FUNDAMENTALS OF DEEP LEARNING

PD. Dr. Juan J. Durillo - Deep Learning and GPU Programming Workshop @ LRZ – 16-19 May 2022

DEEP
LEARNING
INSTITUTE

THE GOALS OF THIS COURSE

- Get you up and on your feet quickly
- Build a foundation to tackle a deep learning project right away
- We won’t cover the whole field, but we’ll get a great head start
- Foundation from which to read articles, follow tutorials, take further classes

AGENDA

- Part 1: An Introduction to Deep Learning
- Part 2: How a Neural Network Trains
- Part 3: Convolutional Neural Networks
- Part 4: Data Augmentation and Deployment
- Part 5: Pre-trained Models
- Part 6: Advanced Architectures

HAVE FUN!

HISTORY OF AI

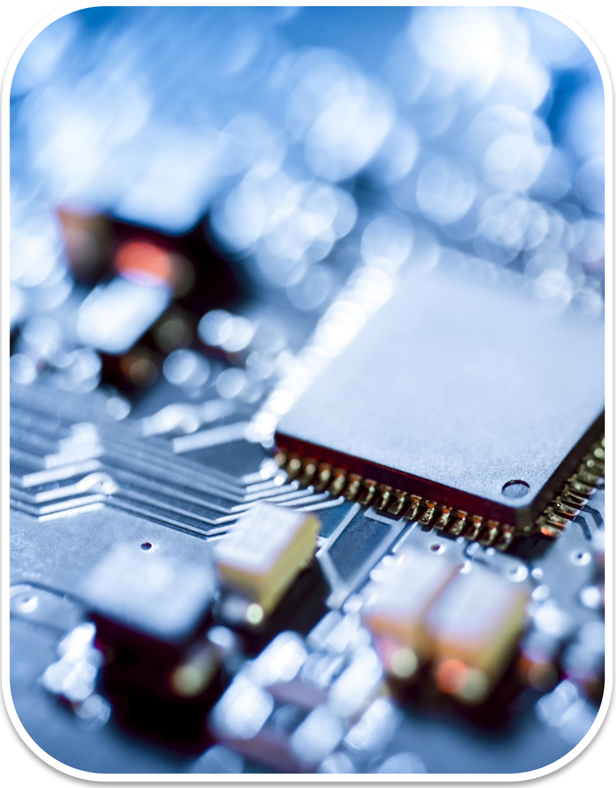
BEGINNING OF ARTIFICIAL INTELLIGENCE

COMPUTERS ARE MADE IN PART TO COMPLETE HUMAN TASKS

EARLY ON, GENERALIZED INTELLIGENCE LOOKED POSSIBLE

TURNED OUT TO BE HARDER THAN EXPECTED

EARLY NEURAL NETWORKS



Inspired by biology

Created in the 1950’s

Outclassed by Von Neumann Architecture

EXPERT SYSTEMS

Highly complex

Programmed by hundreds of engineers

Rigorous programming of many rules

EXPERT SYSTEMS - LIMITATIONS

What are these three images?

THE DEEP LEARNING REVOLUTION

DATA

- Networks need a lot of information to learn from
- The digital era and the internet has supplied that data

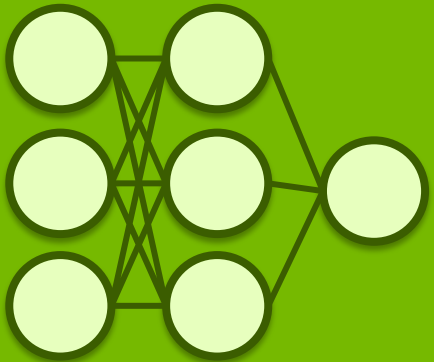
COMPUTING POWER

Need a way for our artificial “brain” to observe lots of data within a practical amount of time.

THE IMPORTANCE OF THE GPU

A Rendered Image

A Neural Network

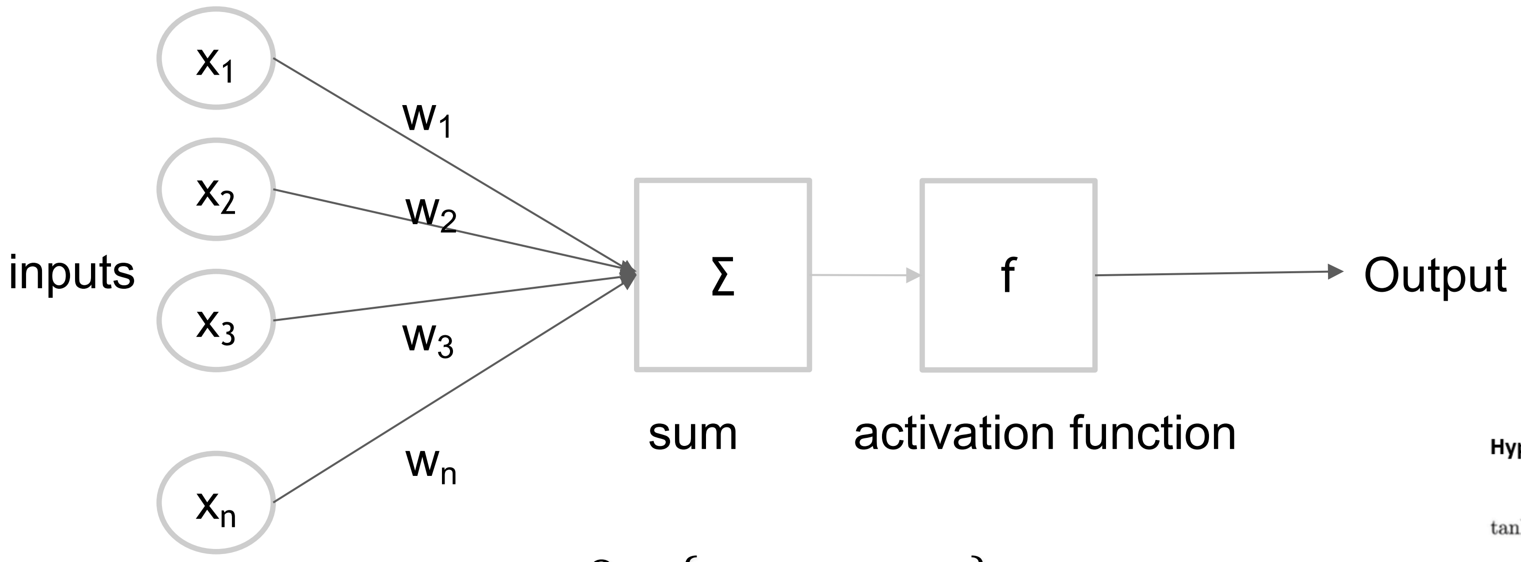


WHAT IS DEEP LEARNING?

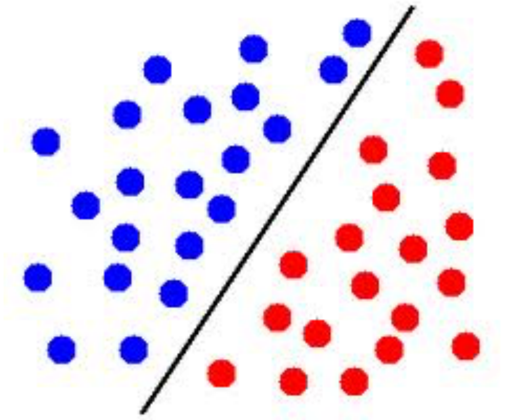
A (brief) introduction to Machine Learning

28.04.2021 | PD Dr. Juan J. Durillo

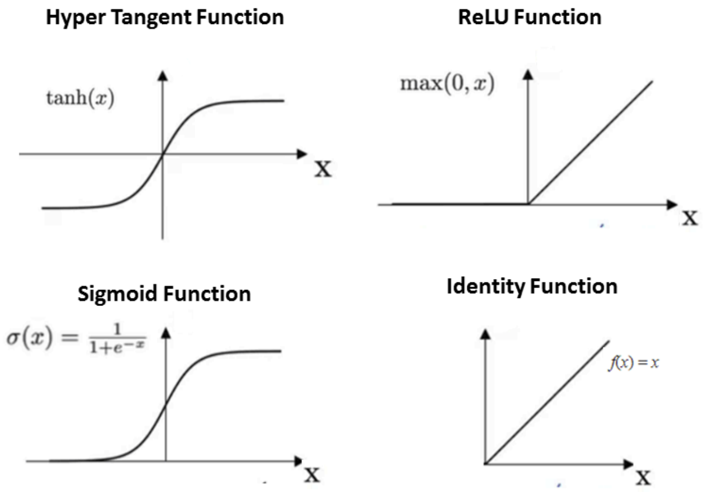
Perceptron - Artificial Neuron



$$\Theta = \{w_1\,, w_2\,\ldots, w_n\,\}$$



Single artificial neurons work well for linearly separable datasets (indeed output is the activation effect on a linear combination of the input)



Hyper Tangent Function

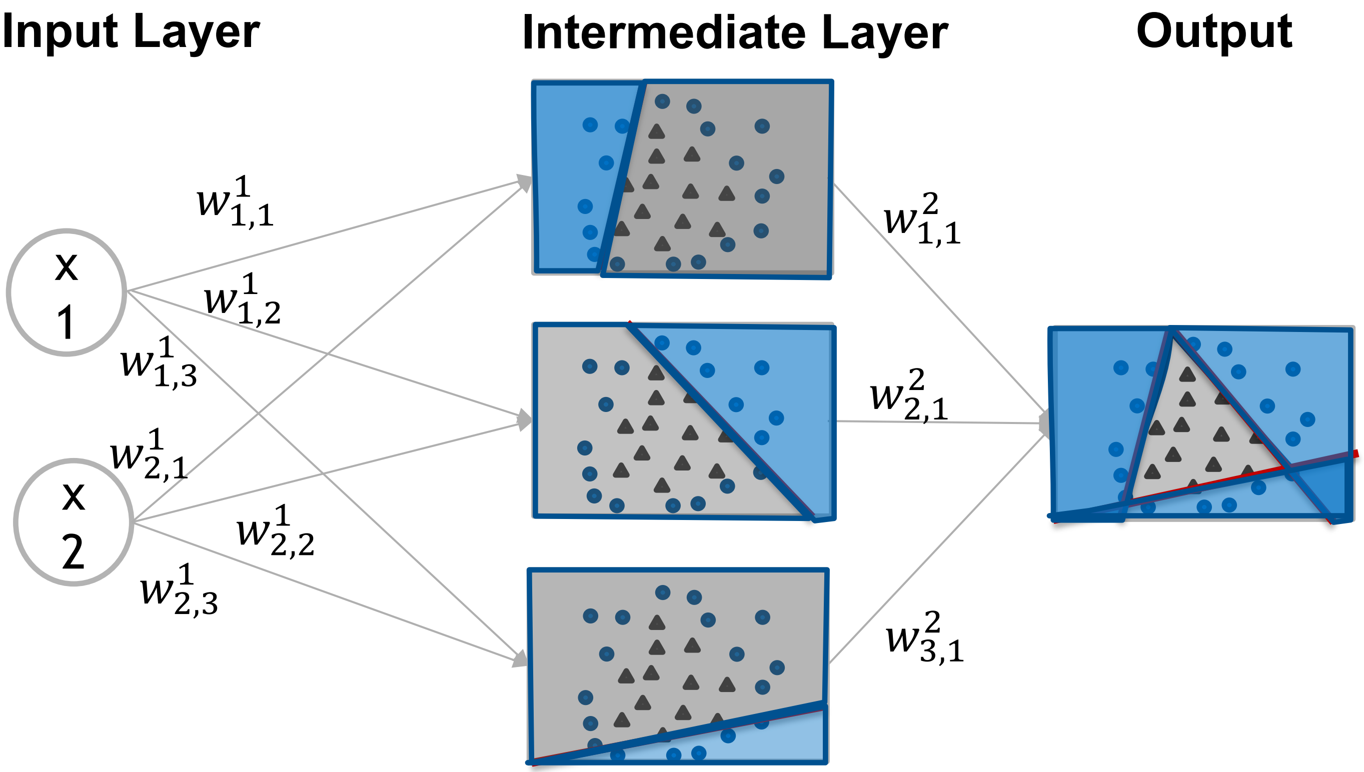
ReLU Function

Sigmoid Function

Identity Function

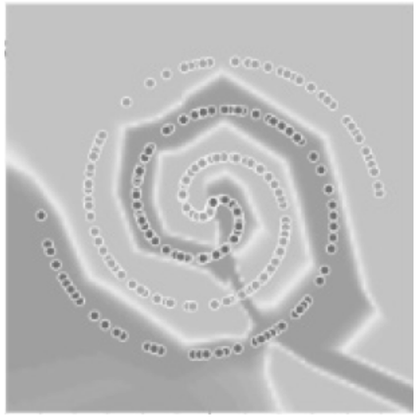
most popular activation functions

NEURAL NETWORK



$$\Theta = \{w^1_{1,1}, w^1_{1,2}, w^1_{1,3}, w^1_{2,1}, w^1_{2,2}, w^1_{2,3}, w^2_{1,1}, w^2_{2,1}, w^2_{2,3}\}$$

- Works well even when the data is not linearly separable



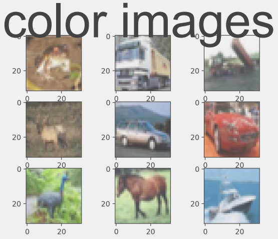
(SUPERVISED) LEARNING

- Data domain Z: $\mathrm{X}\times\Upsilon$

X → domain of the input data

Υ → set of labels (knowledge)

X: 32 x 32 color images



Υ : labels

{truck, car, horse, bird, boat}

Example (CIFAR10 dataset)

- Data Distribution is a probability distribution over a data domain
- Training set $z_1, \ldots, z_n$ from Z assumed to be drawn from the Data Distribution D
- Validation set $v_1, \ldots, v_m$ from Z also assumed to be drawn from D
- A machine learning model is a function that given a set of parameters Θ and z from Z produces a prediction
- The prediction quality is measured by a differentiable non-negative scalar-valued loss function, that we denote $\ell(\Theta; z)$

(SUPERVISED) LEARNING

- Given Θ we can define the expected loss as: $L(\Theta) = \mathbb{E}_{z\sim D}[\ell(\Theta; z)]$
- Given D, ℓ, and a model with parameter set Θ, we can define learning as:

  "The task of finding parameters Θ that achieve low values of the expected loss, while we are given access to only n training examples"

- The mentioned task before is commonly referred to as *training*
- Empirical average loss given a subset of the training data set S(z_1, …, z_n) as:

$$\hat{L}(\Theta) = \frac{1}{n}\sum_{t=1}^{n}[\ell(\Theta; z_t)]$$

- Usually a proxy function, easier to understand by humans, is used for describing how well the training is performed (e.g., accuracy)

(SUPERVISED) LEARNING

- The dominant algorithms for training neural networks are based on mini-batch stochastic gradient descent (SGD)

- Given an initial point Θ_0 SGD attempt to decrease $\hat{L}$ via the sequence of iterates

$$\Theta_t \longleftarrow \Theta_{t-1} - n_t g(\Theta_{t-1}; B_t)$$

$$g(\Theta; B) = \frac{1}{|B|} \sum\nolimits_{z \epsilon B} \nabla \ell(\Theta; z)$$

B_t: random subset of training examples

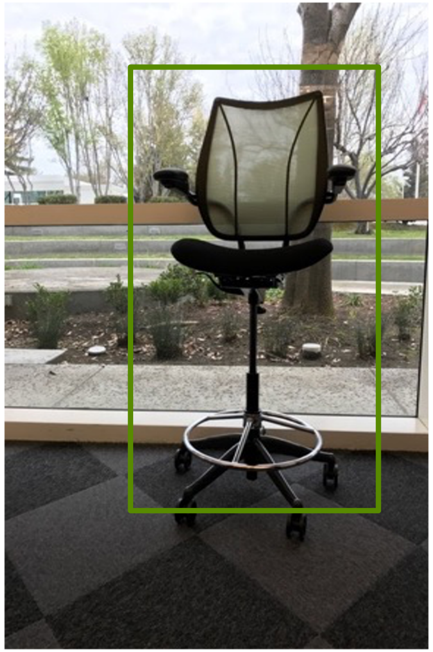
n_t: positive scalar (learning rate)

epoch: update the weights after going over all training set

COMPUTER VISION TASKS

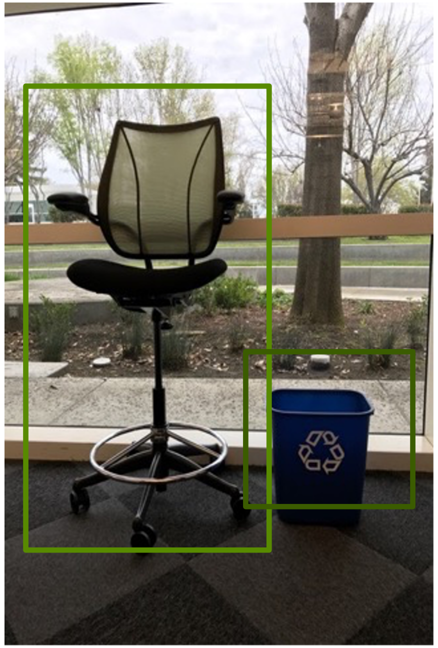
predicting the type or class of an object in an image

Image Classification



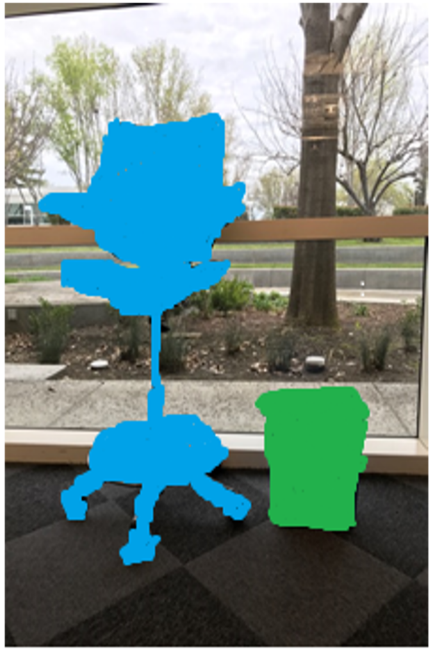
predicting the type or class on an object in an image and draw a bounding box around

Image Classification + Localization



predicting the location of objects in an image via bounding boxes and the classes of the located objects

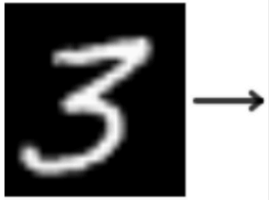
Object Detection



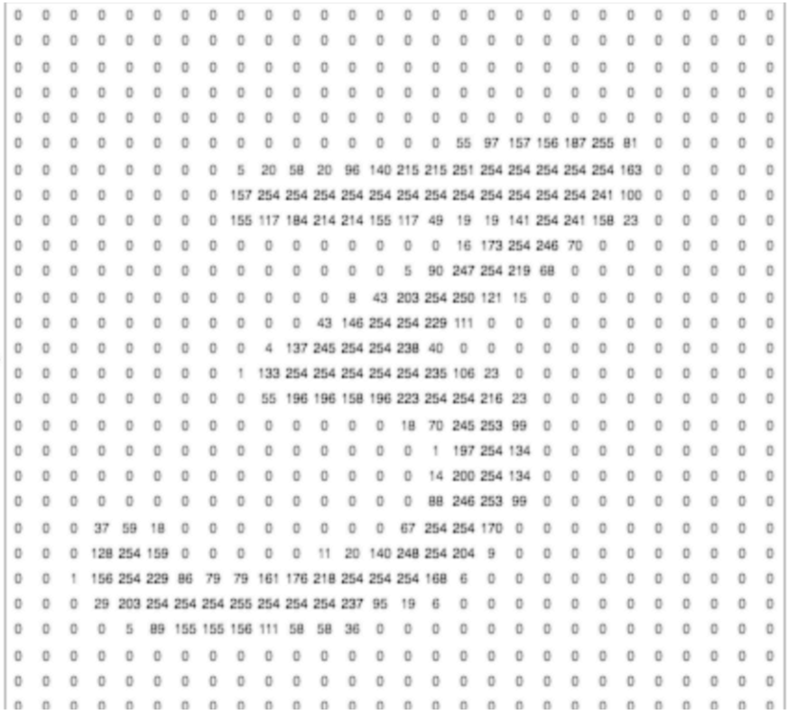
predicting the class to which each pixel in the image belongs to

