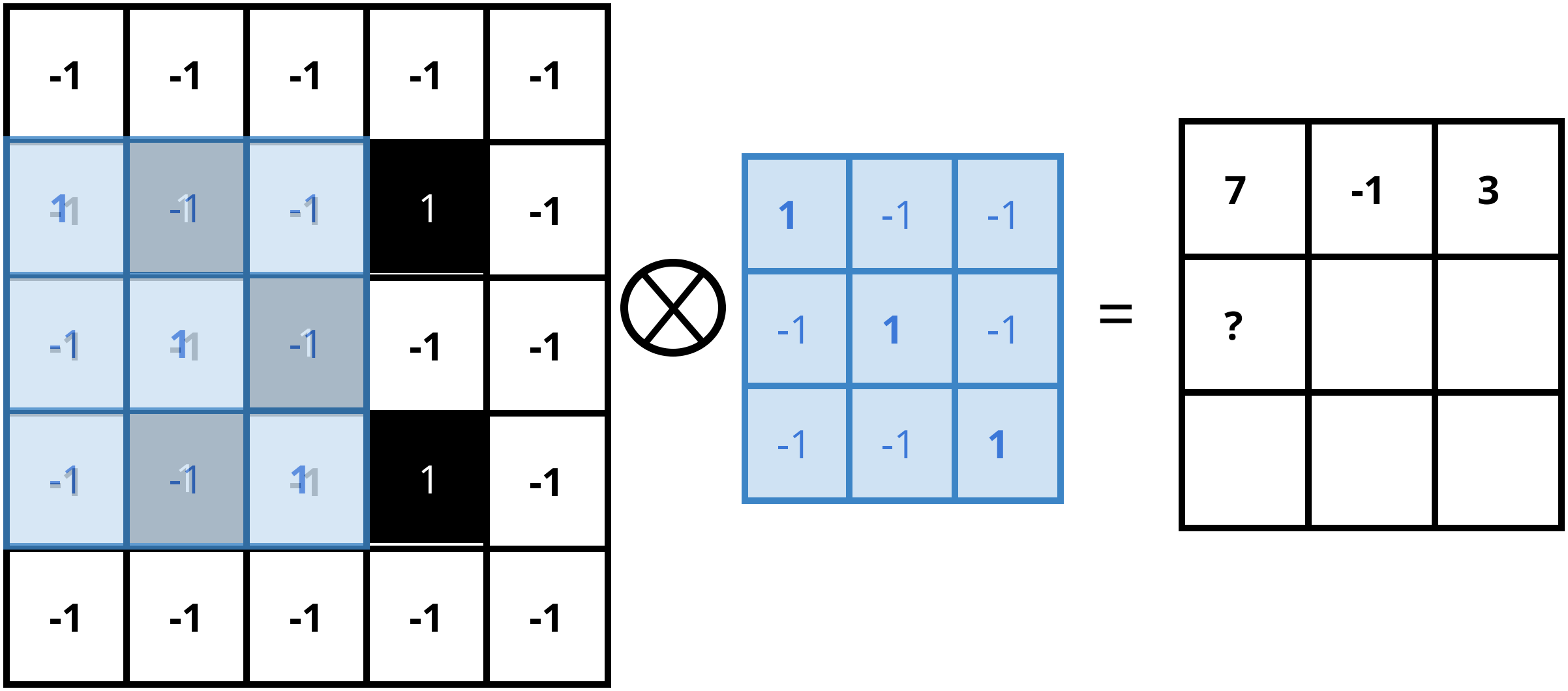
$$
\begin{gathered}
(-1 * 1) + (-1 * -1) + (-1 * -1) + \\
(-1 * 1) + (-1 * 1) + (-1 * 1) \\
(-1 * -1) + (-1 * 1) + (-1 * 1) \\
= -3
\end{gathered}
$$



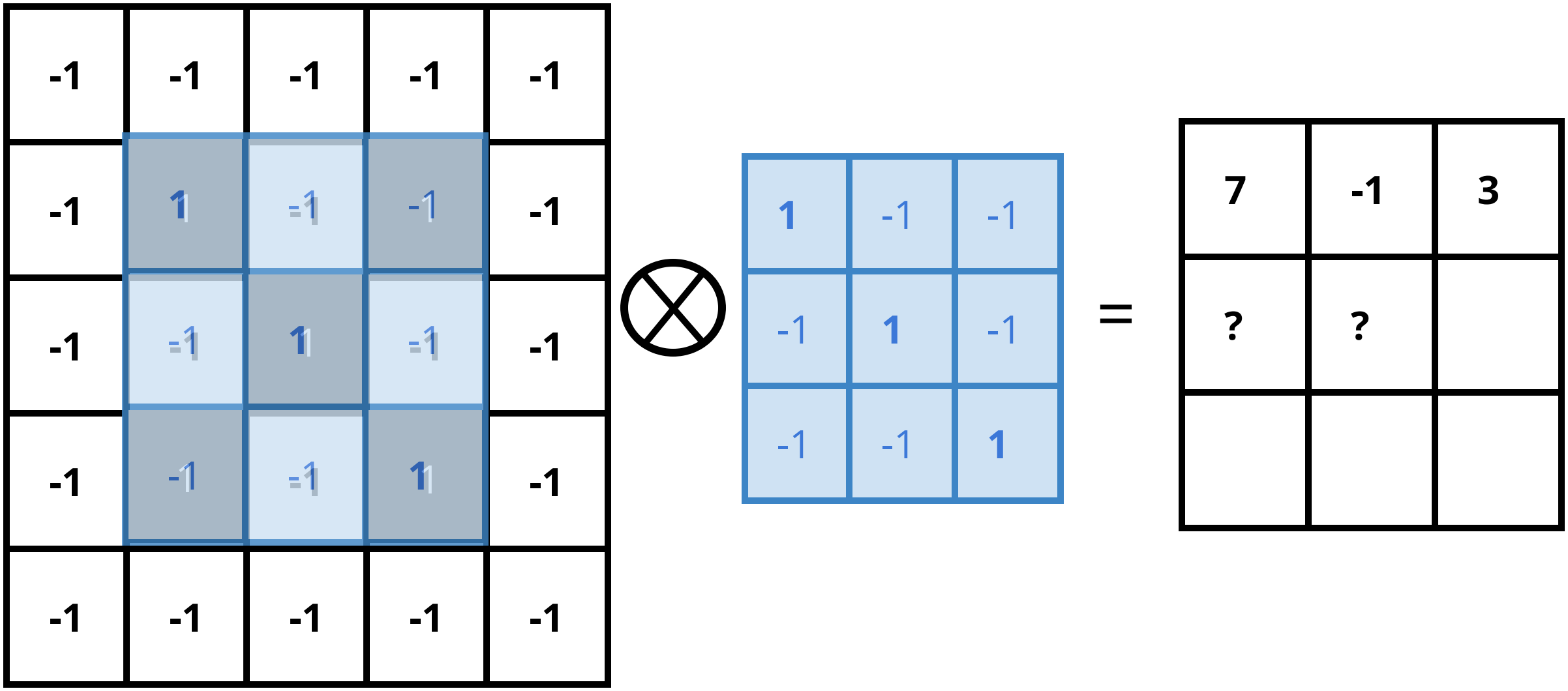
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



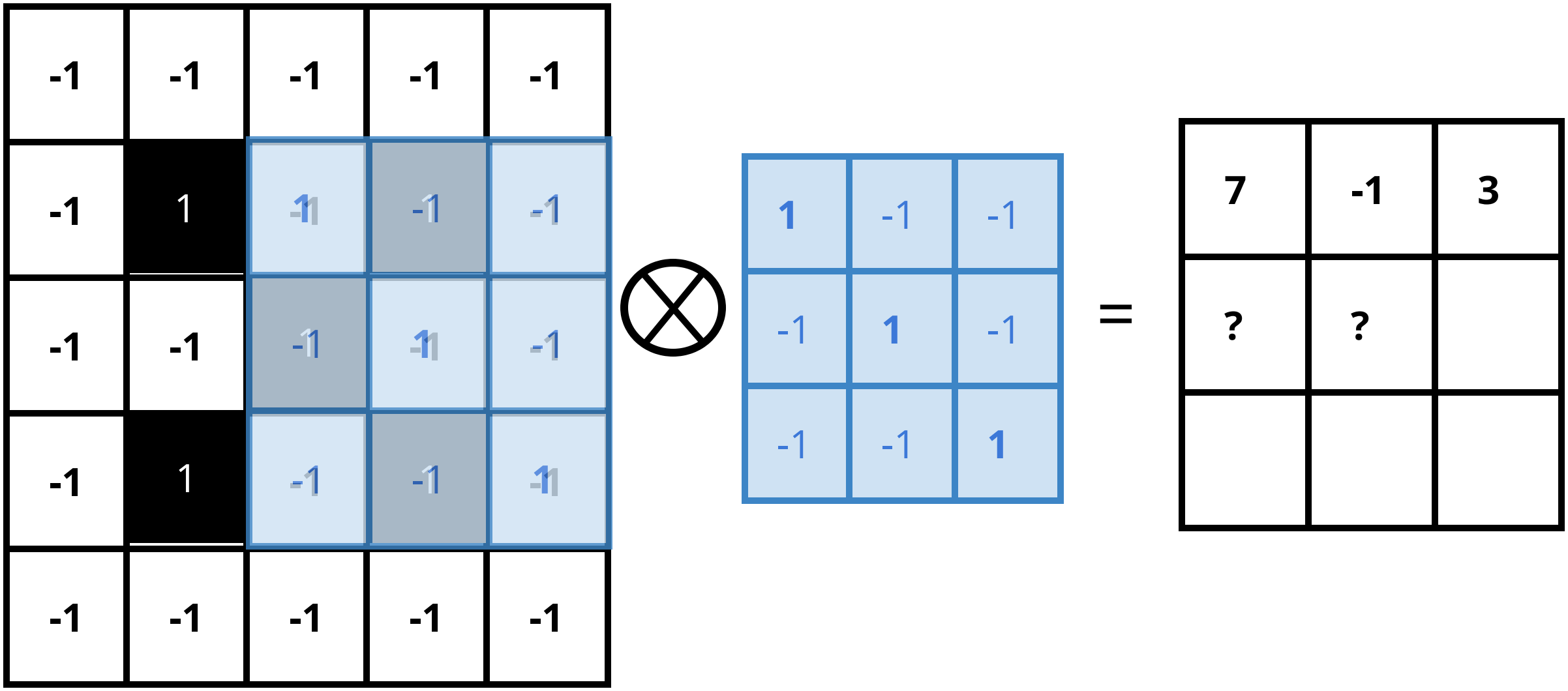
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