Image Segmentation

ON INPUT REPRESENTATION

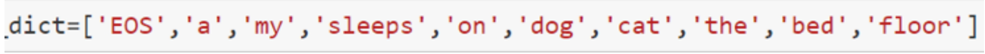


28 x 28
= 784 pixels



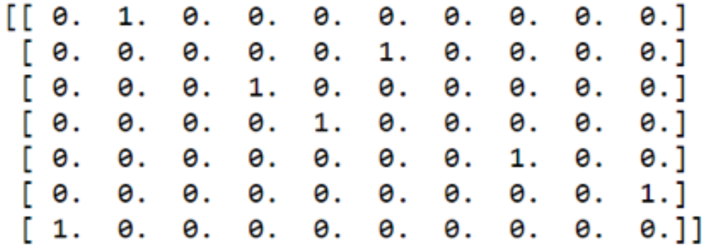
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	97	157	156	187	255	81	0	0	0	0	0
0	0	0	0	0	0	0	0	5	20	58	20	96	140	215	215	251	254	254	254	254	254	163	0	0	0	0	0
0	0	0	0	0	0	0	0	157	254	254	254	254	254	254	254	254	254	254	254	254	241	100	0	0	0	0	0
0	0	0	0	0	0	0	0	155	117	184	214	214	155	117	49	19	19	141	254	241	158	23	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	173	254	246	70	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	90	247	254	219	68	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	8	43	203	254	250	121	15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	43	146	254	254	229	111	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	137	245	254	254	238	40	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	133	254	254	254	254	254	235	106	23	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	55	196	196	158	196	223	254	254	216	23	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	70	245	253	99	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	197	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	200	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	246	253	99	0	0	0	0	0	0	0	0	0
0	0	0	37	59	18	0	0	0	0	0	0	0	0	67	254	254	170	0	0	0	0	0	0	0	0	0	0
0	0	0	128	254	159	0	0	0	0	0	11	20	140	248	254	204	9	0	0	0	0	0	0	0	0	0	0
0	0	1	156	254	229	86	79	79	161	176	218	254	254	254	168	6	0	0	0	0	0	0	0	0	0	0	0
0	0	0	29	203	254	254	254	255	254	254	254	237	95	19	6	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	89	155	155	156	111	58	58	36	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

image



```
dict=['EOS','a','my','sleeps','on','dog','cat','the','bed','floor']
```

sentence = `['a', 'dog', 'sleeps', 'on', 'the', 'floor', 'EOS']`

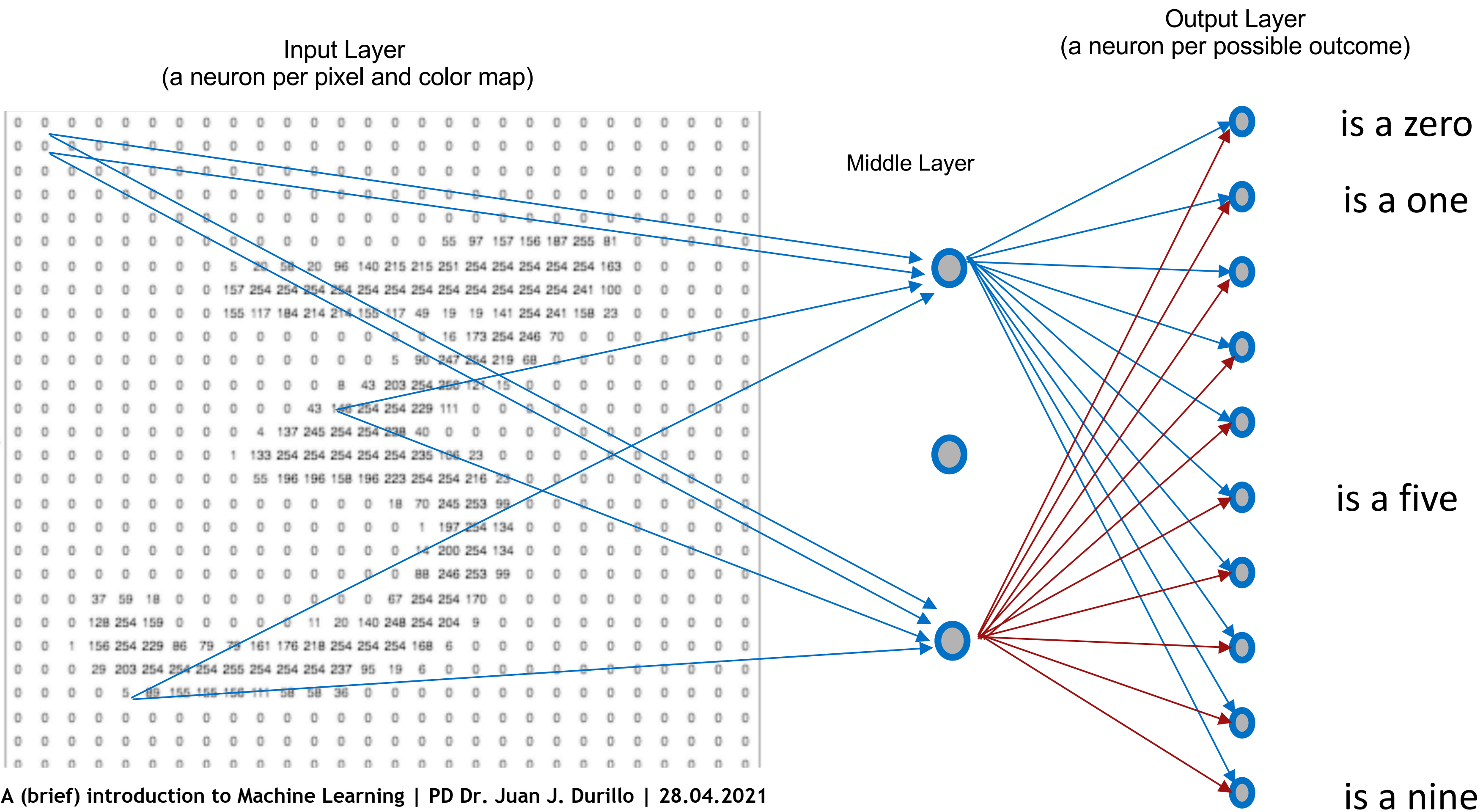


```
[[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

language

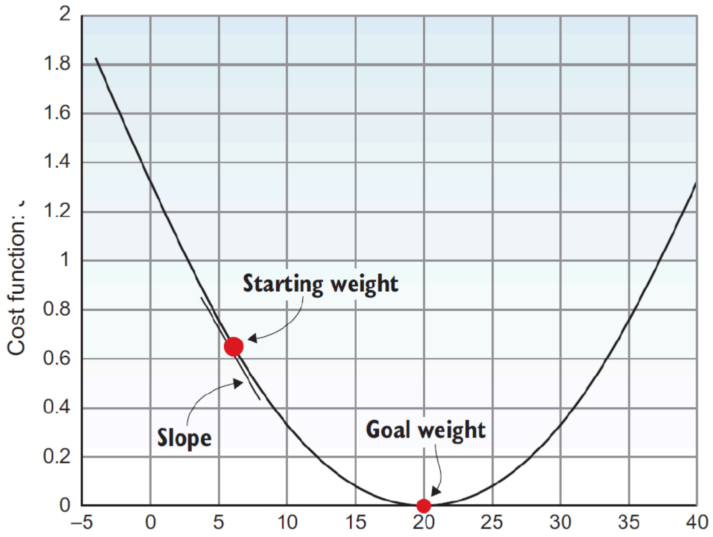
NEURAL NETWORKS FOR IMAGE CLASSIFICATION

Ful

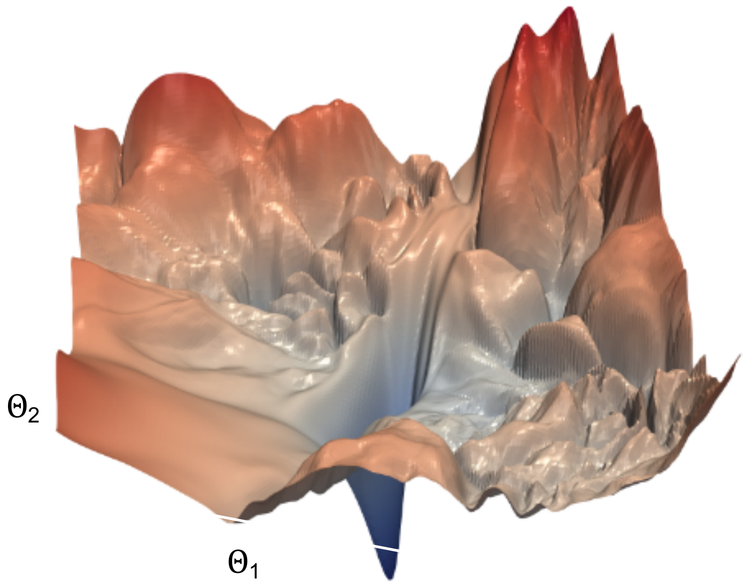


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	97	157	156	187	255	81	0	0	0	0	0
0	0	0	0	0	0	0	0	5	20	58	20	96	140	215	215	251	254	254	254	254	254	163	0	0	0	0	0
0	0	0	0	0	0	0	0	157	254	254	254	254	254	254	254	254	254	254	254	254	241	100	0	0	0	0	0
0	0	0	0	0	0	0	0	155	117	184	214	214	155	117	49	19	19	141	254	241	158	23	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	173	254	246	70	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	90	247	254	219	68	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	8	43	203	254	250	121	15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	43	148	254	254	229	111	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	137	245	254	254	238	40	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	133	254	254	254	254	254	235	106	23	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	55	196	196	158	196	223	254	254	216	23	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	70	245	253	99	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	197	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	200	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	246	253	99	0	0	0	0	0	0	0	0	0
0	0	0	37	59	18	0	0	0	0	0	0	0	0	67	254	254	170	0	0	0	0	0	0	0	0	0	0
0	0	0	128	254	159	0	0	0	0	0	11	20	140	248	254	204	9	0	0	0	0	0	0	0	0	0	0
0	0	1	156	254	229	86	79	79	161	176	218	254	254	254	168	6	0	0	0	0	0	0	0	0	0	0	0
0	0	0	29	203	254	254	254	255	254	254	254	237	95	19	6	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	89	155	155	156	111	58	58	36	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

TRAINING NEURAL NETWORKS



main idea



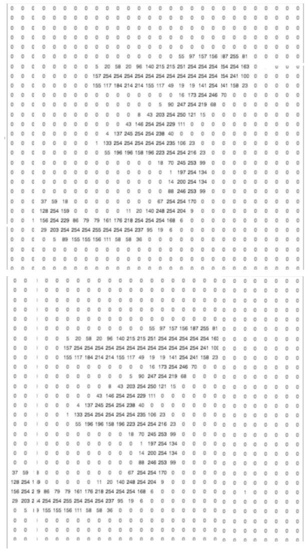
how the surface looks like in reality

Stochastic Gradient Descent

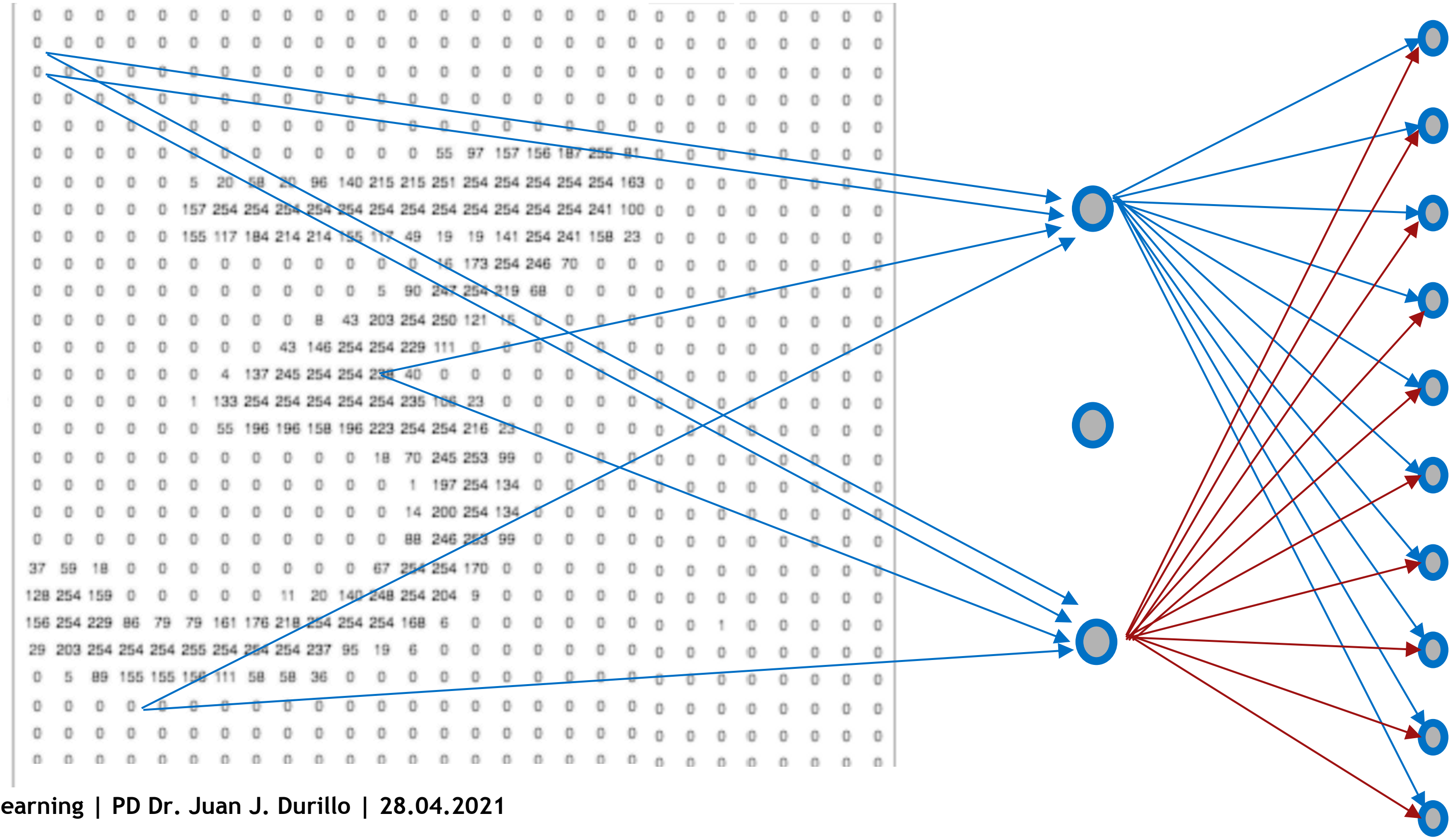
$$\Theta_t \leftarrow \Theta_{t-1} - \eta_t g(\Theta_{t-1}; B_t)$$

$$g(\Theta; B) = \frac{1}{|B|} \sum_{z \epsilon B} \nabla \ell(\Theta; z)$$

NEURAL NETWORKS FOR IMAGE CLASSIFICATION



shift to the left



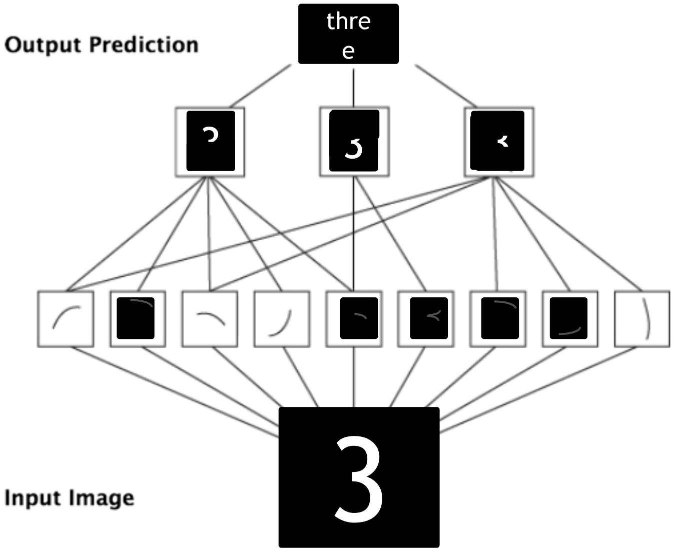
is a zero

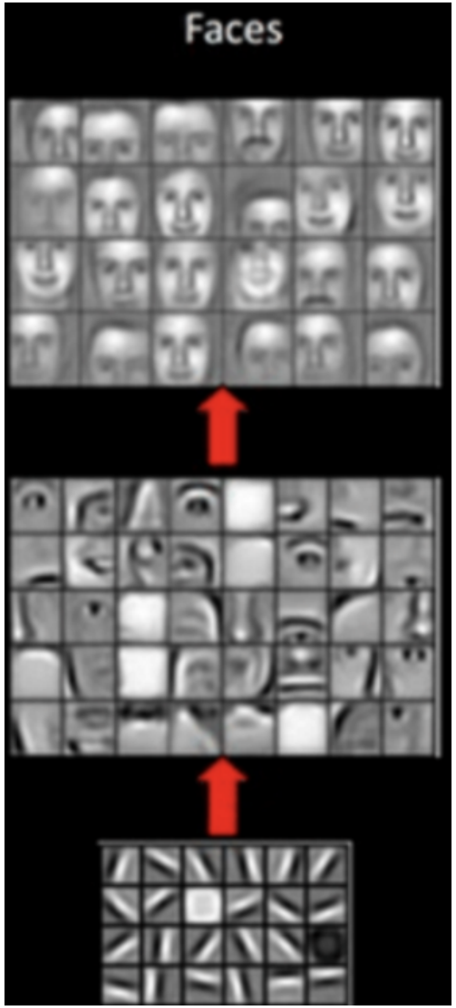
is a one

is a five

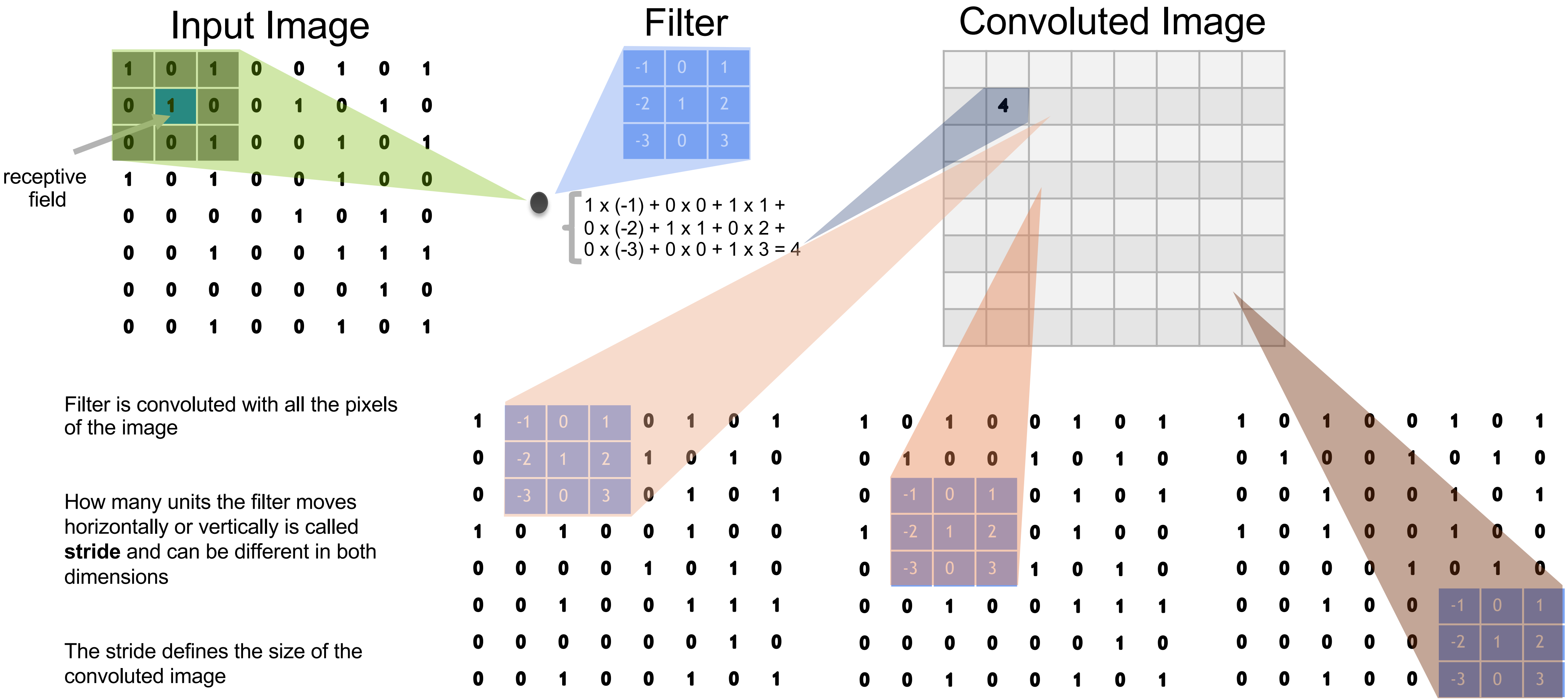
is a nine

NO MORE FEATURE ENGINEERING





LEARNING FEATURES FROM DATA: CONVOLUTIONS



Input Image

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

Filter

-1	0	1
-2	1	2
-3	0	3

1 x (-1) + 0 x 0 + 1 x 1 +
0 x (-2) + 1 x 1 + 0 x 2 +
0 x (-3) + 0 x 0 + 1 x 3 = 4

Convoluted Image

	4						

Filter is convoluted with all the pixels of the image

1	-1	0	1	0	1	0	1
0	-2	1	2	1	0	1	0
0	-3	0	3	0	1	0	1
1	0	1	0	0	1	0	0
0	0	0	0	1	0	1	0
0	0	1	0	0	1	1	1
0	0	0	0	0	0	1	0
0	0	1	0	0	1	0	1

How many units the filter moves horizontally or vertically is called **stride** and can be different in both dimensions

1	0	1	0	0	1	0	1
0	1	0	0	1	0	1	0
0	-1	0	1	0	1	0	1
1	-2	1	2	0	1	0	0
0	-3	0	3	1	0	1	0
0	0	1	0	0	1	1	1
0	0	0	0	0	0	1	0
0	0	1	0	0	1	0	1

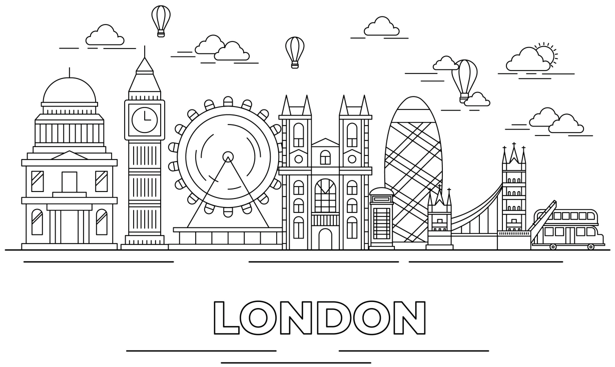
The stride defines the size of the convoluted image

1	0	1	0	0	1	0	1
0	1	0	0	1	0	1	0
0	0	1	0	0	1	0	1
1	0	1	0	0	1	0	0
0	0	0	0	1	0	1	0
0	0	1	0	0	-1	0	1
0	0	0	0	0	-2	1	2
0	0	1	0	0	-3	0	3

*The London skyline image is designed by Freepik

FILTERS

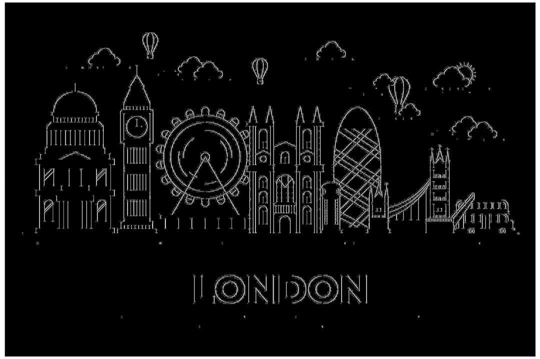
Input Image:



Can we get only vertical lines out of this picture?

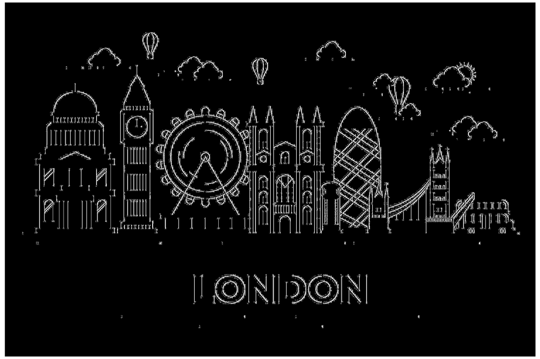
$$\begin{matrix} 1 & 0 & -1 \end{matrix}$$

filter 1



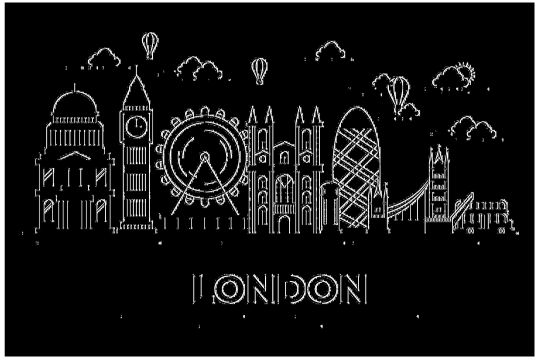
$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

filter 2



$$\begin{matrix} 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \end{matrix}$$

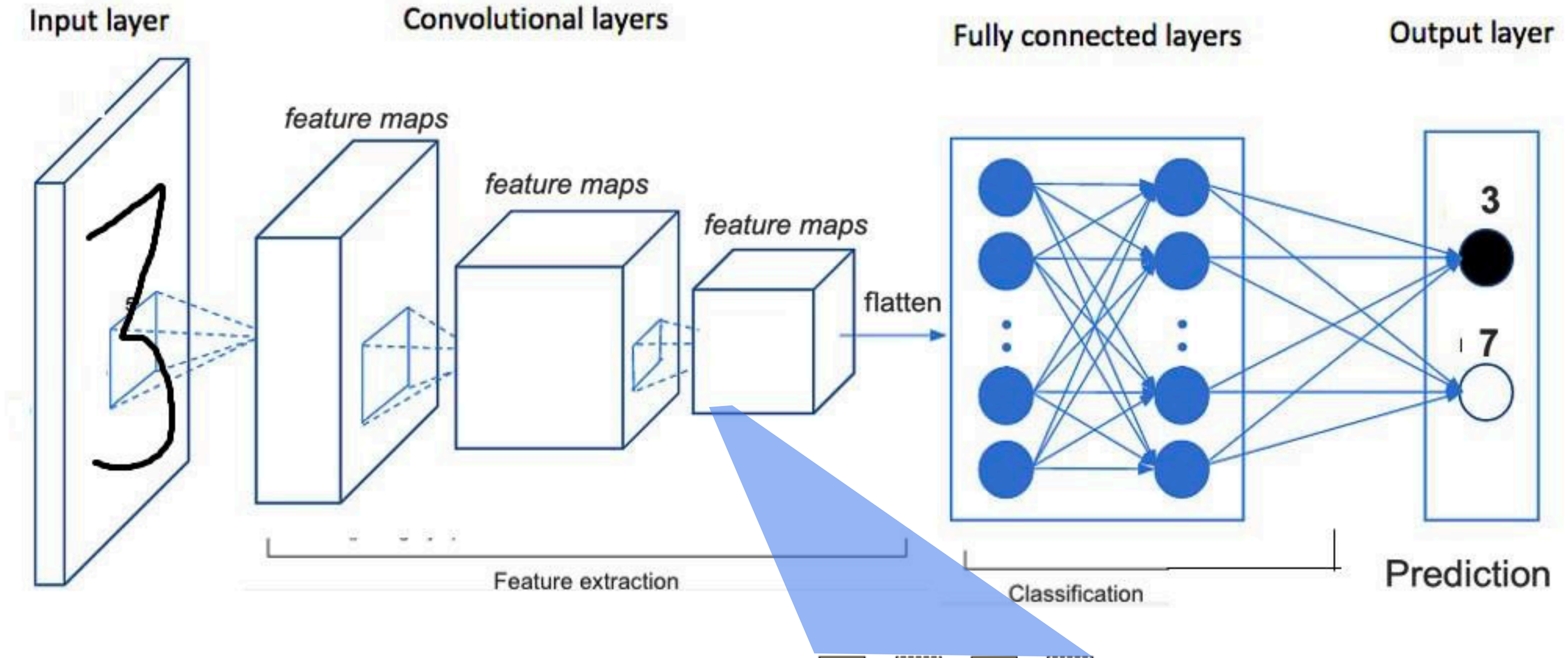
filter 3

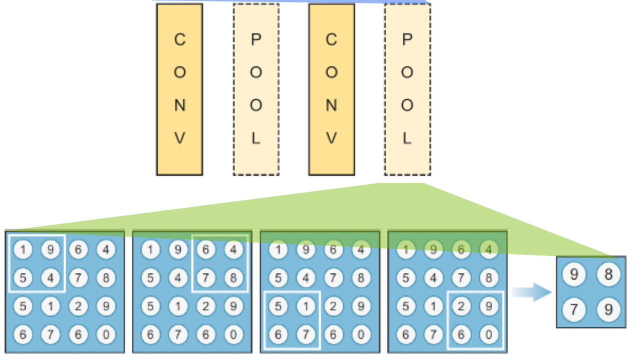


try the code yourself (in octave)!