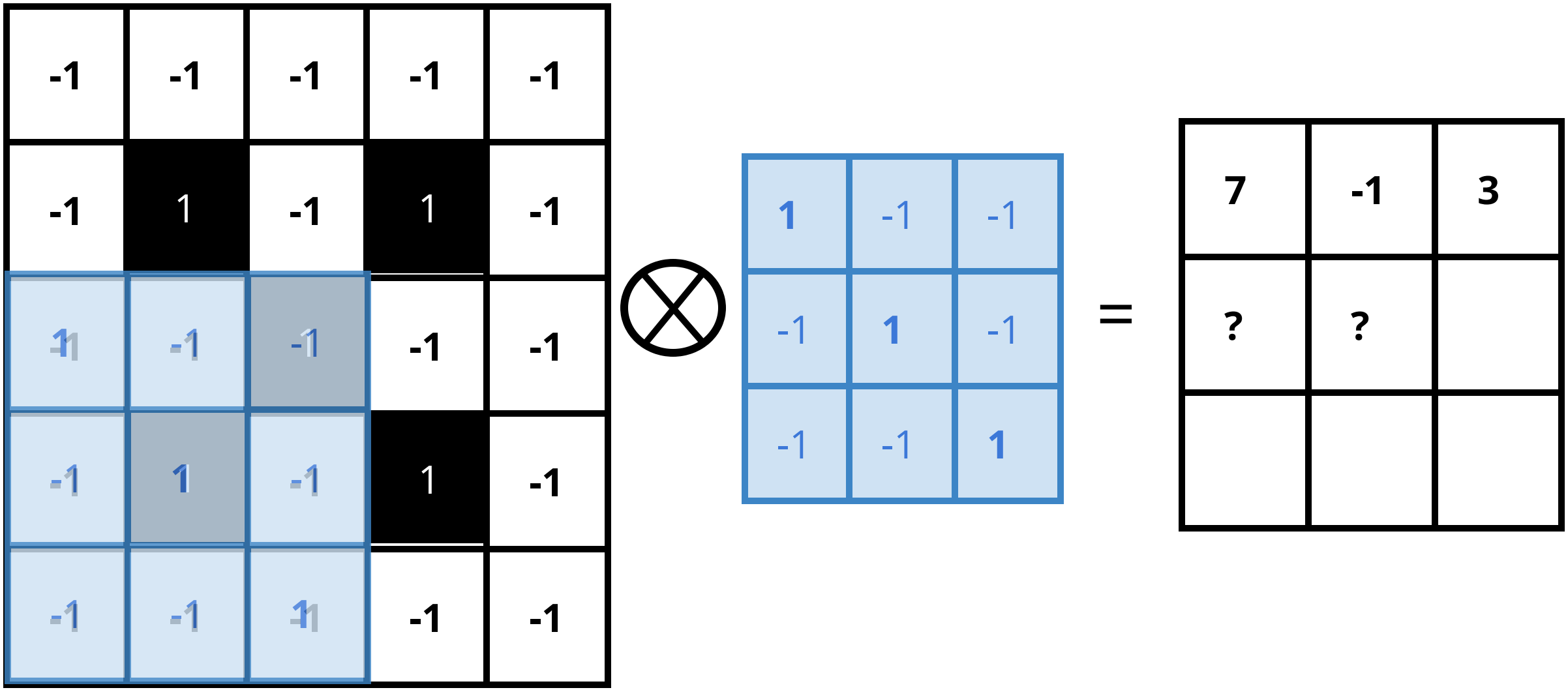
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



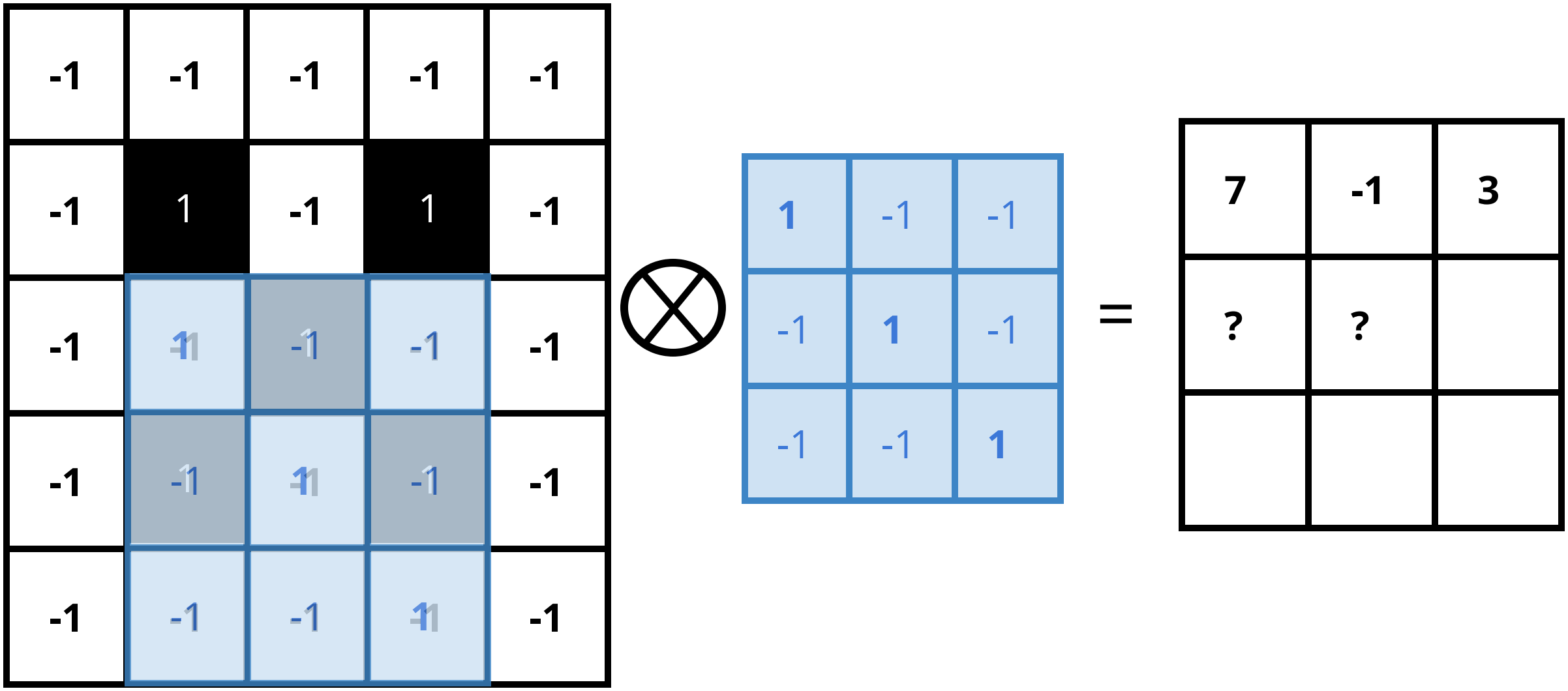
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