```
 I=imread(<path-to-image>);
GRAY=rgb2gray(I)
FILTER=[ 1 0 -1; 1 0 -1; 1 0 -1]; % filter 2
CONVOLUTED=conv2(GREY,FILTER);
Imwrite(CONVOLUTED, <path-to-result>);
```

CONVOLUTIONAL NEURAL NETWORKS (CNN)





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

9	8
7	9

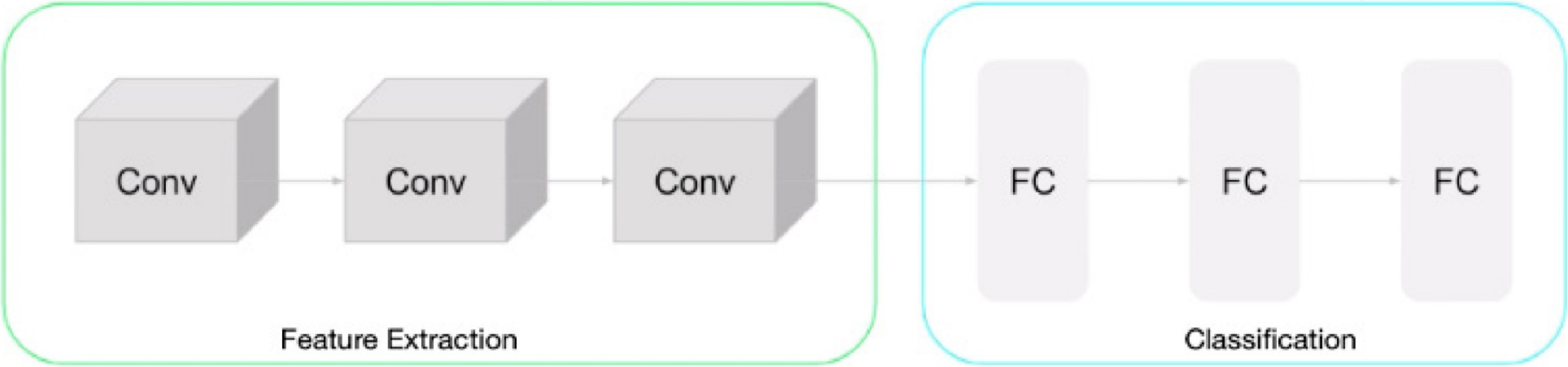
A pooling layer down sample the feature maps produced by a convolution into smaller number of parameters to reduce the computational complexity.

It is a common practice to add pooling layers after each one or two convolutions layers in the CNN architecture.

CNN ARCHITECTURE: A COMMON PATTERN AND ITS INFLUENCE

Convolutional Layers

Classification Layer

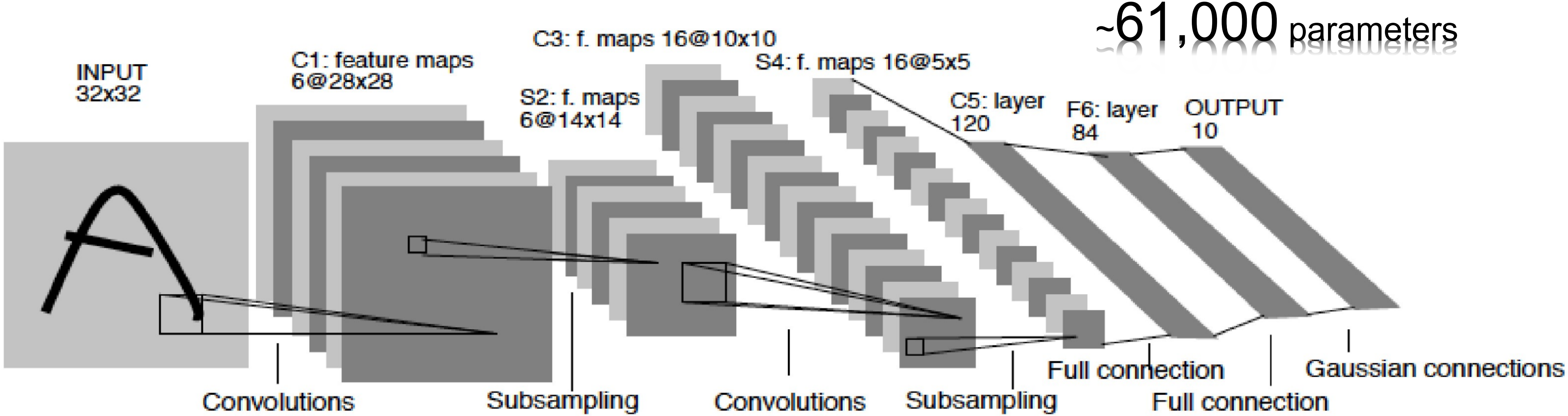


The execution time required during a forward pass through a neural network is bounded from below by the number of floating point operations (FLOPs).

This FLOP count depends on the deep neural network architecture and the amount of data.

LENET ARCHITECTURE

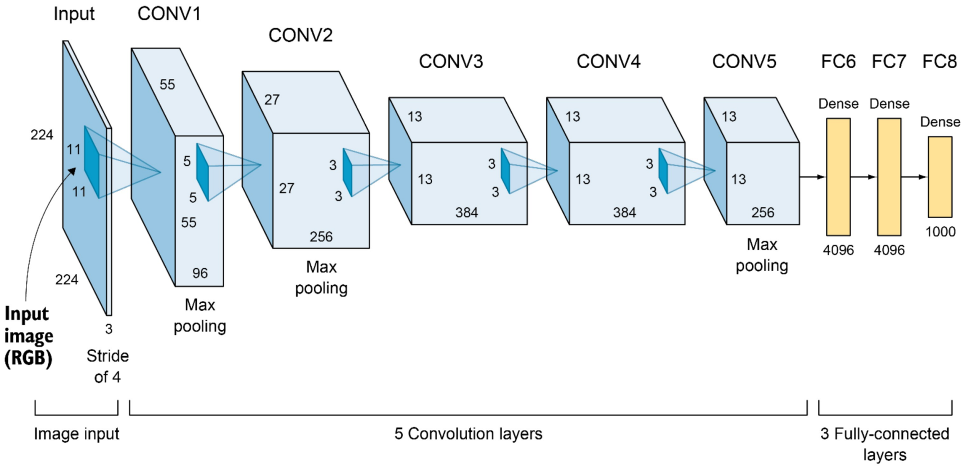
~61,000 parameters



Architecture summary :

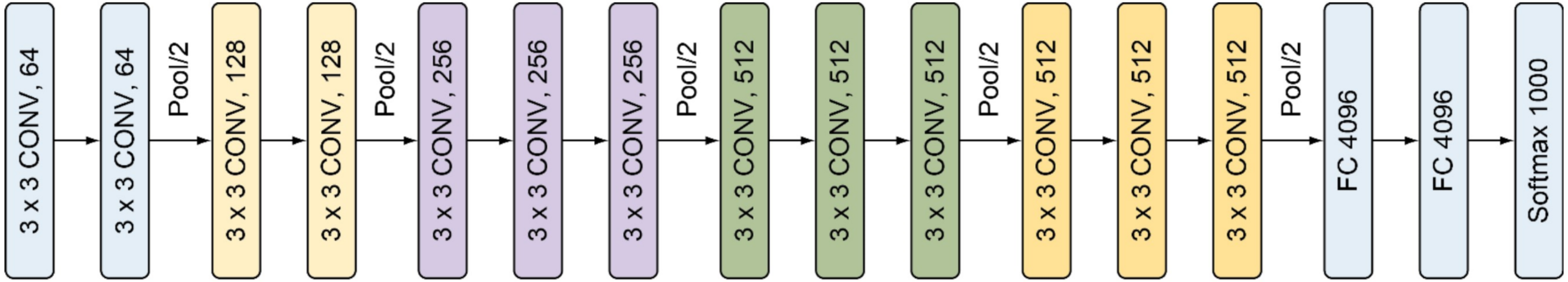
- 3 convolutional layers filters in all the layers equal to 5x5
  (layer 1 depth = 6, layer 2 depth = 16, layer 3 depth = 120)
- As activation function the tanh function is used

ALEXNET AND VGG ARCHITECTURES



~60,000,000 parameters

AlexNet

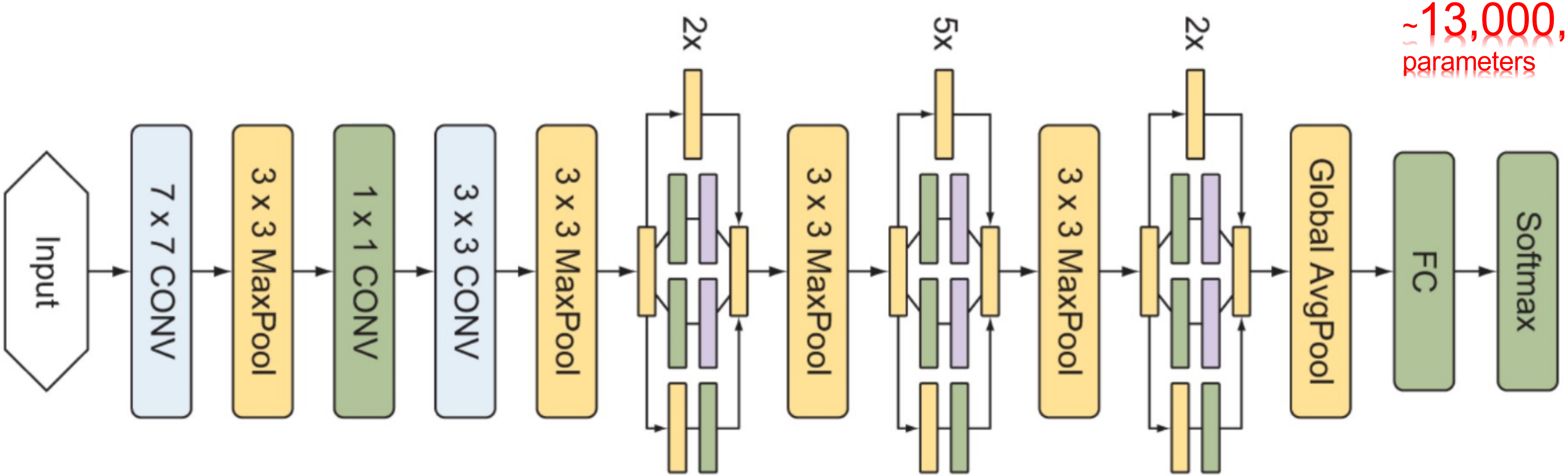


~138,000,000 parameters

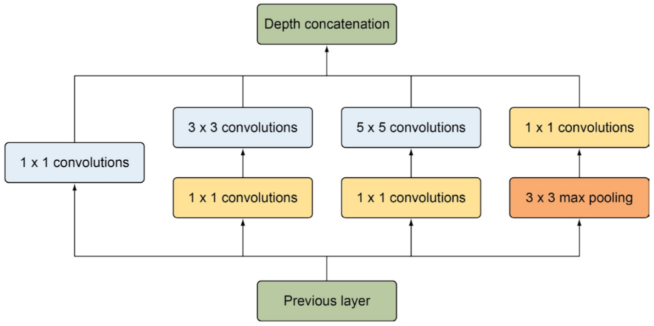
VGG16

GOOGLENET

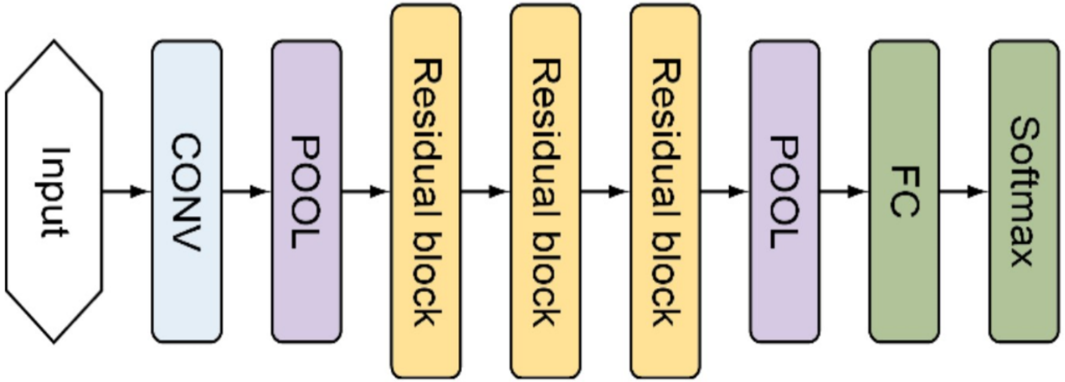
~13,000,000 parameters

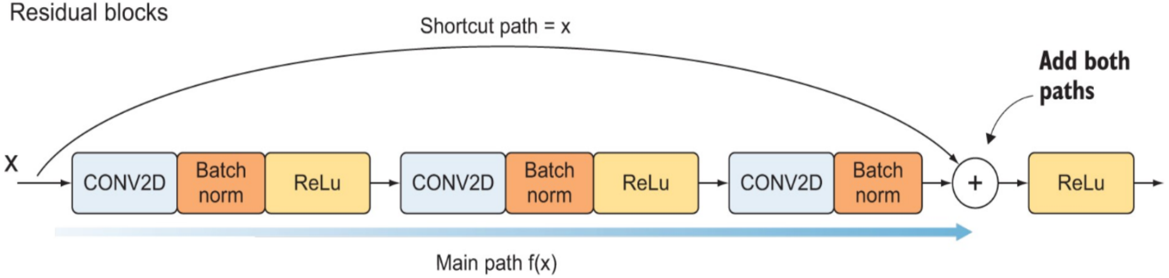


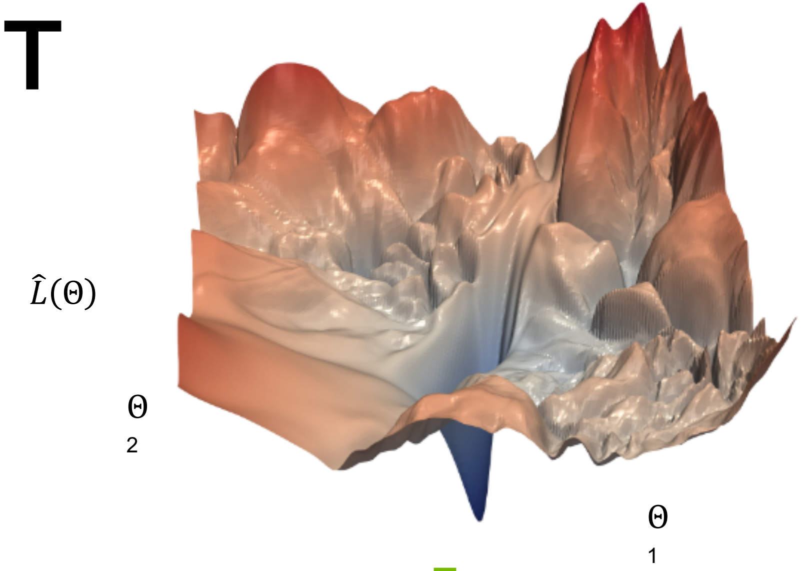
- What is the best kernel size for each layer?
- Concatenating filters instead of stacking them for reducing computational expenses



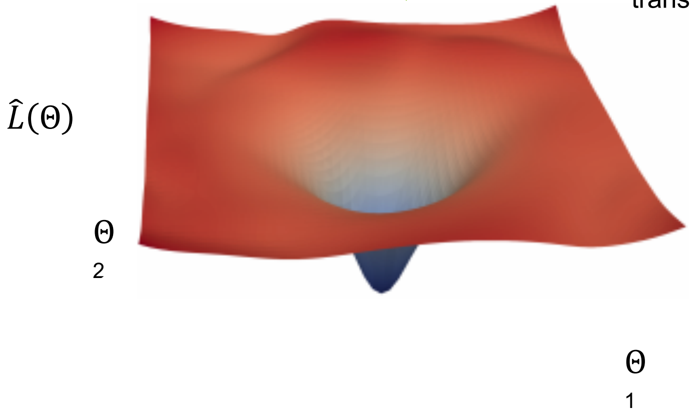
RESTNET





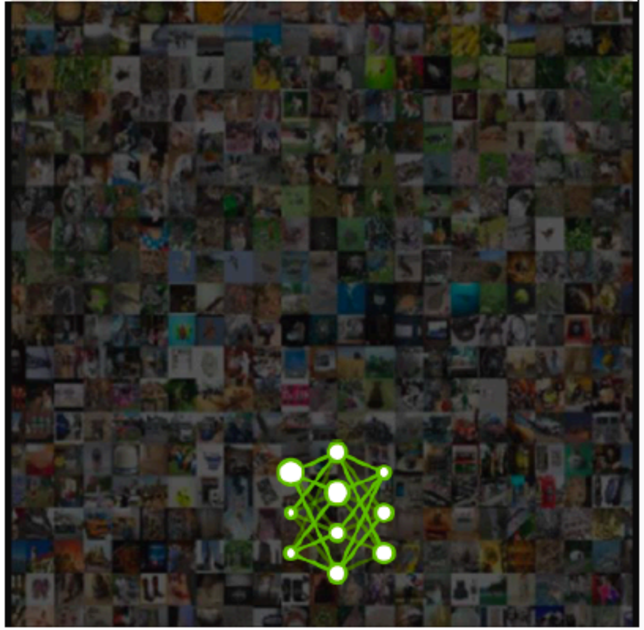


Thanks to the shortcut is transformed into



INCREASING COMPLEXITY

7 Exaflops
60 Million Parameters

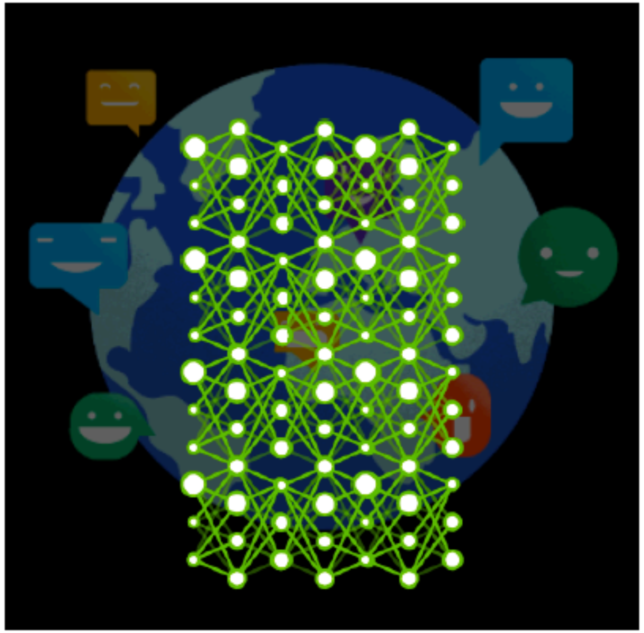


2015 - Microsoft ResNet
Superhuman Image Recognition

20 Exaflops
300 Million Parameters

2016 - Baidu Deep Speech 2
Superhuman Voice Recognition

100 Exaflops
8700 Million Parameters



2017 - Google Neural Machine Translation
Near Human Language Translation

SUMMARY

Brief introduction to Deep Learning with emphasis in Deep Convolutional Neural Networks