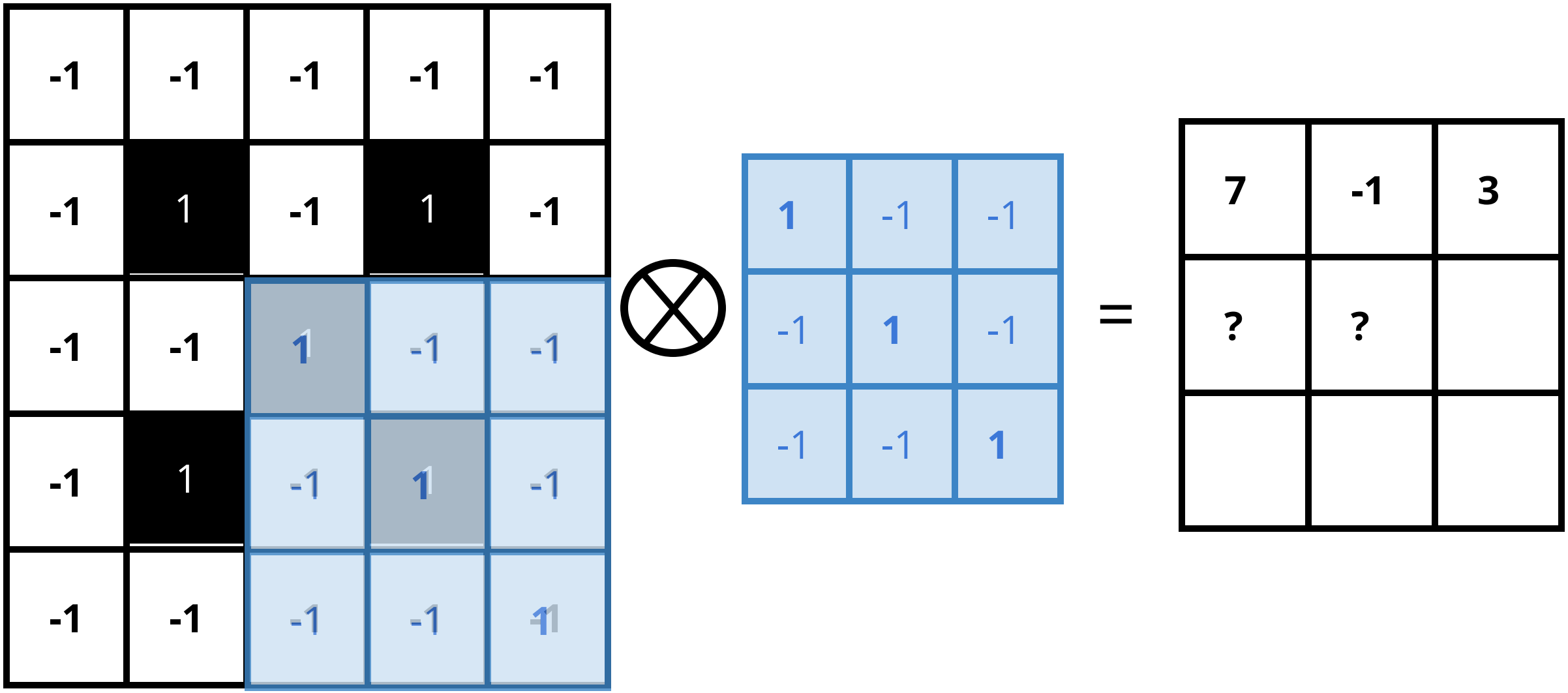
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



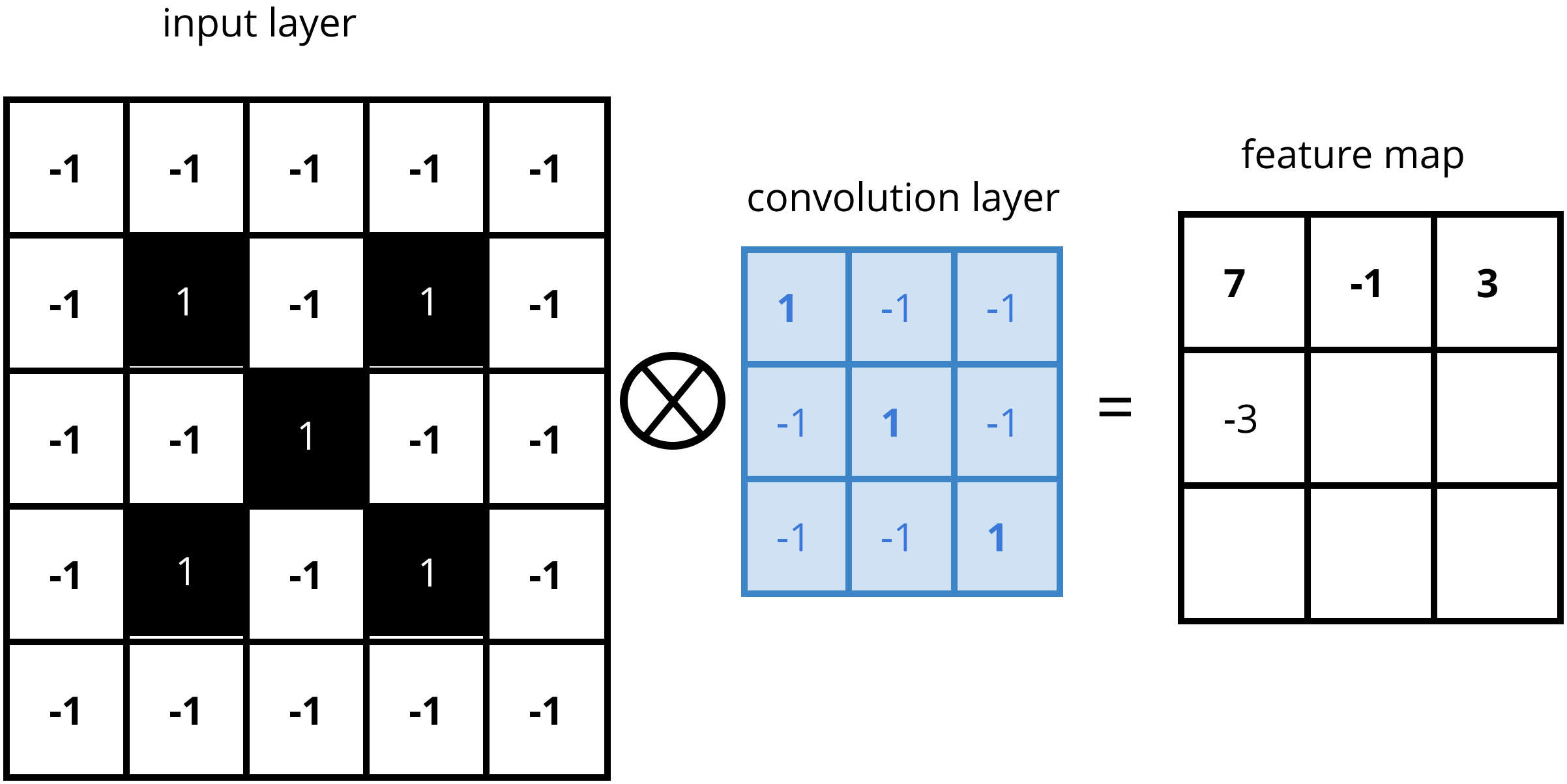
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