Review of basic concepts: from perceptron to the learning task

Debrief of most important concepts of neural network architectures

DEEP LEARNING FLIPS TRADITIONAL PROGRAMMING ON ITS HEAD

TRADITIONAL PROGRAMMING

Building a Classifier

1

Define a set of rules for classification

2

Program those rules into the computer

3

Feed it examples, and the program uses the rules to classify

MACHINE LEARNING

Building a Classifier

1

Show model the examples with the answer of how to classify

2

Model takes guesses, we tell it if it's right or not

3

Model learns to correctly categorize as it's training. The system learns the rules on its own

THIS IS A FUNDAMENTAL SHIFT

WHEN TO CHOOSE DEEP LEARNING

Classic Programming

If rules are clear and straightforward, often better to just program it

Deep Learning

If rules are nuanced, complex, difficult to discern, use deep learning

DEEP LEARNING COMPARED TO OTHER AI

- Depth and complexity of networks
- Up to billions of parameters (and growing)
- Many layers in a model
- Important for learning complex rules

HOW DEEP LEARNING IS TRANSFORMING THE WORLD

COMPUTER VISION

ROBOTICS AND MANUFACTURING

OBJECT DETECTION

SELF DRIVING CARS

NATURAL LANGUAGE PROCESSING

REAL TIME TRANSLATION

VOICE RECOGNITION

VIRTUAL ASSISTANTS

RECOMMENDER SYSTEMS

CONTENT CURATION

TARGETED ADVERTISING

SHOPPING RECOMMENDATIONS

REINFORCEMENT LEARNING

ALPHAGO BEATS WORLD CHAMPION IN GO

AI BOTS BEAT PROFESSIONAL VIDEOGAMERS

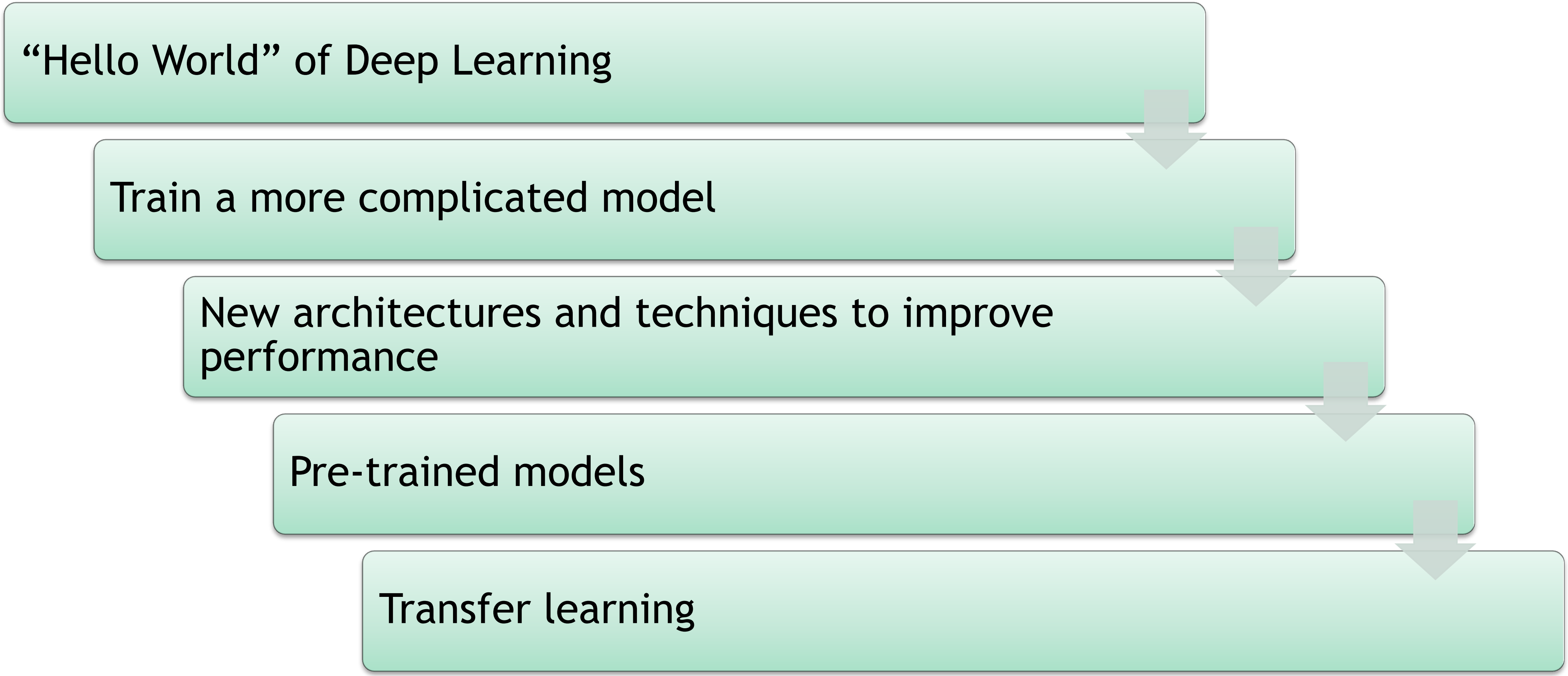
STOCK TRADING ROBOTS

OVERVIEW OF THE COURSE

HANDS ON EXERCISES

- Get comfortable with the process of deep learning
- Exposure to different models and datatypes
- Get a jump-start to tackle your own projects

STRUCTURE OF THE COURSE



“Hello World” of Deep Learning

Train a more complicated model

New architectures and techniques to improve performance

Pre-trained models

Transfer learning

PLATFORM OF THE COURSE

GPU powered cloud server

JupyterLab platform

Jupyter notebooks for interactive coding

SOFTWARE OF THE COURSE

- Major deep learning platforms:
  - TensorFlow + Keras (Google)
  - Pytorch (Facebook)
  - MXNet (Apache)
- We’ll be using TensorFlow and Keras
- Good idea to gain exposure to others moving forward

FIRST EXERCISE: CLASSIFY HANDWRITTEN DIGITS

HELLO NEURAL NETWORKS

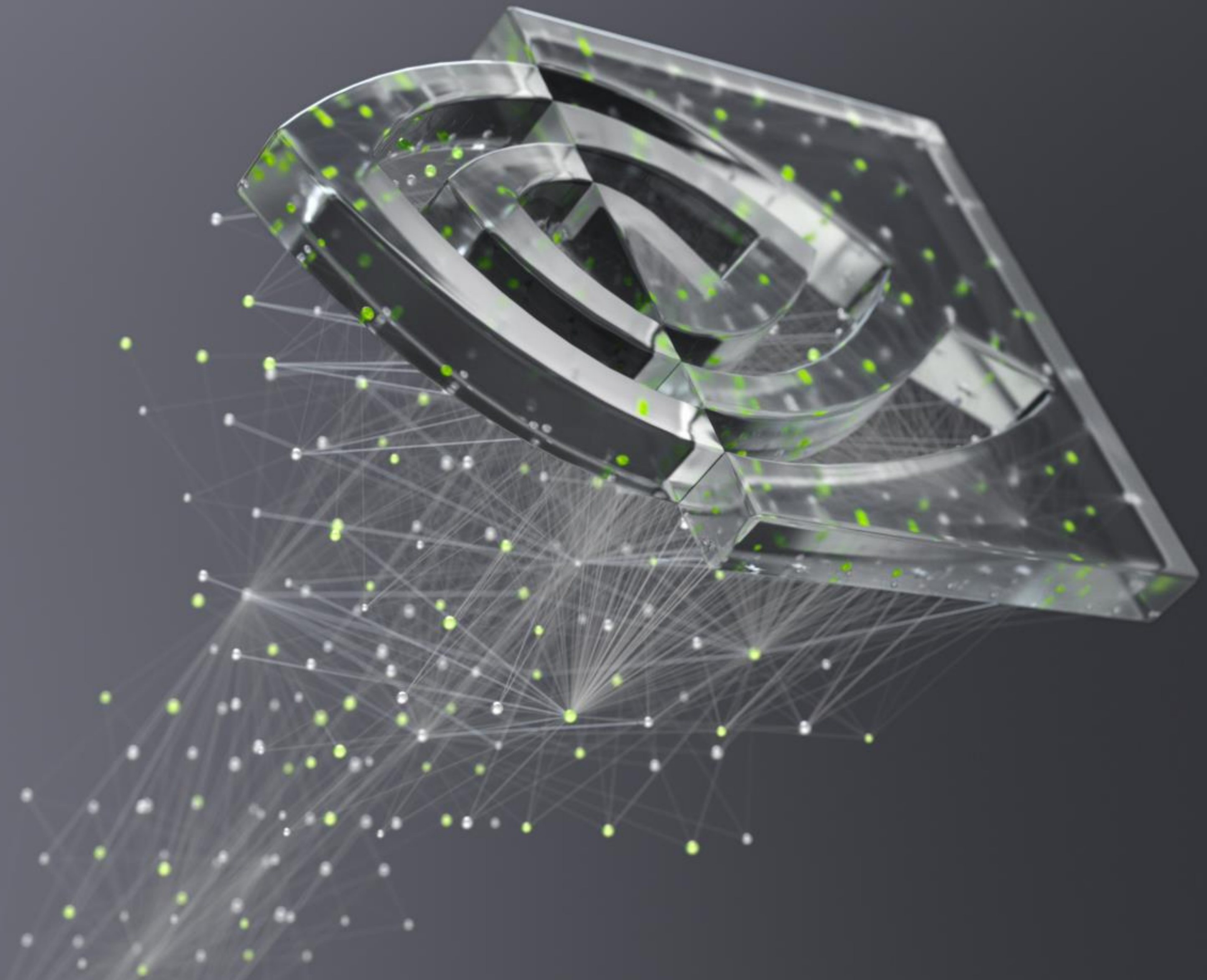
Train a network to correctly classify handwritten digits

- Historically important and difficult task for computers

Try learning like a Neural Network

- Get exposed to the example, and try to figure out the rules to how it works

LET’S GO!



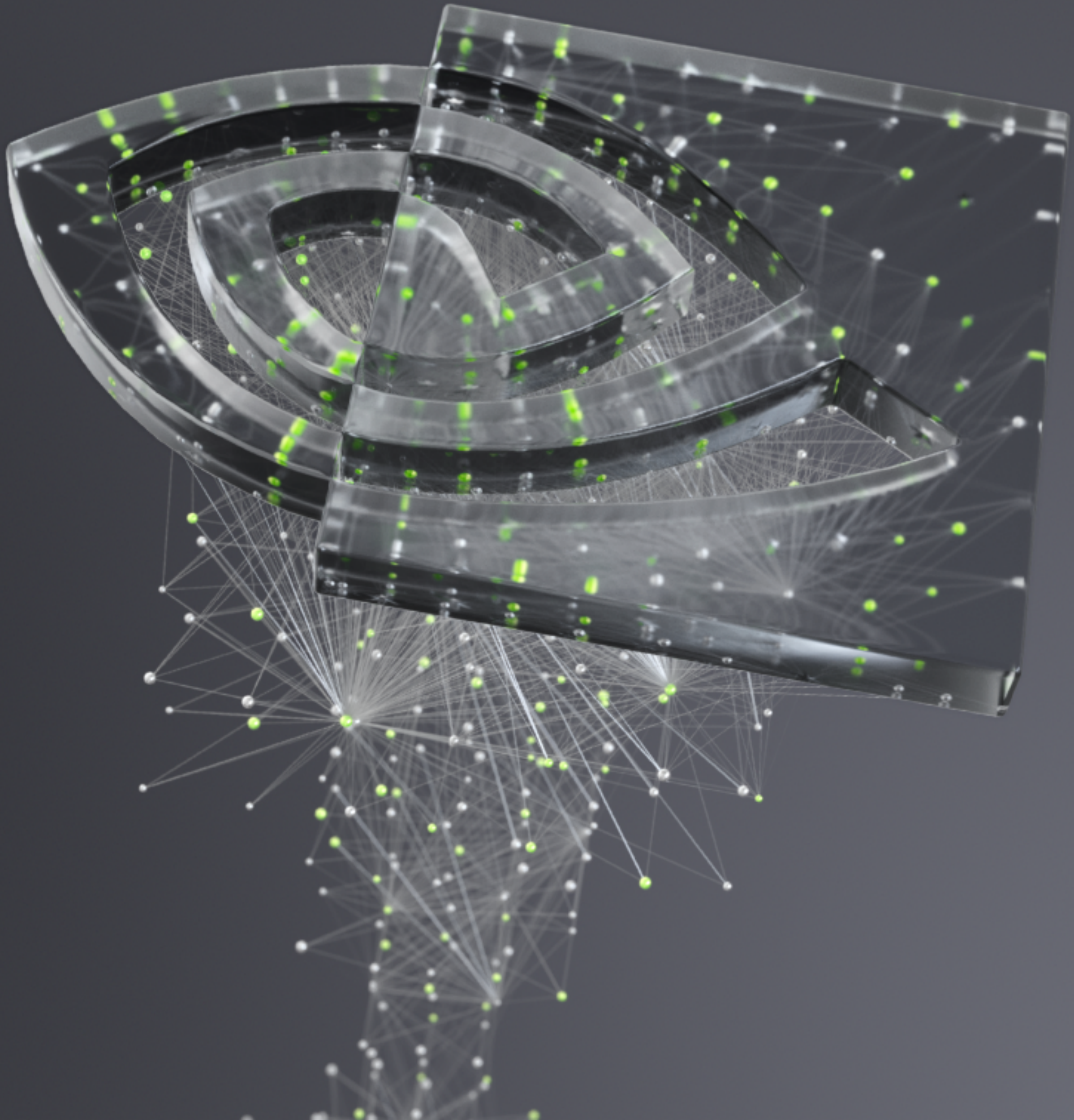
NVIDIA

DEEP
LEARNING
INSTITUTE

NVIDIA | DEEP LEARNING INSTITUTE

FUNDAMENTALS OF DEEP LEARNING

Part 2: How a Neural Network Trains



Part 1: An Introduction to Deep Learning

Part 2: How a Neural Network Trains

Part 3: Convolutional Neural Networks

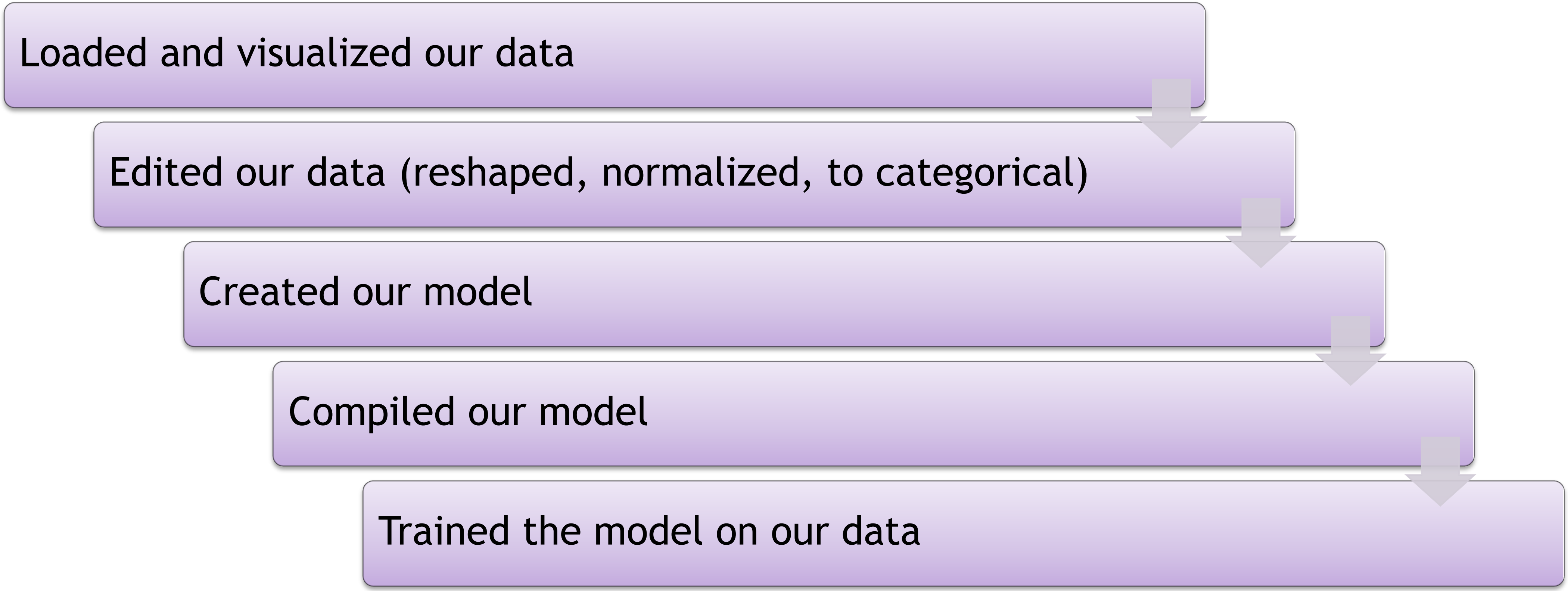
Part 4: Data Augmentation and Deployment

Part 5: Pre-trained Models

Part 6: Advanced Architectures

RECAP OF THE EXERCISE

What just happened?



Loaded and visualized our data

Edited our data (reshaped, normalized, to categorical)

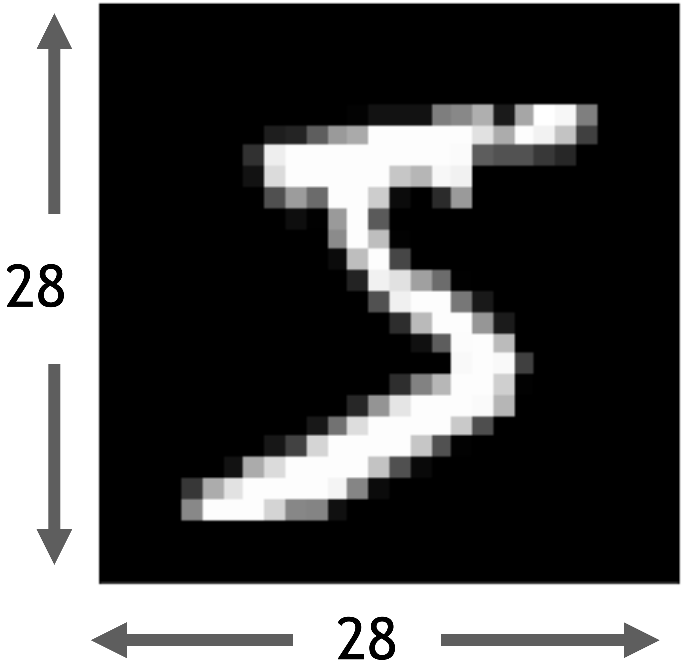
Created our model

Compiled our model

Trained the model on our data

DATA PREPARATION

Input as an array



[0,0,0,24,75,184,185,78,32,55,0,0,0...]

DATA PREPARATION

Targets as categories

0 → [1,0,0,0,0,0,0,0,0,0]

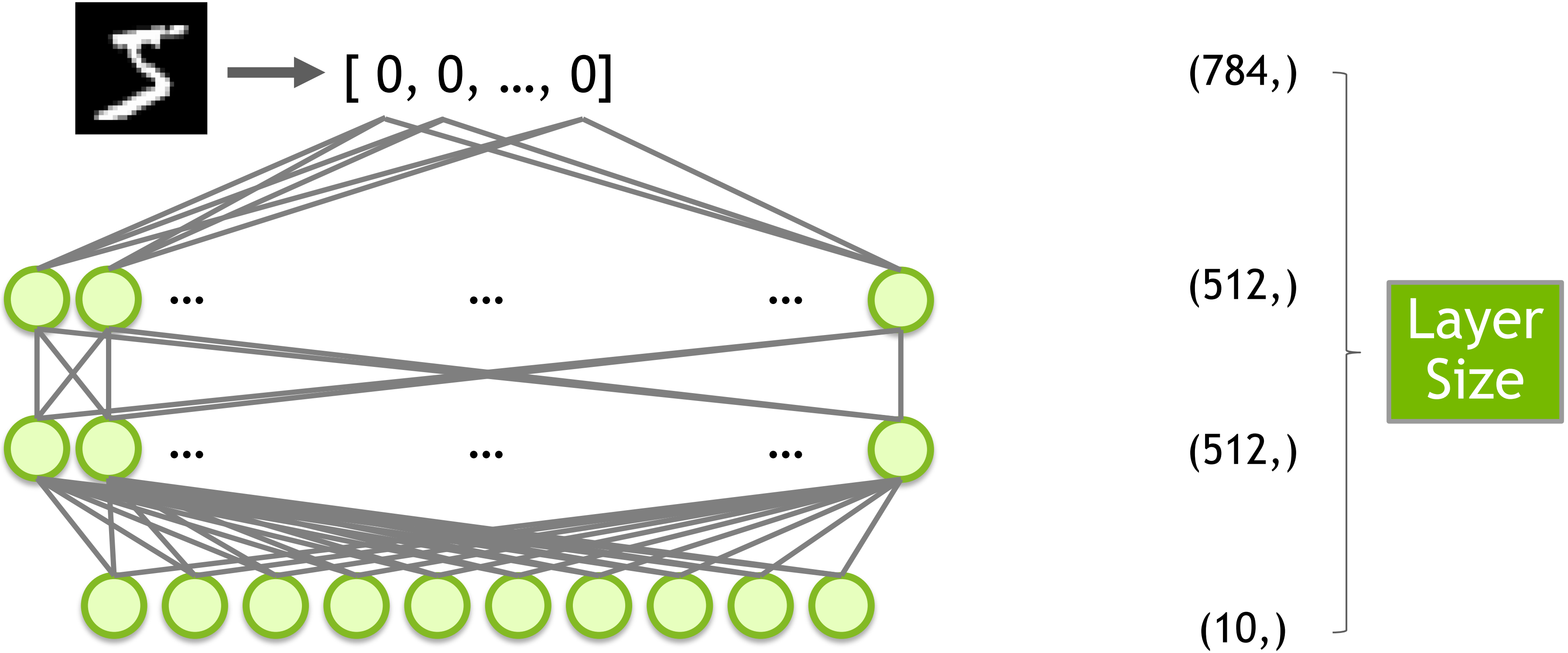
1 → [0,1,0,0,0,0,0,0,0,0]

2 → [0,0,1,0,0,0,0,0,0,0]

3 → [0,0,0,1,0,0,0,0,0,0]

⋮

AN UNTRAINED MODEL

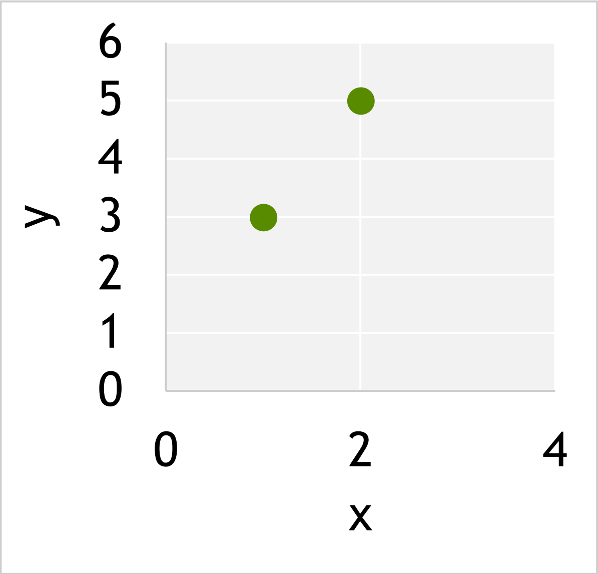


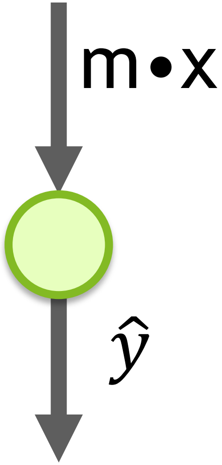
A SIMPLER MODEL

A SIMPLER MODEL

$$y = mx + b$$

x	y
1	3
2	5





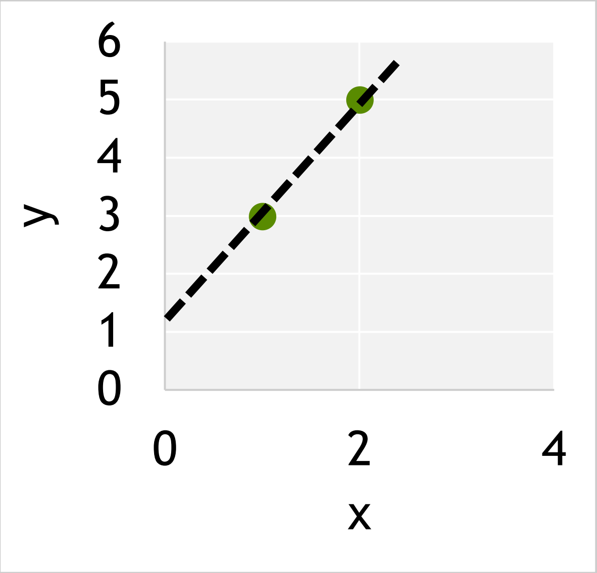
$m = ?$

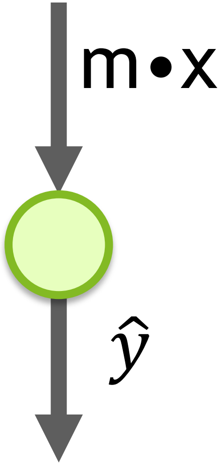
$b = ?$

A SIMPLER MODEL

$$y = mx + b$$

x	y
1	3
2	5





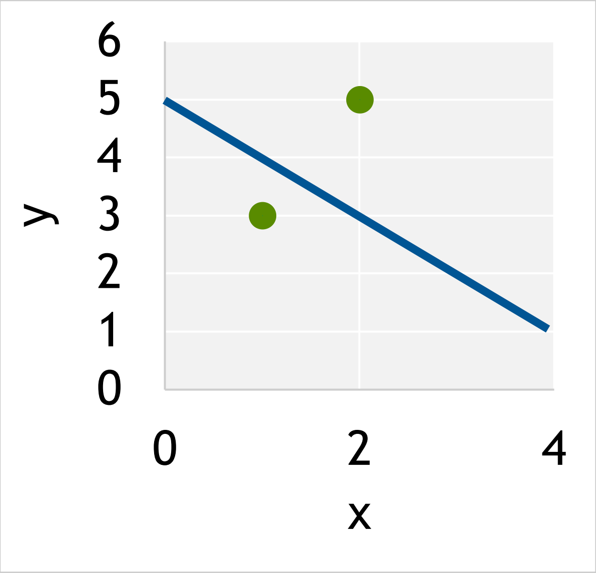
$$m = ?$$

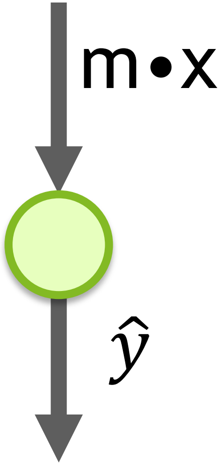
$$\mathrm{b} = ?$$

A SIMPLER MODEL

$$y = mx + b$$

x	y	$\hat{y}$
1	3	4
2	5	3





Start Random

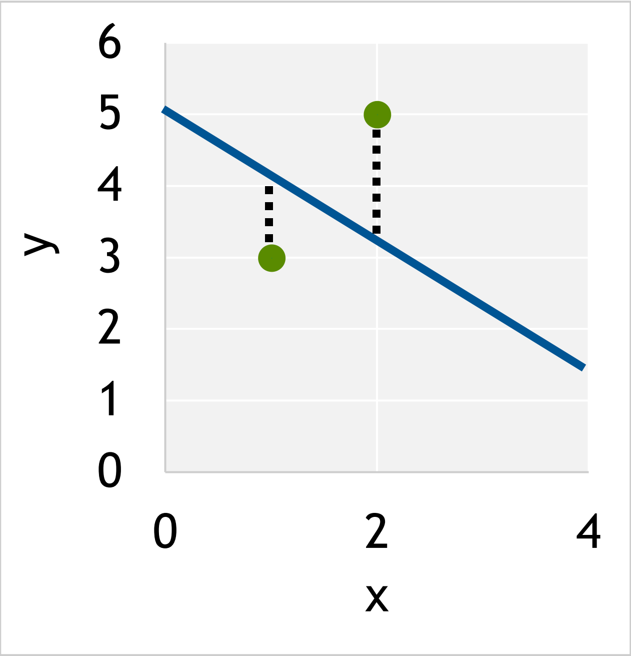
$m = -1$

$b = 5$

A SIMPLER MODEL

$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4
		MSE =	2.5
		RMSE =	1.6



$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

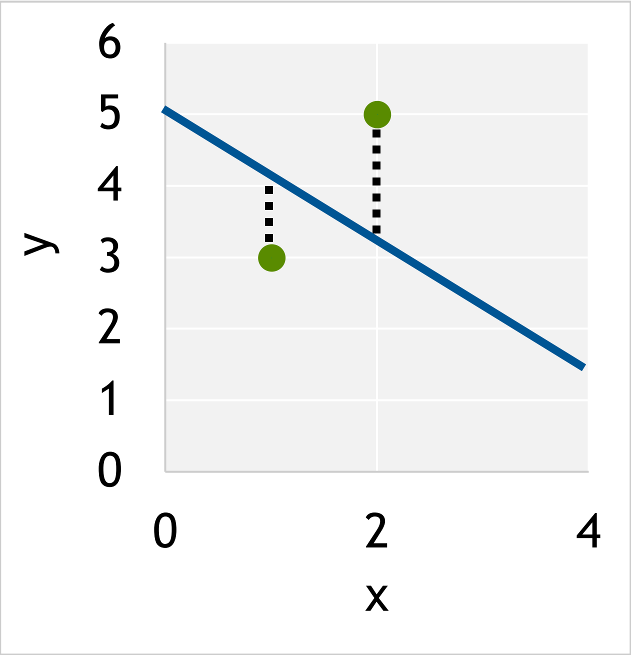
A SIMPLER MODEL

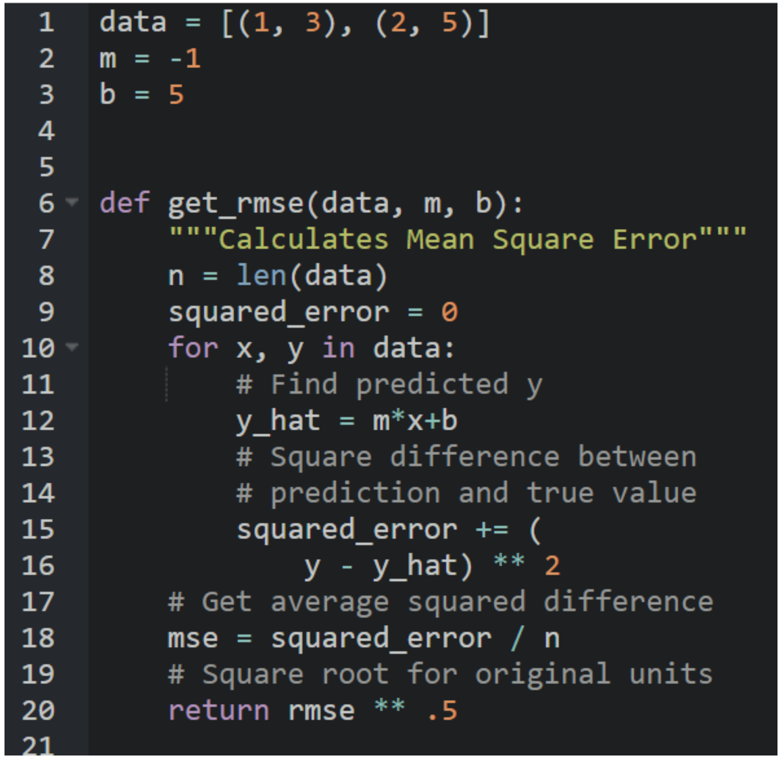
$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4

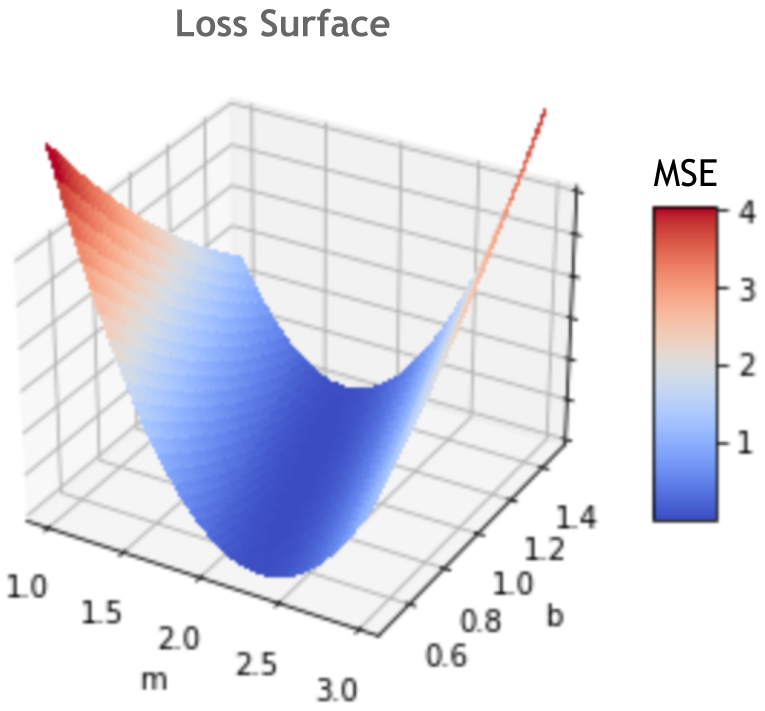
MSE = 2.5

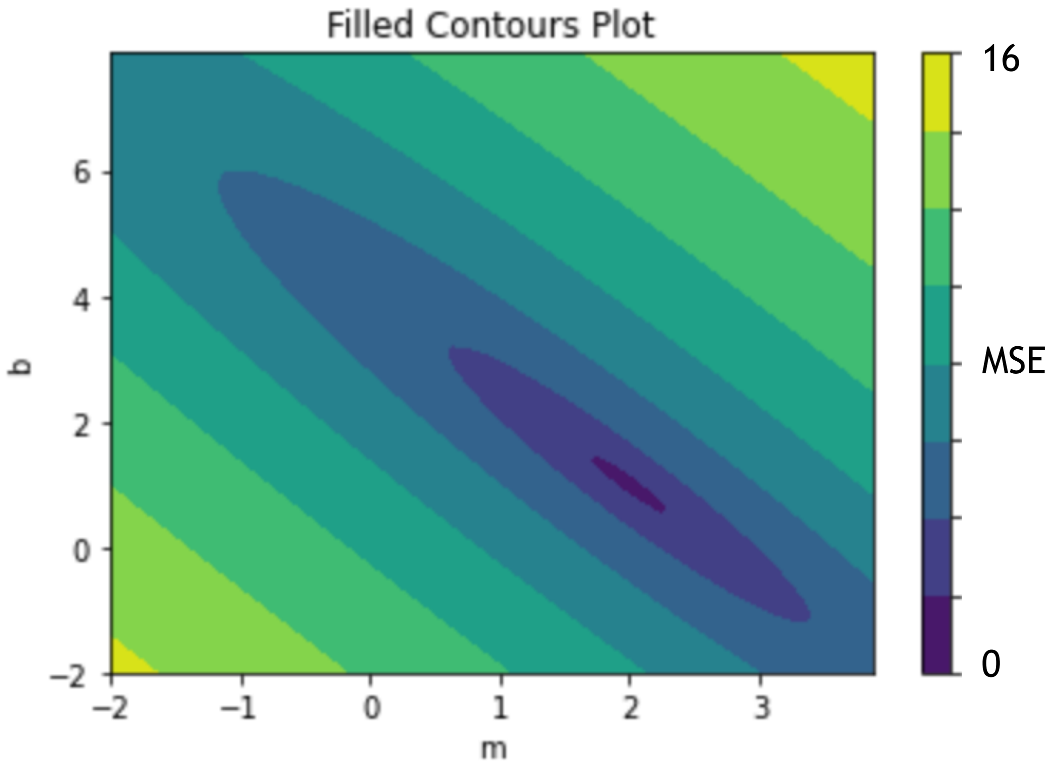
RMSE = 1.6



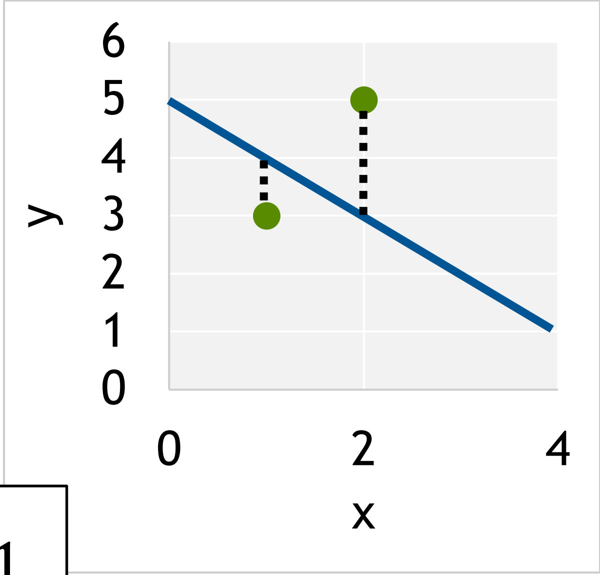