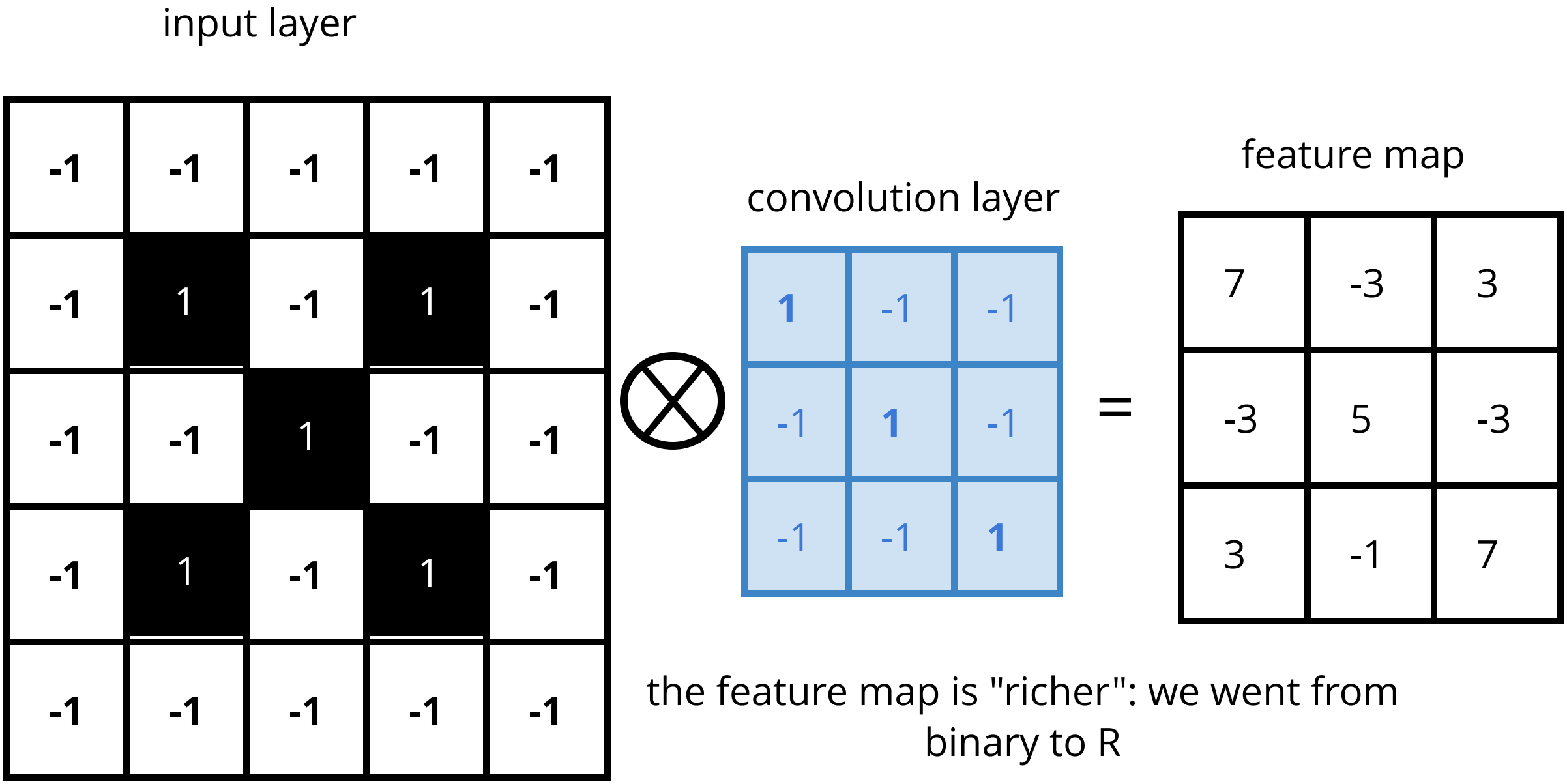
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```



```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

input layer

-1	-1	-1	-1	-1
-1	1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	-1	-1	-1	-1

⊗

convolution layer

1	-1	-1
-1	1	-1
-1	-1	1

=

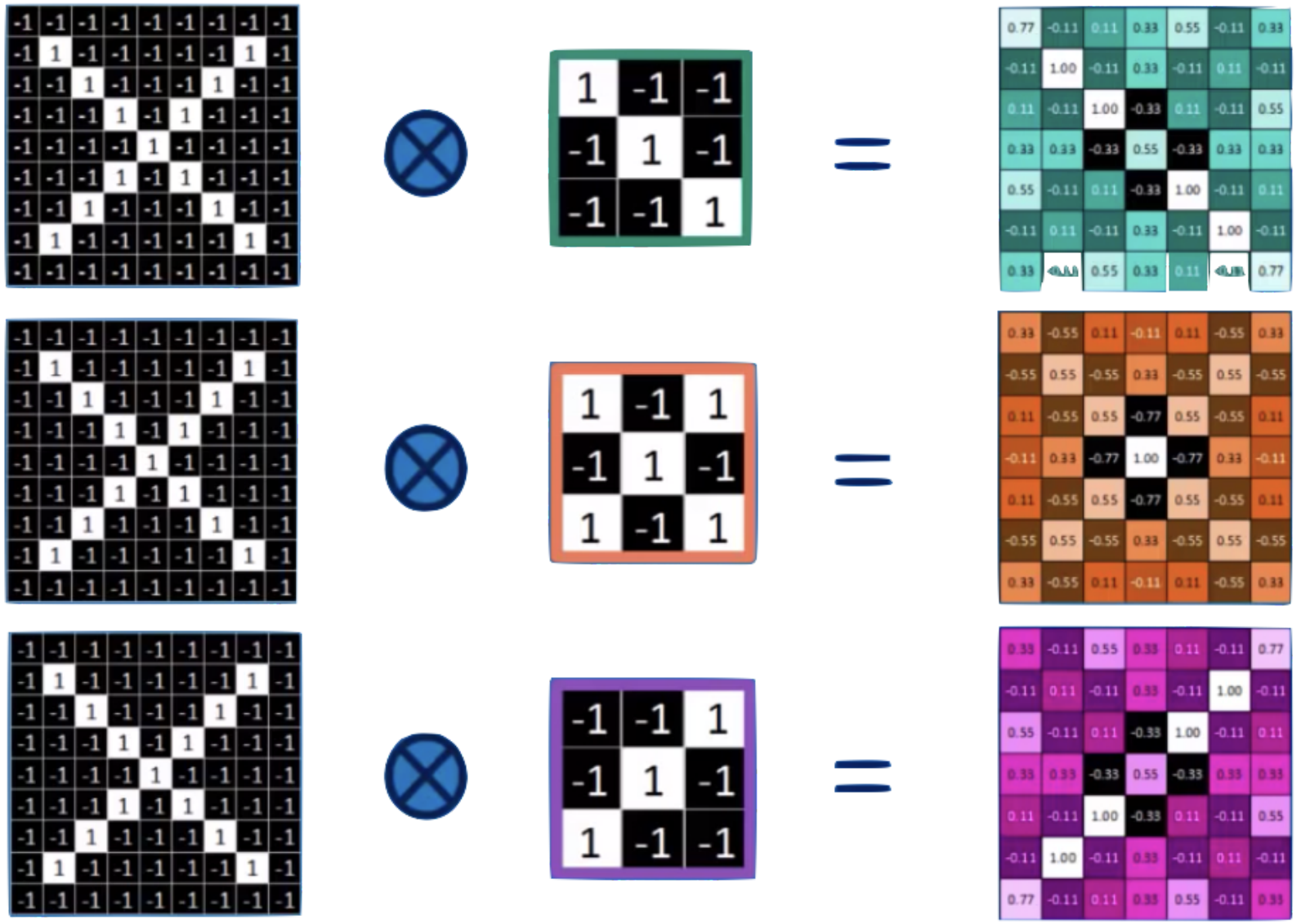
feature map

7	-3	3
-3	5	-3
3	-1	7

the feature map is "richer": we went from binary to R

and it is reminiscent of the original layer

Convolve with different feature: each neuron is 1 feature



ReLu

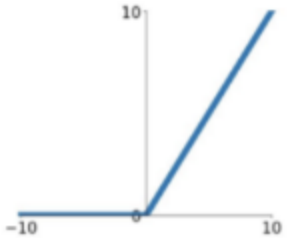
```
1 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