```
data = [(1, 3), (2, 5)]
m = -1
b = 5


def get_rmse(data, m, b):
    """Calculates Mean Square Error"""
    n = len(data)
    squared_error = 0
    for x, y in data:
        # Find predicted y
        y_hat = m*x+b
        # Square difference between
        # prediction and true value
        squared_error += (
            y - y_hat) ** 2
    # Get average squared difference
    mse = squared_error / n
    # Square root for original units
    return rmse ** .5

```

THE LOSS CURVE



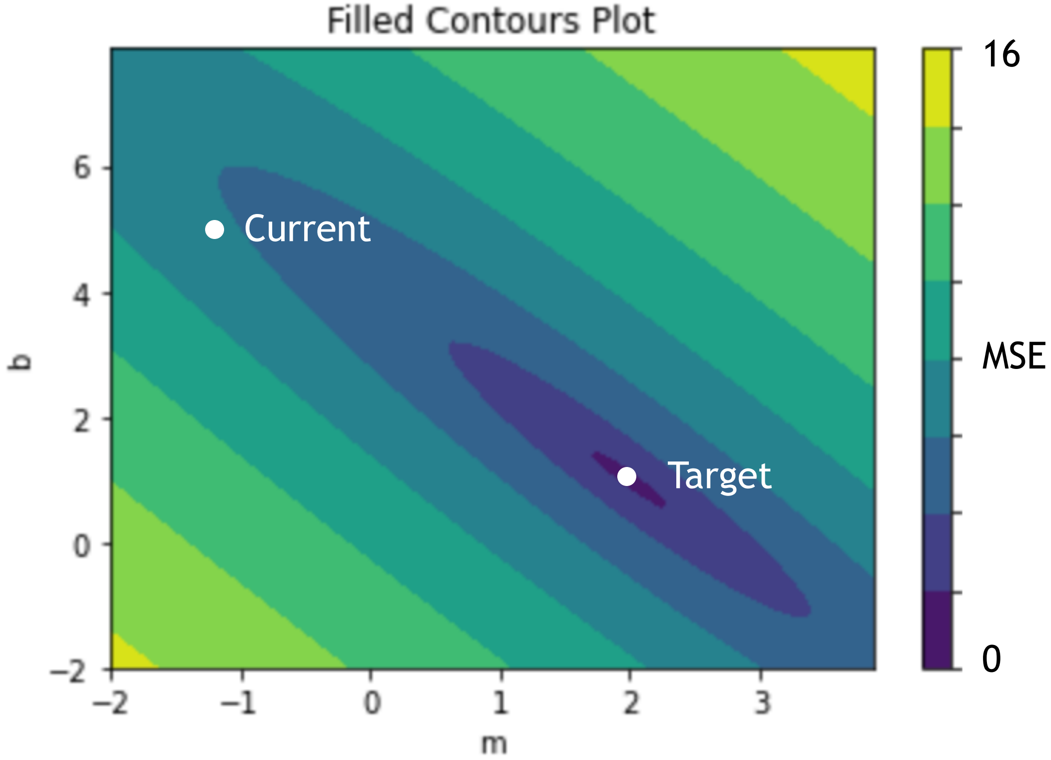


THE LOSS CURVE

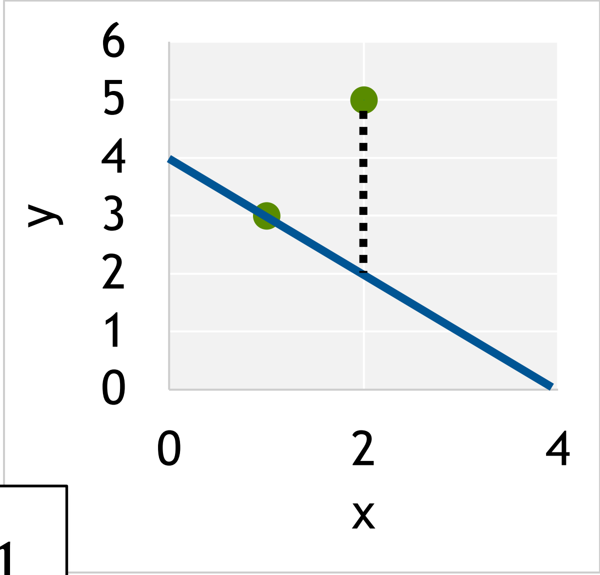


$$m = -1$$

$$b = 5$$

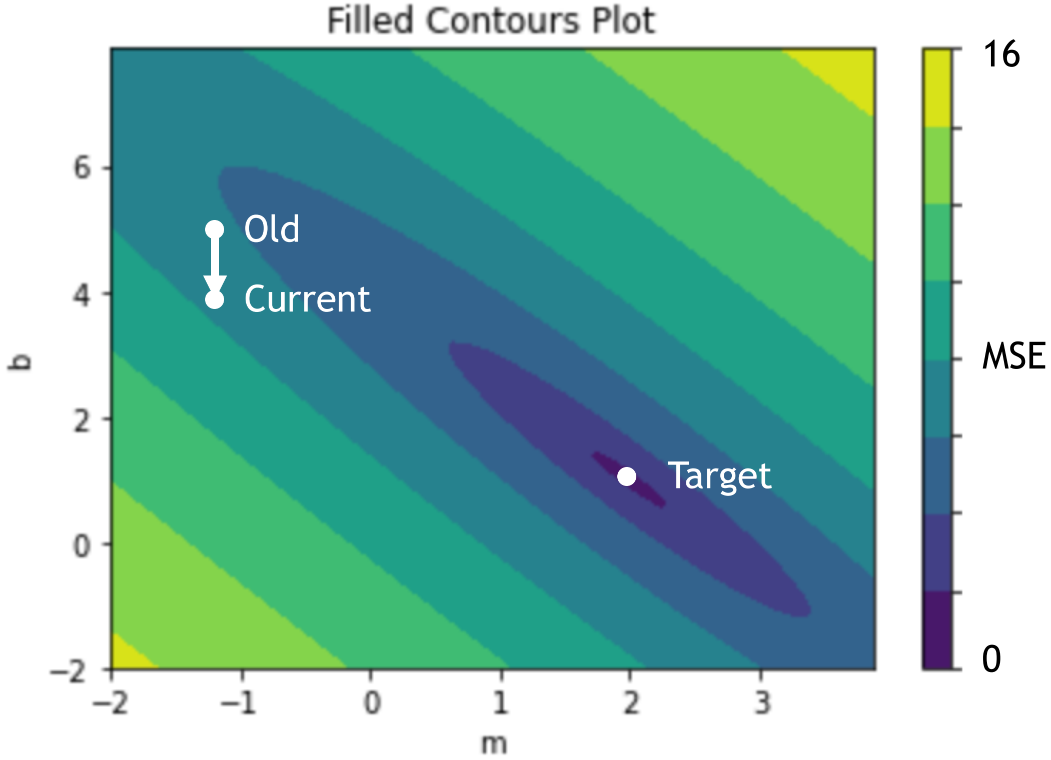


THE LOSS CURVE

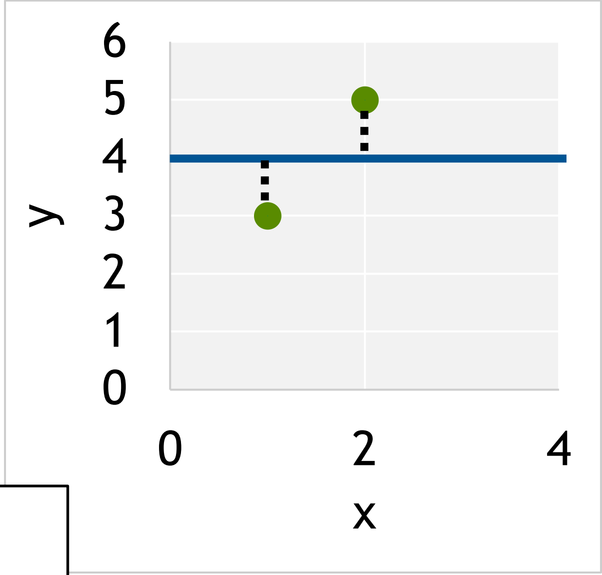


$m = -1$

$b = 4$

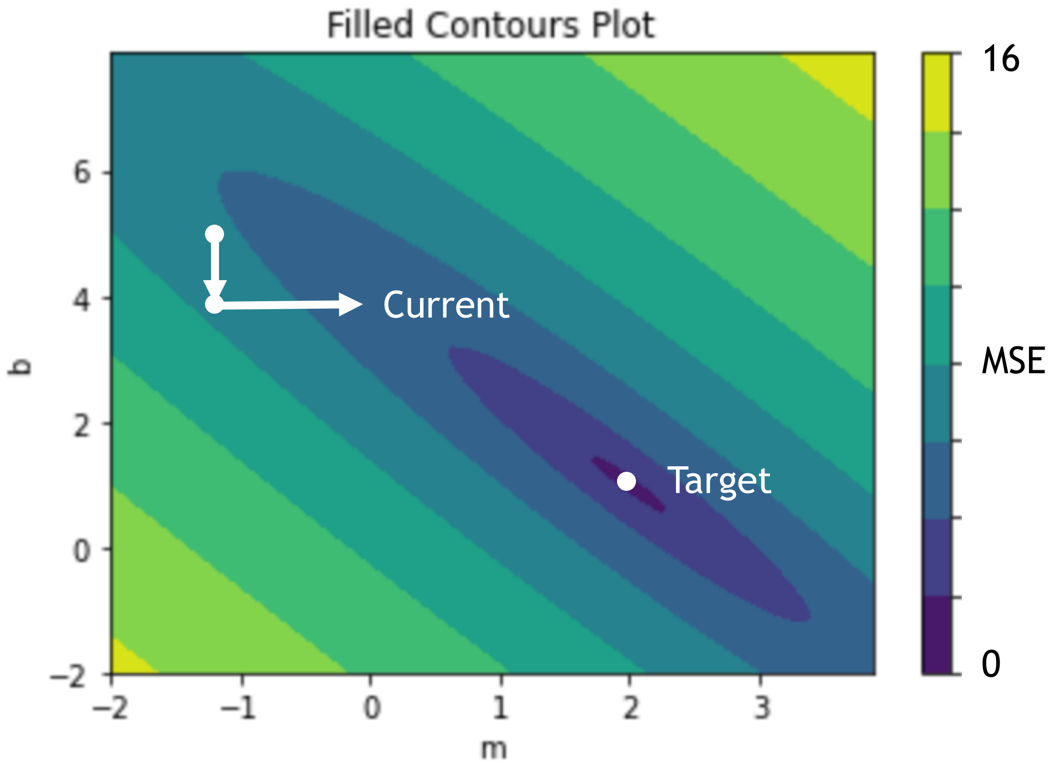


THE LOSS CURVE



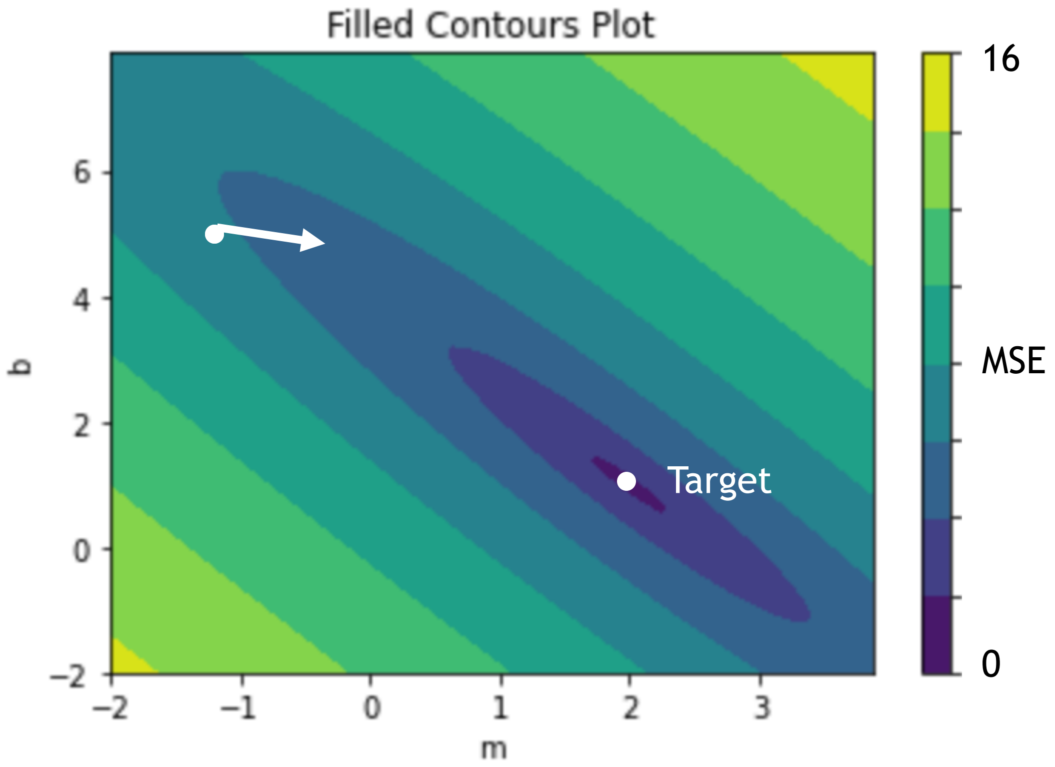
$m = 0$

$\mathrm{b} = 4$



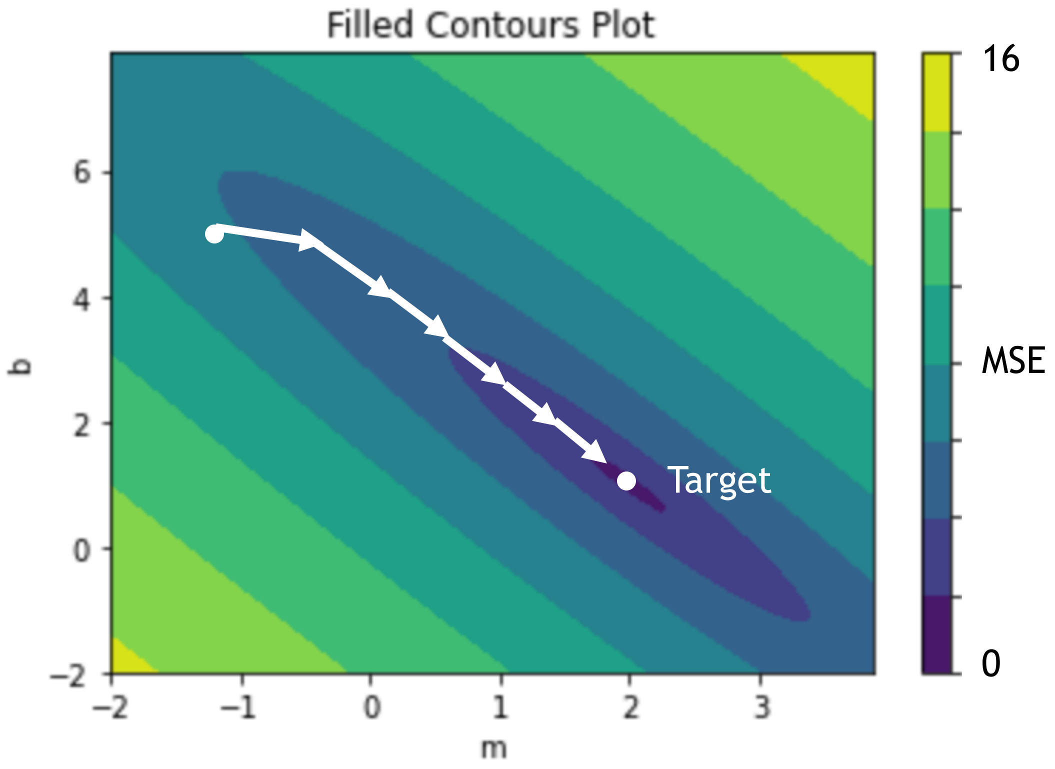
THE LOSS CURVE

The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters

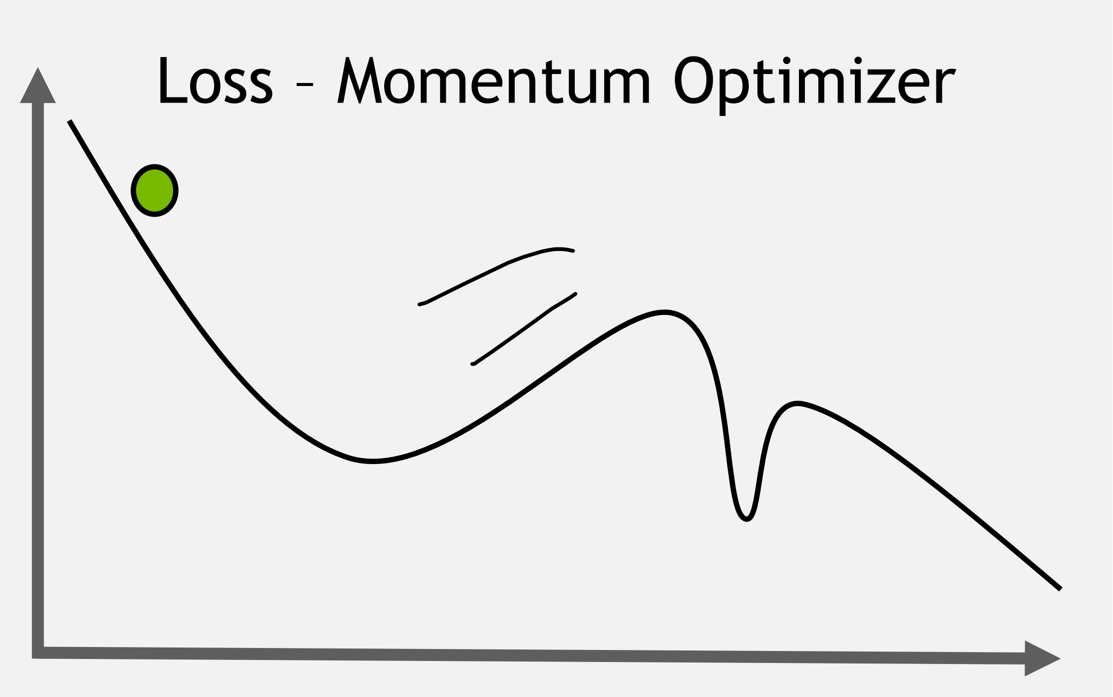


THE LOSS CURVE

The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters



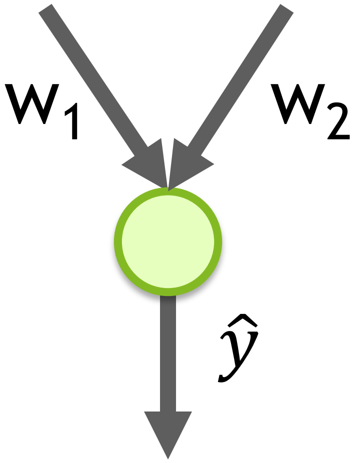
OPTIMIZERS



- Adam
- Adagrad
- RMSprop
- SGD

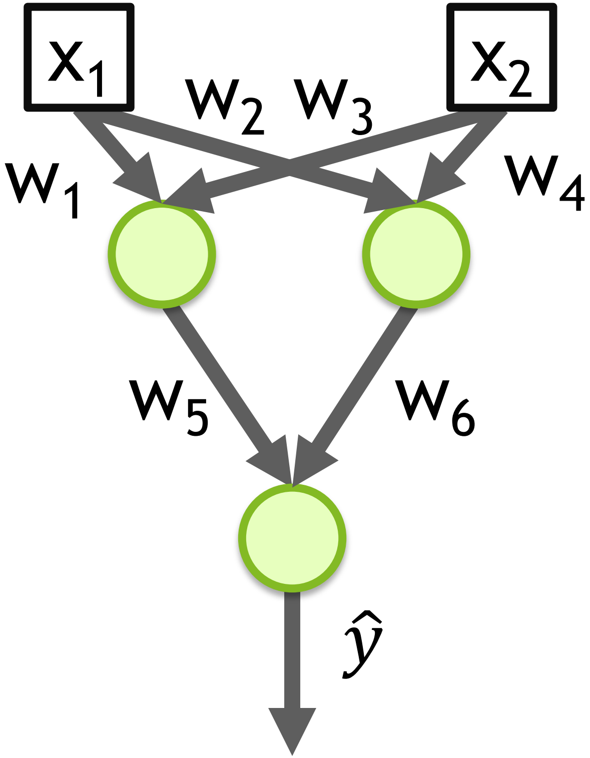
FROM NEURON TO NETWORK

BUILDING A NETWORK



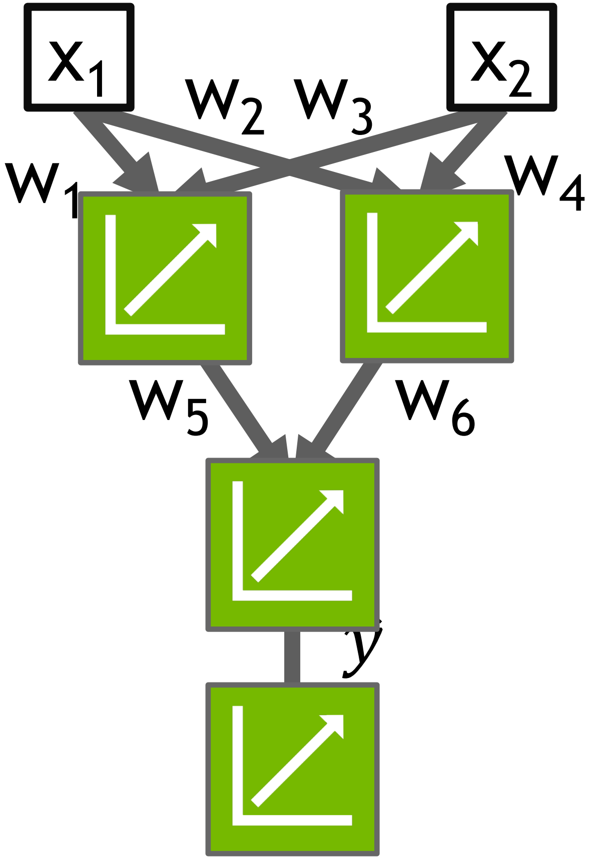
- Scales to more inputs

BUILDING A NETWORK



- Scales to more inputs
- Can chain neurons

BUILDING A NETWORK



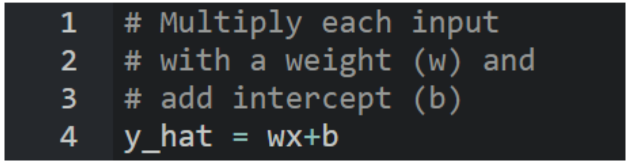
- Scales to more inputs
- Can chain neurons
- If all regressions are linear, then output will also be a linear regression

ACTIVATION FUNCTIONS

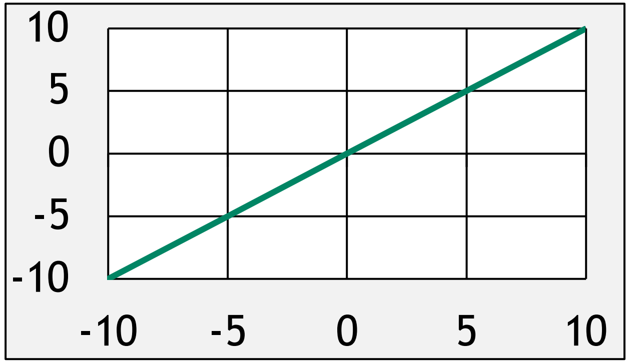
ACTIVATION FUNCTIONS

Linear

$$\hat{y} = wx + b$$

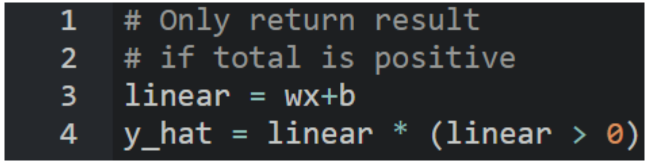


```
# Multiply each input
# with a weight (w) and
# add intercept (b)
y_hat = wx+b
```

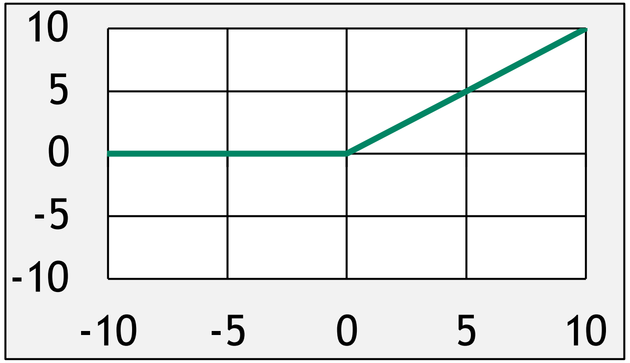


ReLU

$$\hat{y} = \begin{cases} wx + b \; if \; wx + b > 0 \\ 0 \;\; otherwise \end{cases}$$

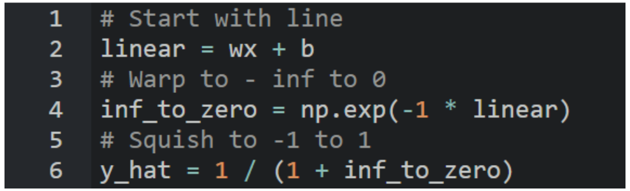


```
# Only return result
# if total is positive
linear = wx+b
y_hat = linear * (linear > 0)
```

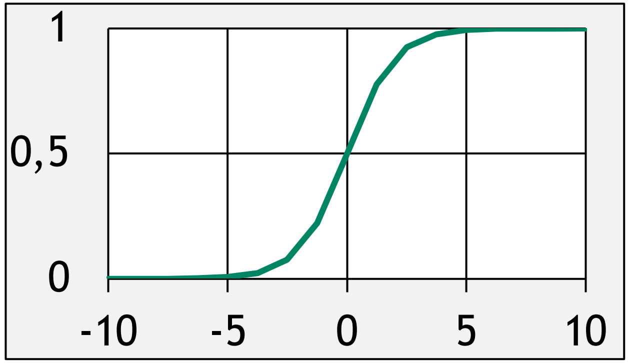


Sigmoid

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

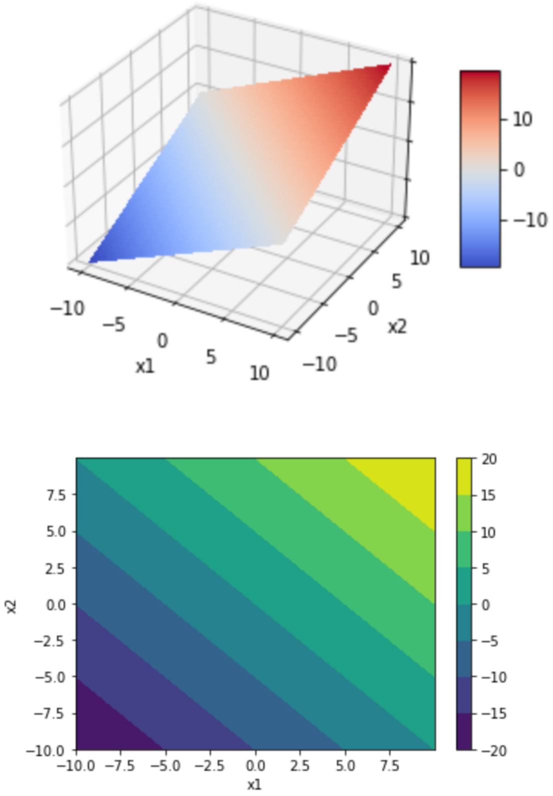