ReLu: normalization that replaces negative values with 0's

7	-3	3
-3	5	-3
3	-1	7

ReLU

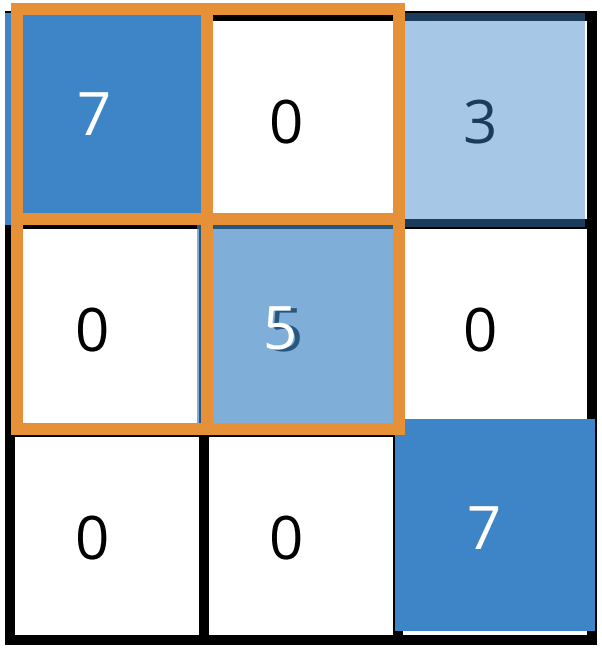
$$\max(0, x)$$

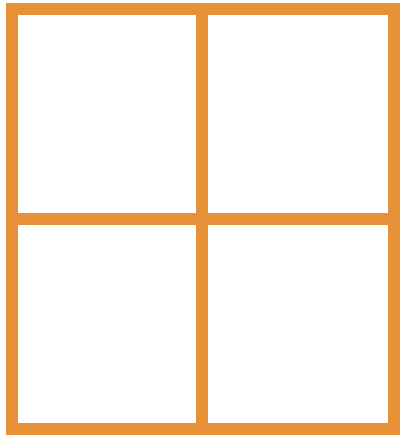


7	0	3
0	5	0
3	0	7

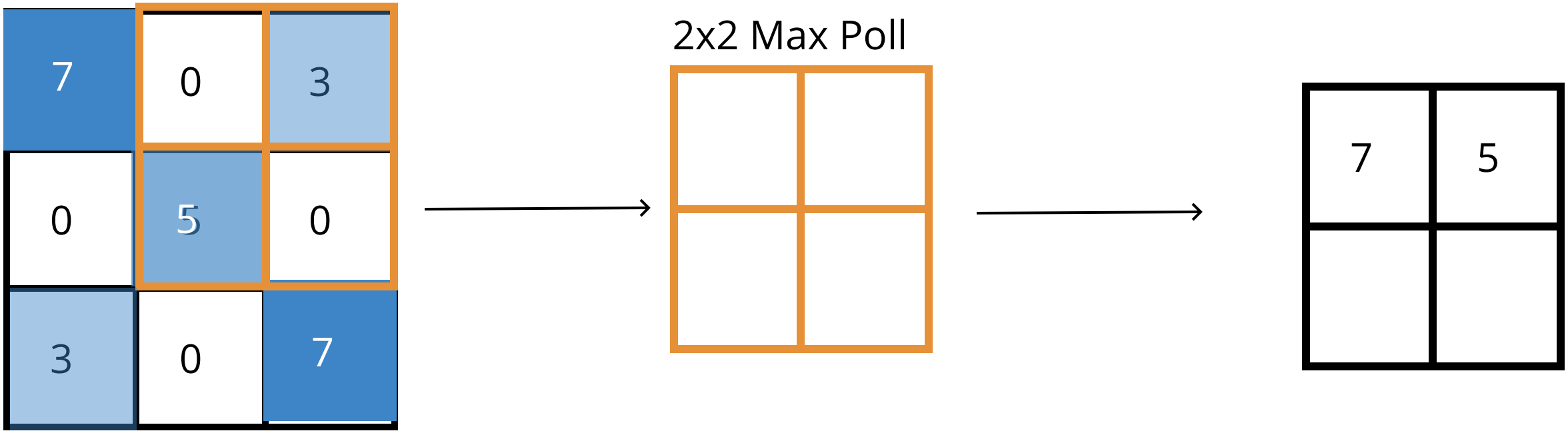
Max-Pool

MaxPooling: reduce image size, generalizes result

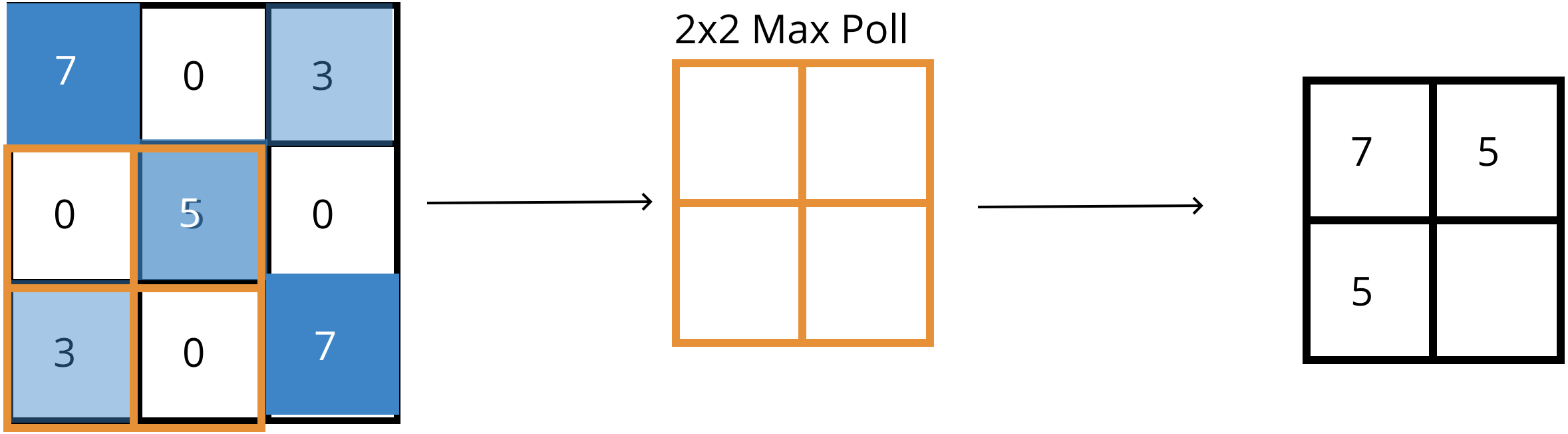




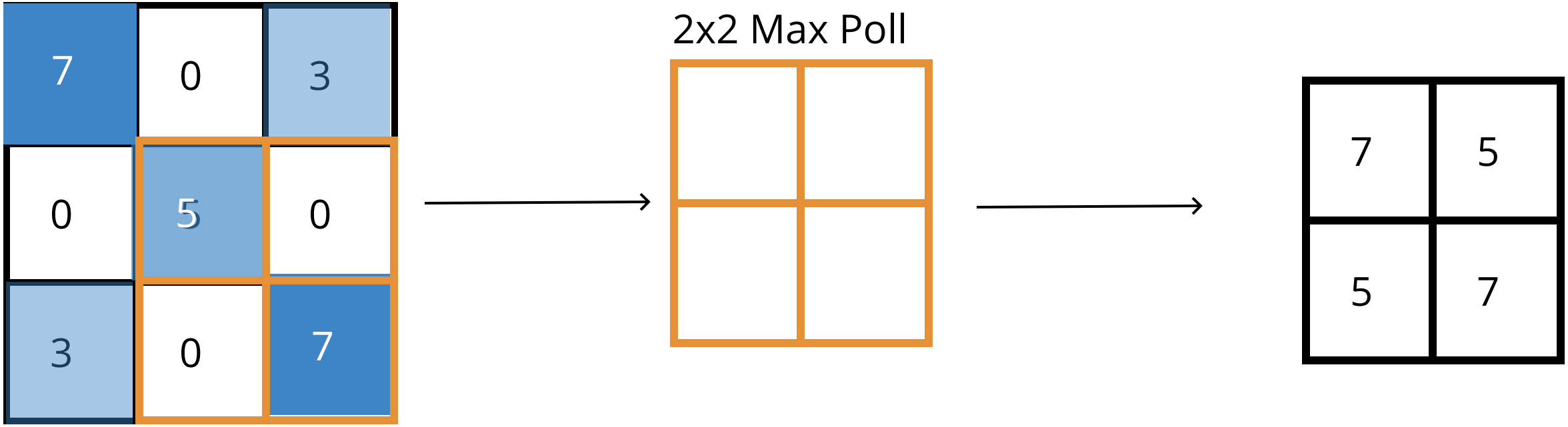
MaxPooling: reduce image size, generalizes result



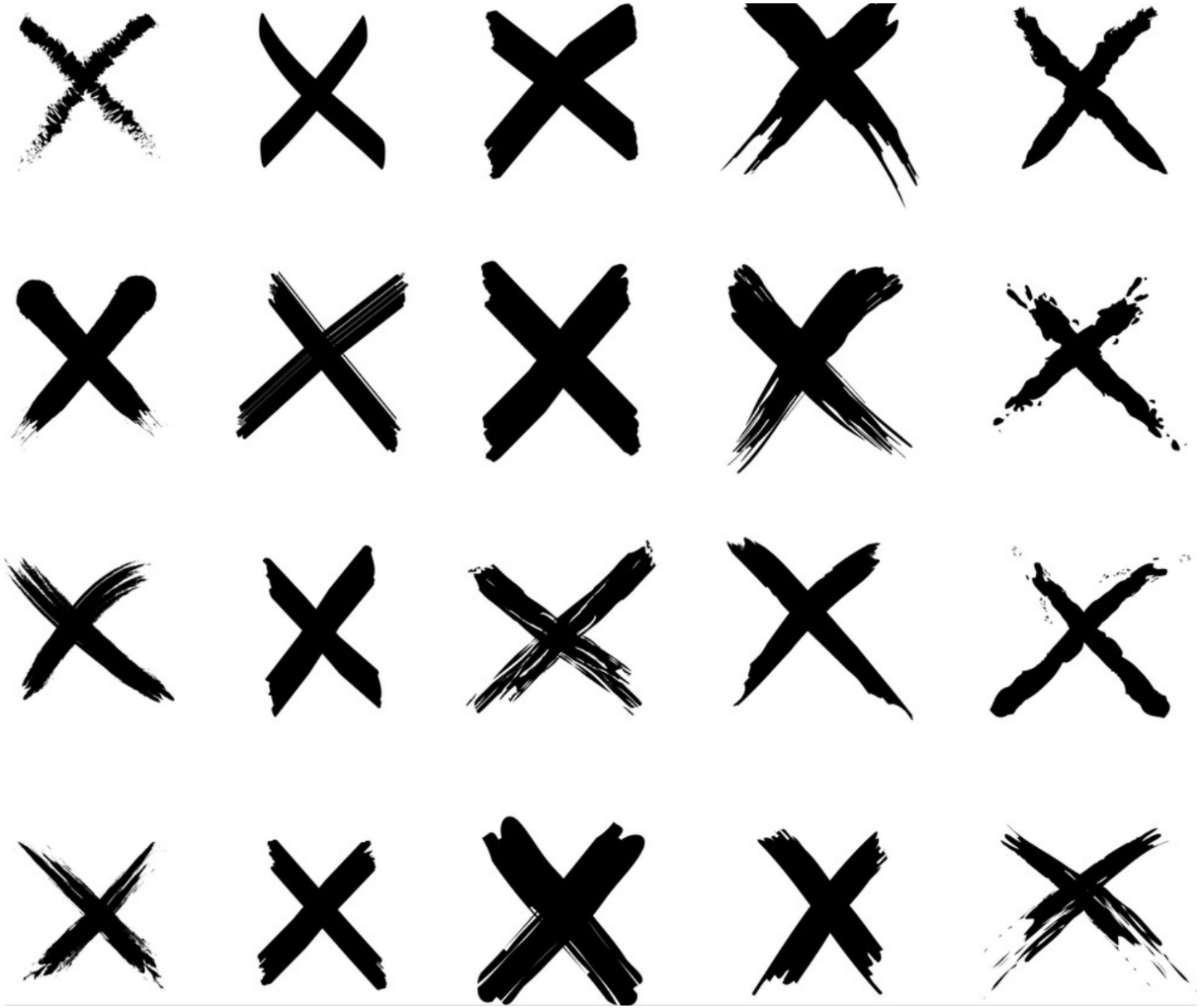
MaxPooling: reduce image size, generalizes result



MaxPooling: reduce image size, generalizes result



```
1 model.add(MaxPooling2D(pool_size=(2, 2)))
```

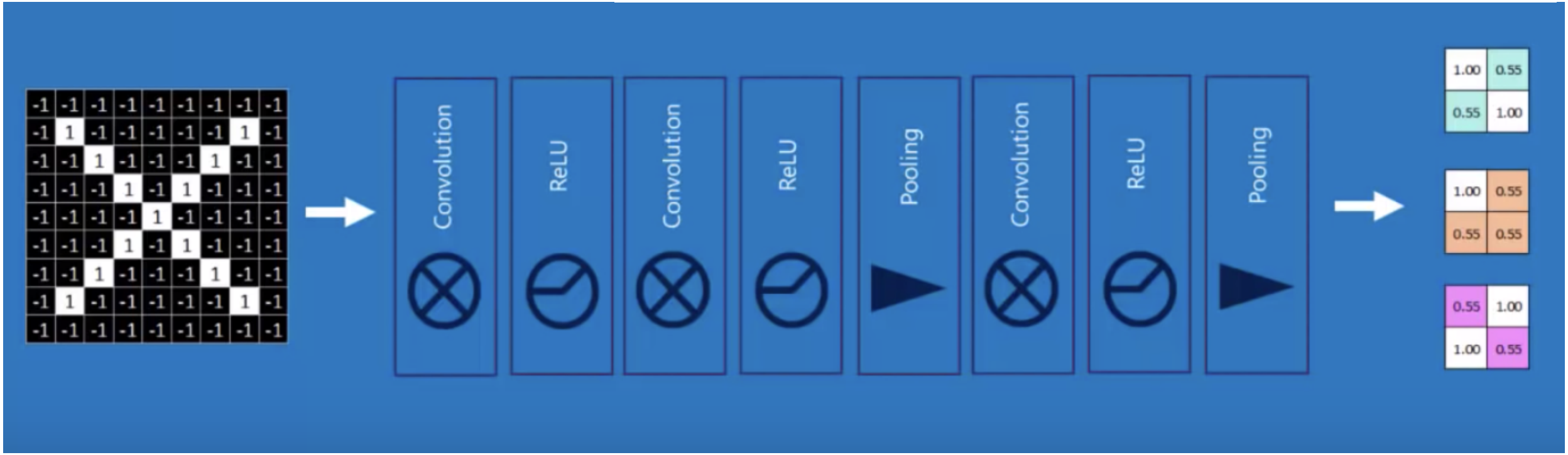


MaxPooling: reduce image size *& generalizes result*

By reducing the size and picking the maximum of a sub-region we make the network less sensitive to specific details

CNN

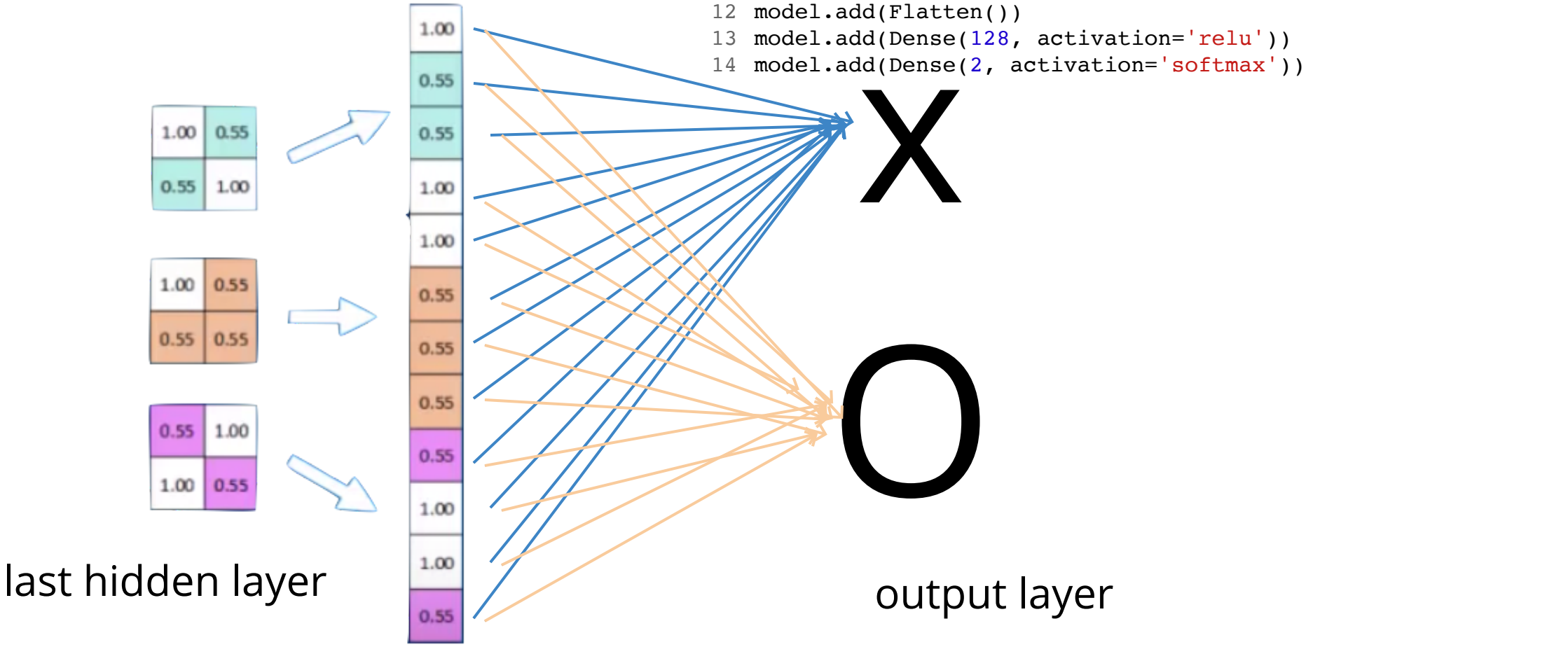
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
model = Sequencial()
model.add(Conv2D(32, kernel_size=(10, 10),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```



final layer:

the final layer is fully connected MPL

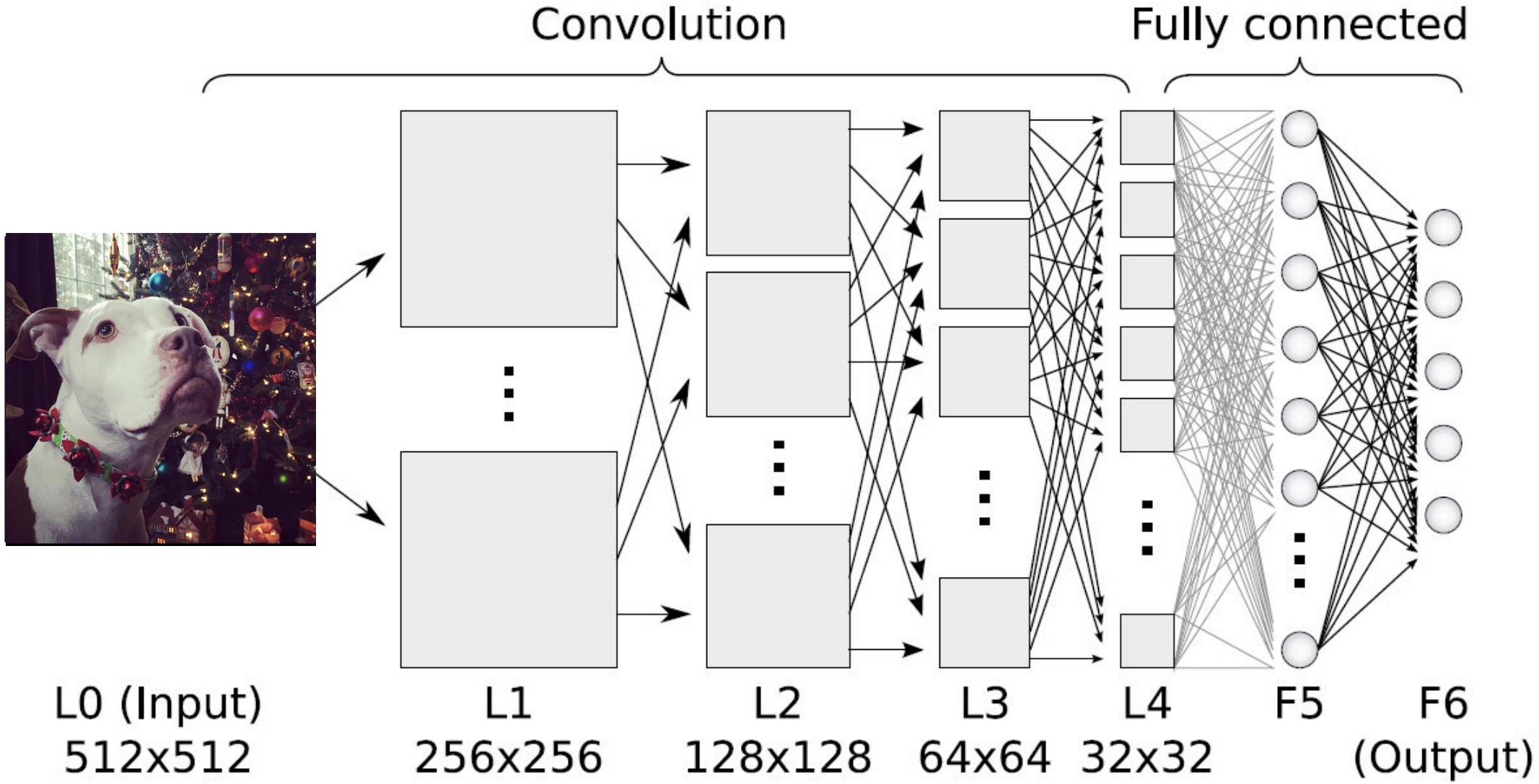
```
1  from keras.models import Sequential
2  from keras.layers import Dense, Dropout, Flatten
3  from keras.layers import Conv2D, MaxPooling2D
4  model = Sequencial()
5  model.add(Conv2D(32, kernel_size=(10, 10),
6                   activation='relu',
7                   input_shape=input_shape))
8  model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
9  model.add(MaxPooling2D(pool_size=(2, 2)))
10 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
11 model.add(MaxPooling2D(pool_size=(2, 2)))
12 model.add(Flatten())
13 model.add(Dense(128, activation='relu'))
14 model.add(Dense(2, activation='softmax'))
```



last hidden layer

output layer

Stack multiple convolution layers



6/6

Generative AI in society

Faces95 - Computer Vision Science Research Projects, Dr Libor Spacek.

2) Faces95: This dataset consists of 72 (male and female) individuals with 20 images per individual. The background of images consists of a red curtain and also has the background variation that caused by shadows. There are some expression and lighting variation.

Examples Faces95 images

White Males

White Female

Non-White Males

Non-White Female

Faces95 - Computer Vision Science Research Projects, Dr Libor Spacek.

2) Faces95: This dataset consists of 72 (male and female) individuals with 20 images per individual. The background of images consists of a red curtain and also has the background variation that caused by shadows. There are some expression and lighting variation.

Examples Faces95 images

White Males: 43

White Female: 6

Non-White Males: 12

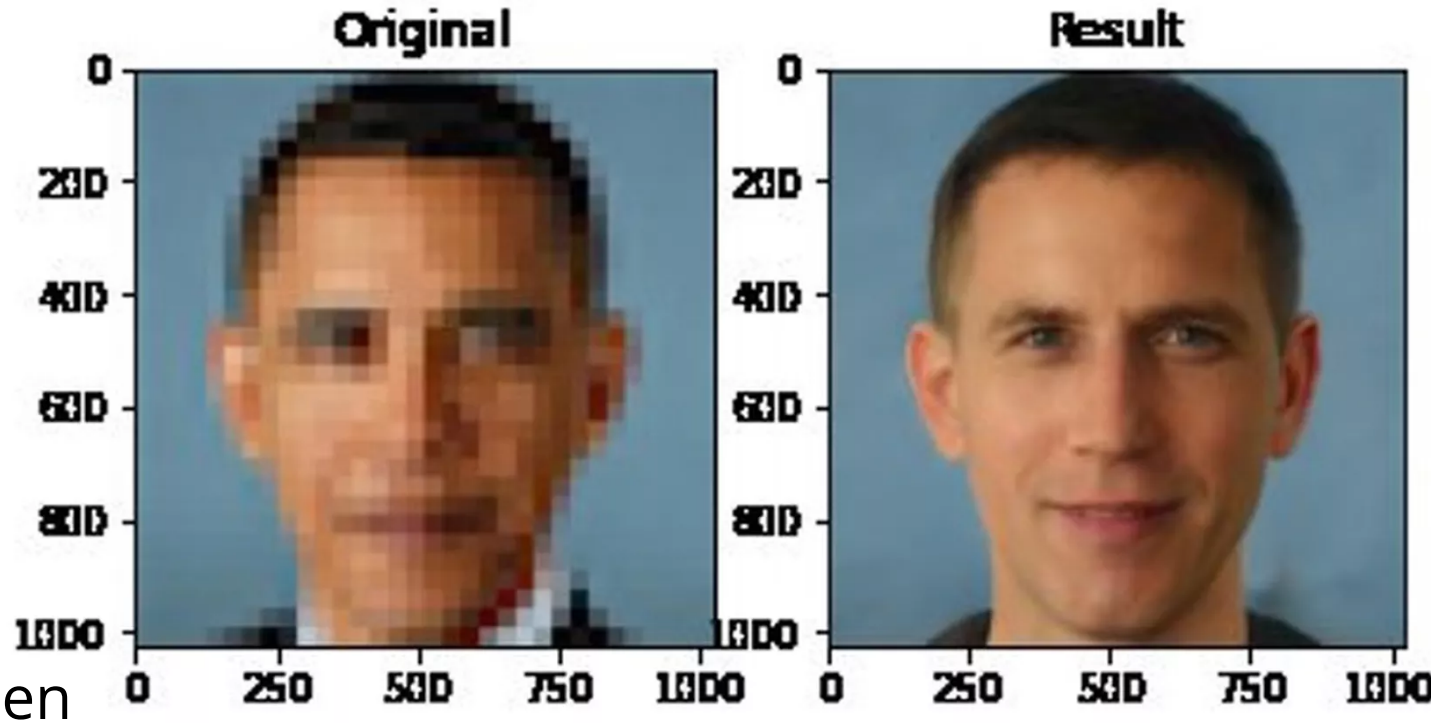
Non-White Female: 1

models are neutral, the bias is in the data (or is it?)

Why does this AI model whitens Obama face?

Simple answer: the data is biased. The algorithm is fed more images of white people

But really, would the opposite have been acceptable? *The bias is in society*



https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias

models are neutral, the bias is in the data (or is it?)

Why does this AI model whitens Obama face?

Simple answer: the data is biased. The algorithm is fed more images of white people

https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias

~~models are neutral,~~

~~the bias is in the data~~

The bias is in the data

The bias is in the models and the decision we make

The bias is in how we choose to optimize our model

The bias is society that provides the framework to validate our biased models

Should AI reflect

who we are

(and *enforce and grow our bias*)

or should it reflect who we aspire to be?

(*and who decides what that is?*)

none of this is new

~~models are neutral,~~

~~the bias is in the data~~

The bias is in the data

The bias is in the models and the decision we make

The bias is in how we choose to optimize our model

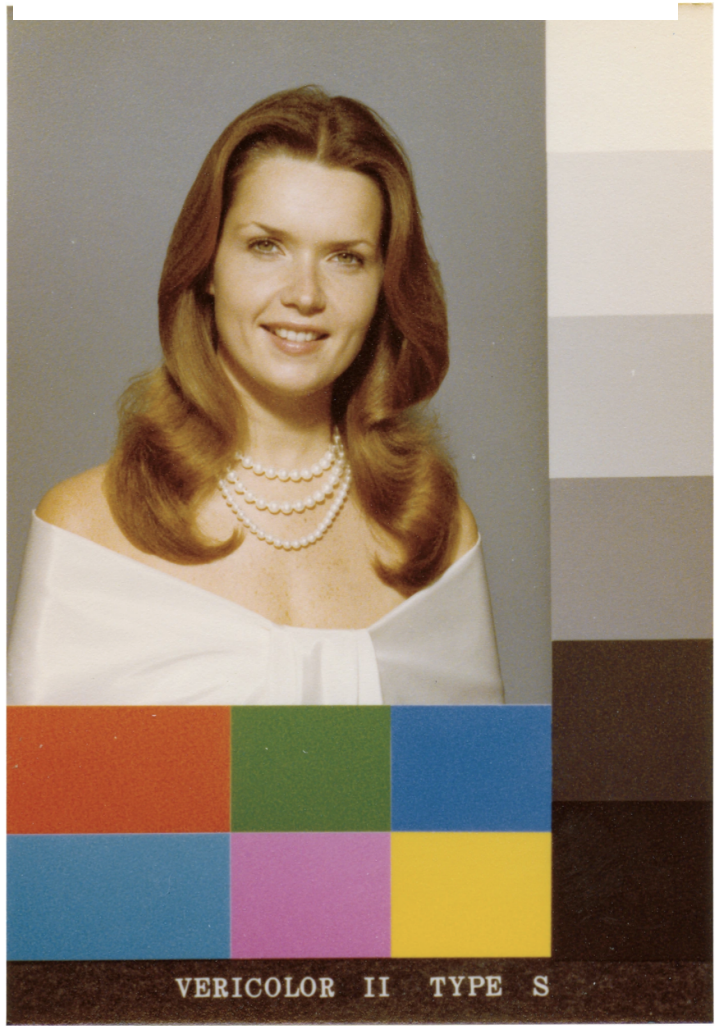
The bias is society that provides the framework to validate our biased models

Lens

LENS

The Racial Bias Built Into Photography

Sarah Lewis explores the relationship between racism and the camera.

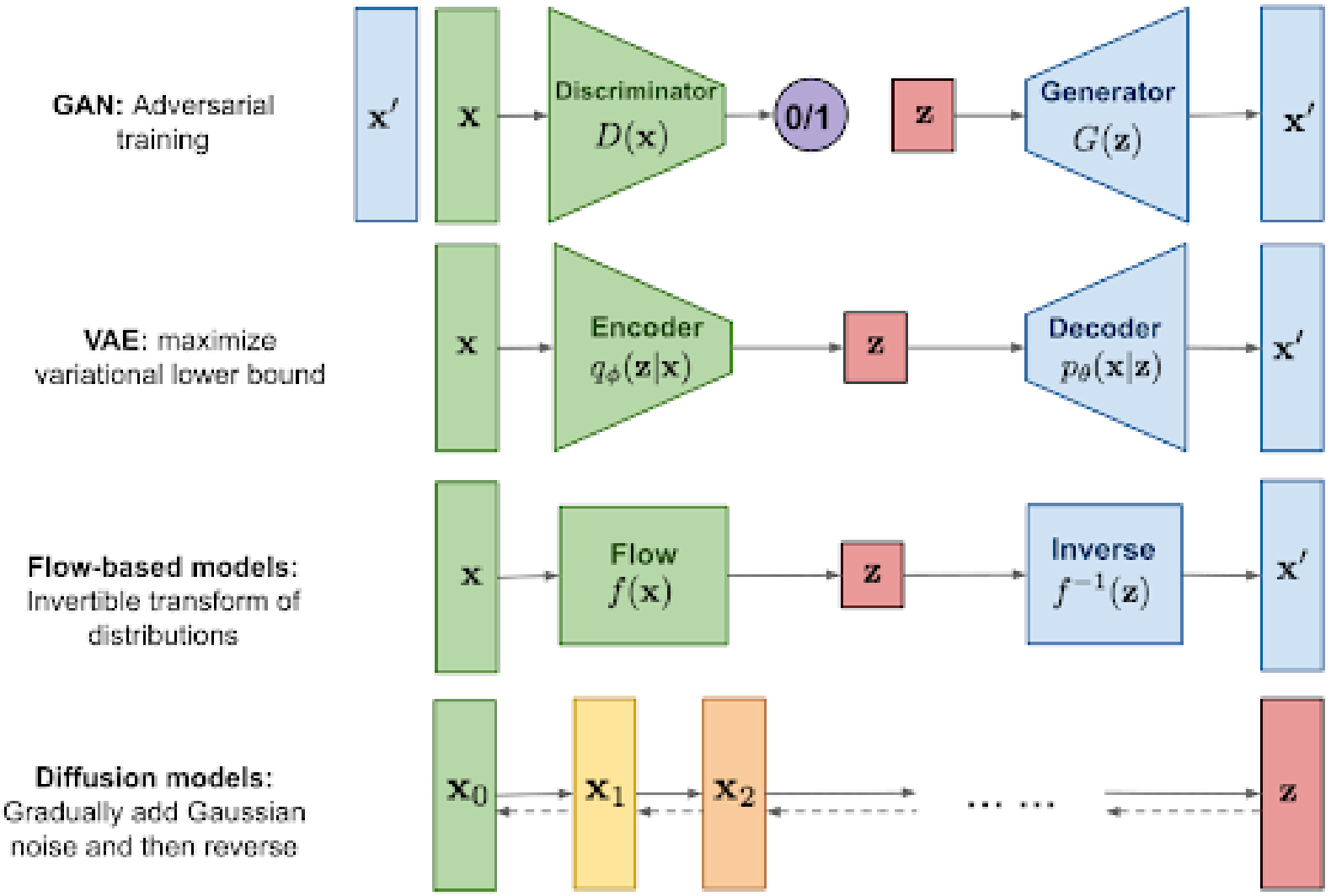


Shirley Card, 1978. Courtesy of Hermann Zschiegner

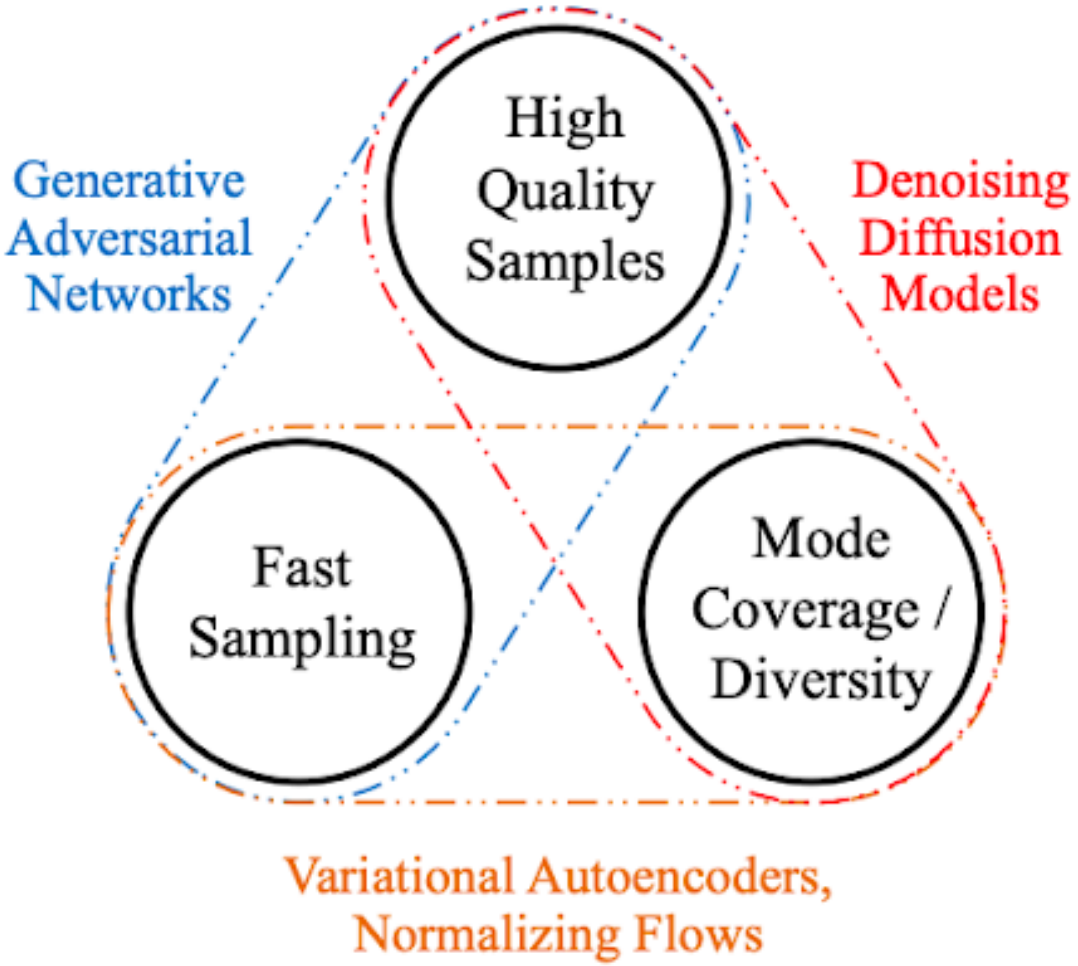
https://www.nytimes.com/2019/04/25/lens/sarah-lewis-racial-bias-photography.html

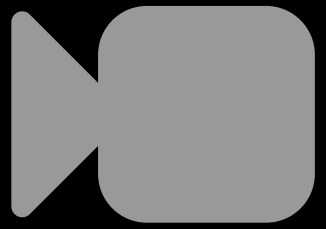
7/6 In defense of autoencoders

see also
https://arxiv.org/pdf/2103.04922.pdf



Which generative AI is right for you??





thank you!

federica bianco

University of Delaware
Department of Physics and Astronomy

Biden School of Public Policy and Administration
Data Science Institute

@fedhere

fbianco@udel.edu