```
# Start with line
linear = wx + b
# Warp to - inf to 0
inf_to_zero = np.exp(-1 * linear)
# Squish to -1 to 1
y_hat = 1 / (1 + inf_to_zero)
```

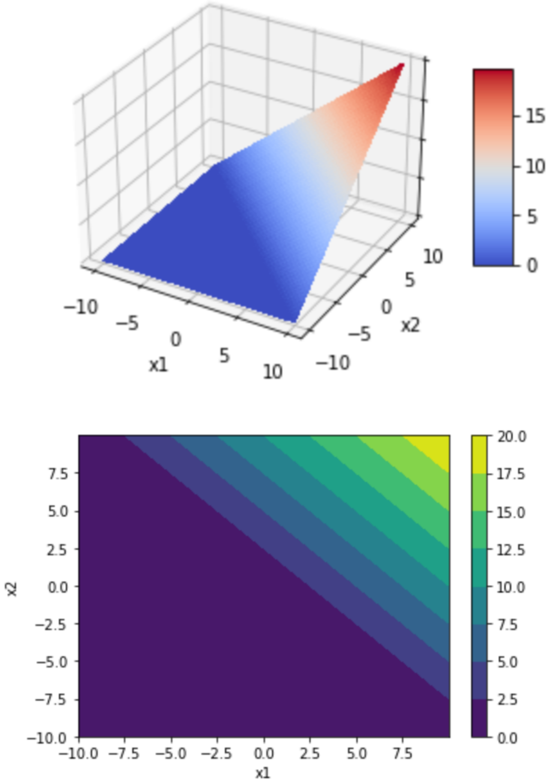


ACTIVATION FUNCTIONS

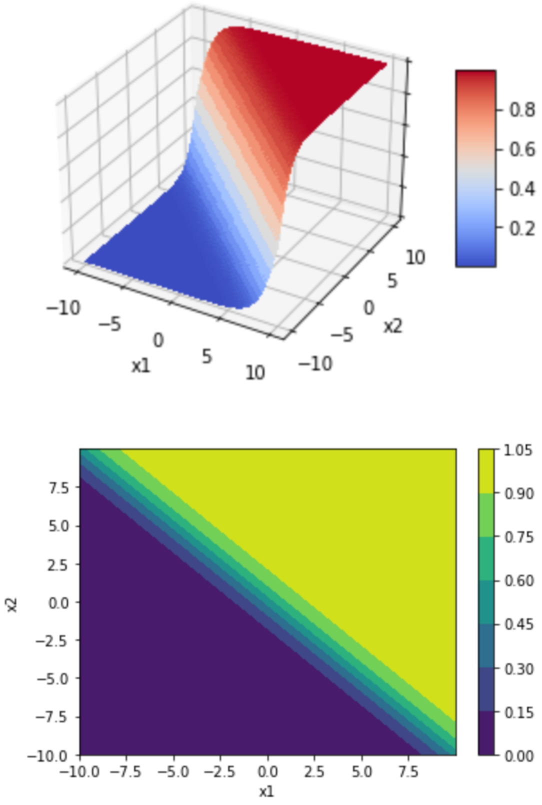
Linear



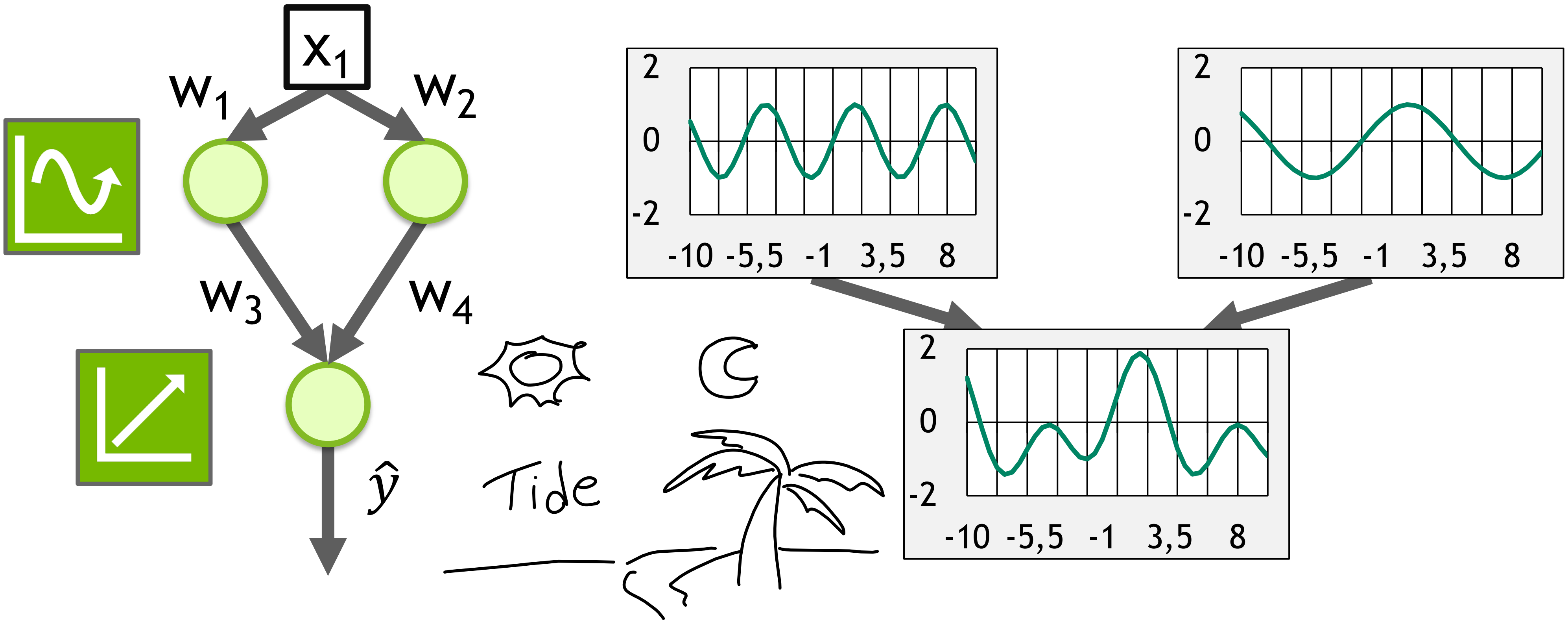
ReLU



Sigmoid



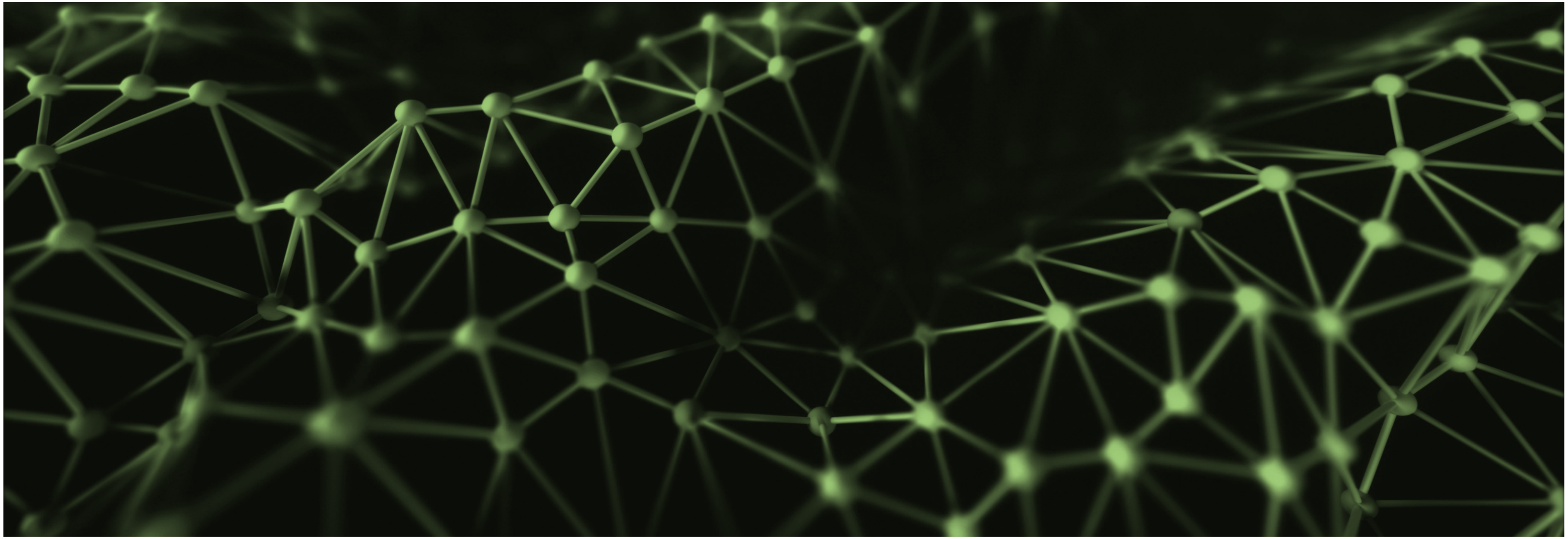
ACTIVATION FUNCTIONS



OVERFITTING

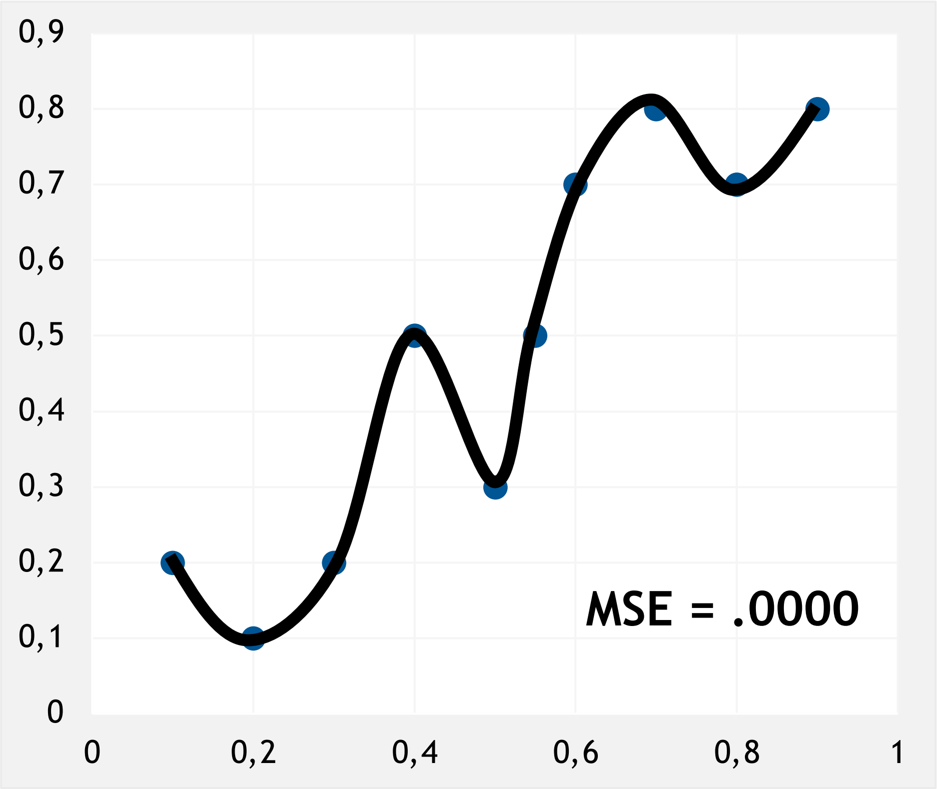
OVERFITTING

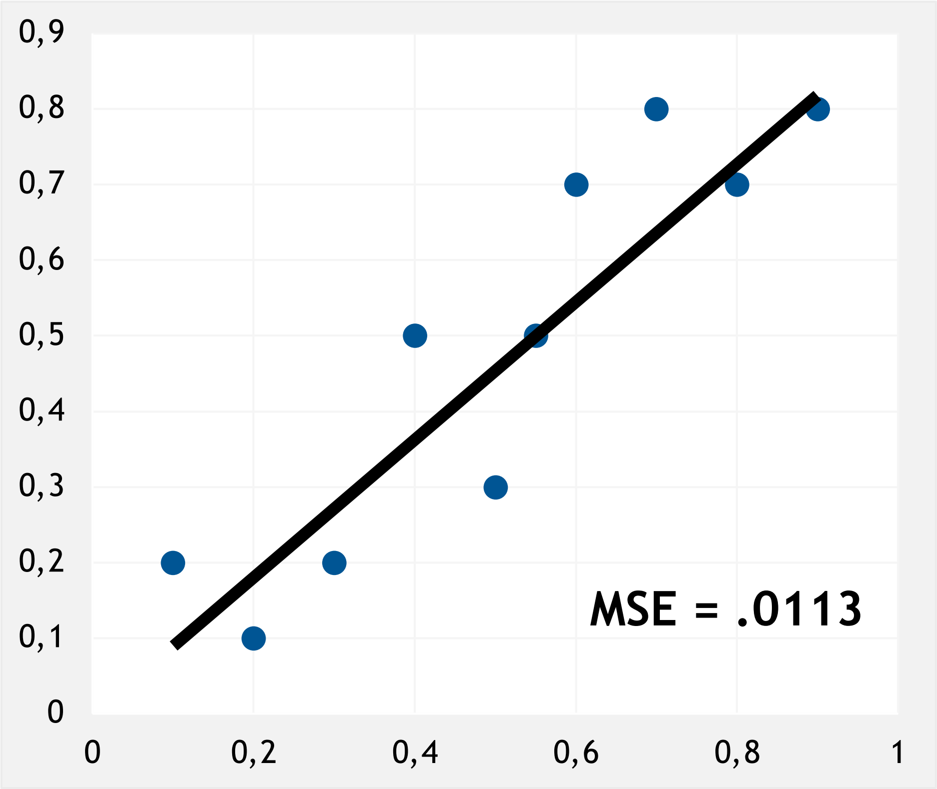
Why not have a super large neural network?



OVERFITTING

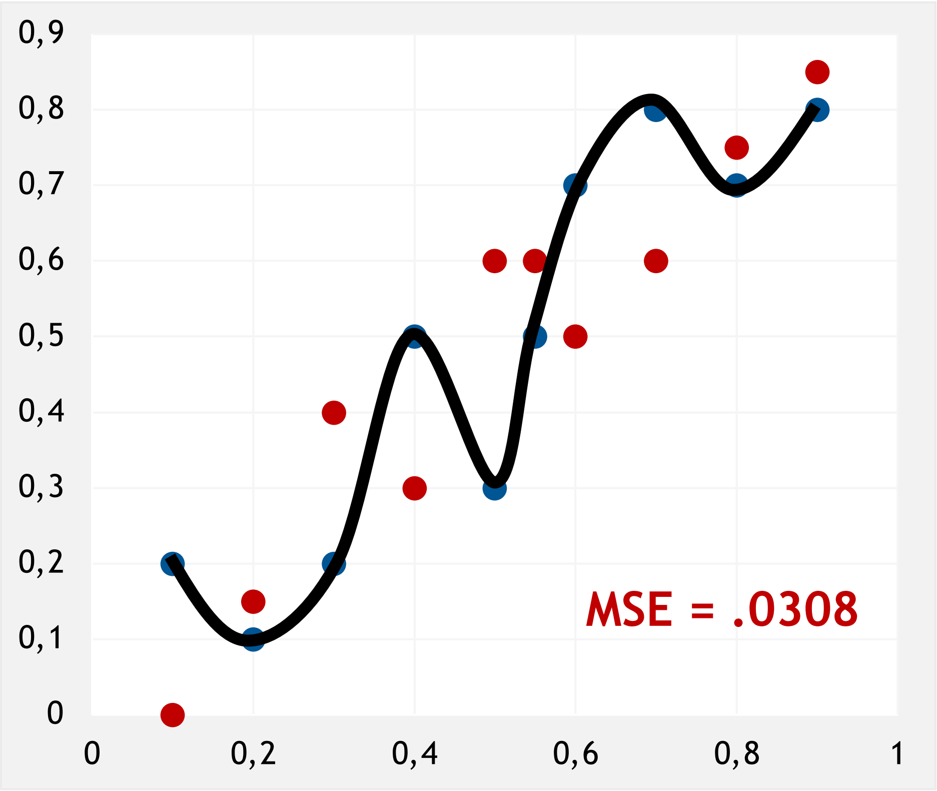
Which Trendline is Better?

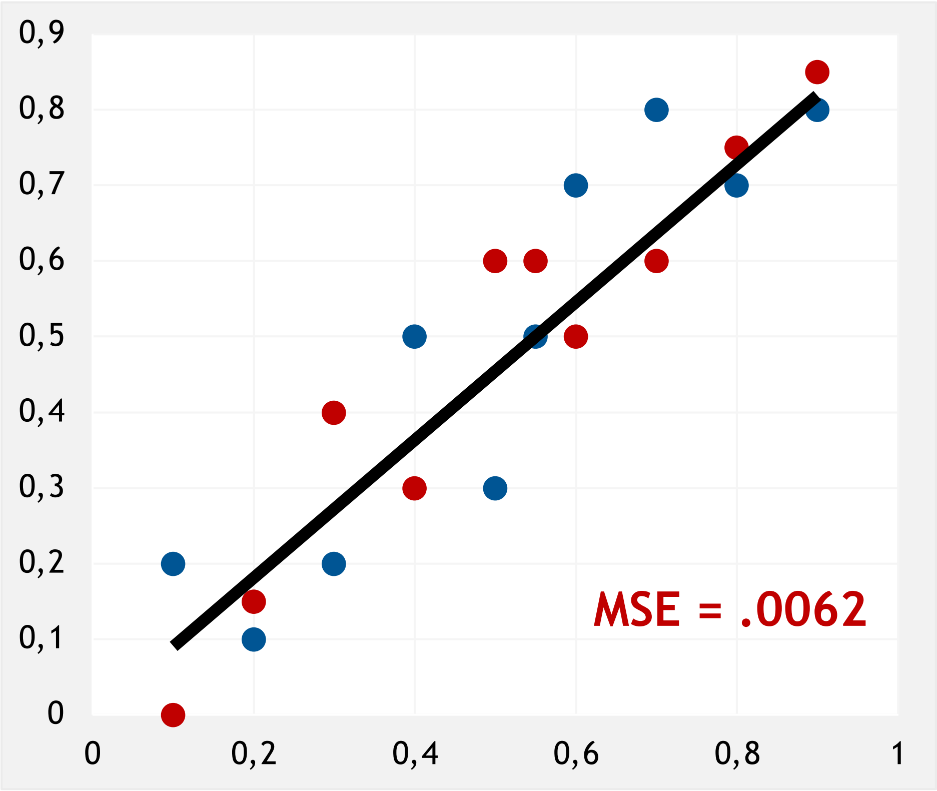




OVERFITTING

Which Trendline is Better?





TRAINING VS VALIDATION DATA

Avoid memorization

Training data

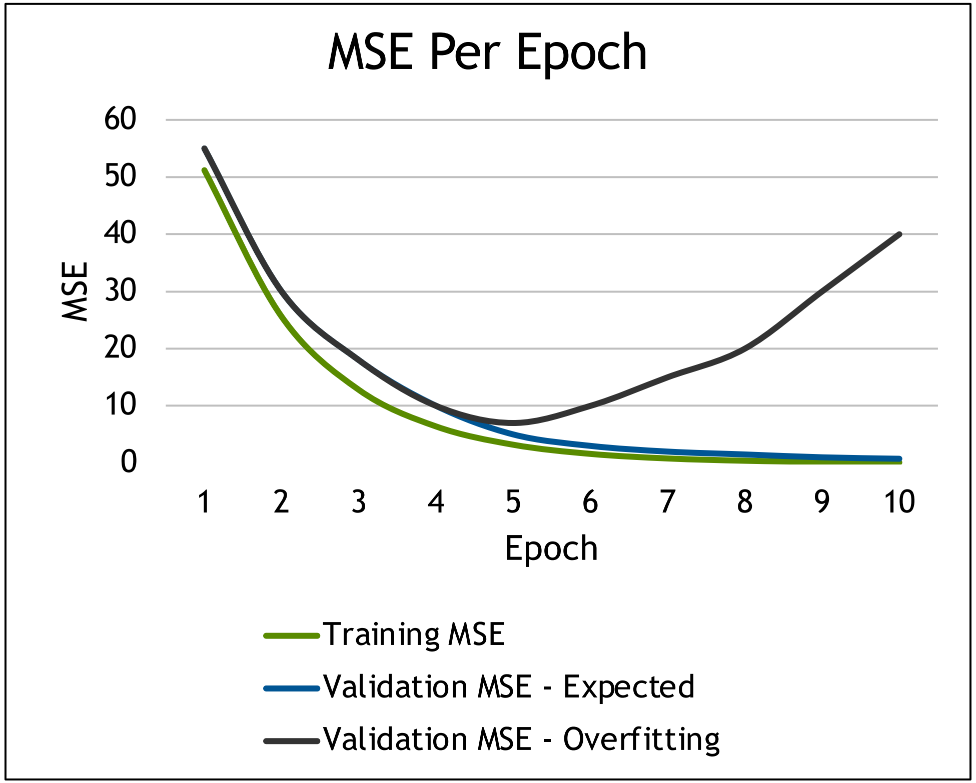
- Core dataset for the model to learn on

Validation data

- New data for model to see if it truly understands (can generalize)

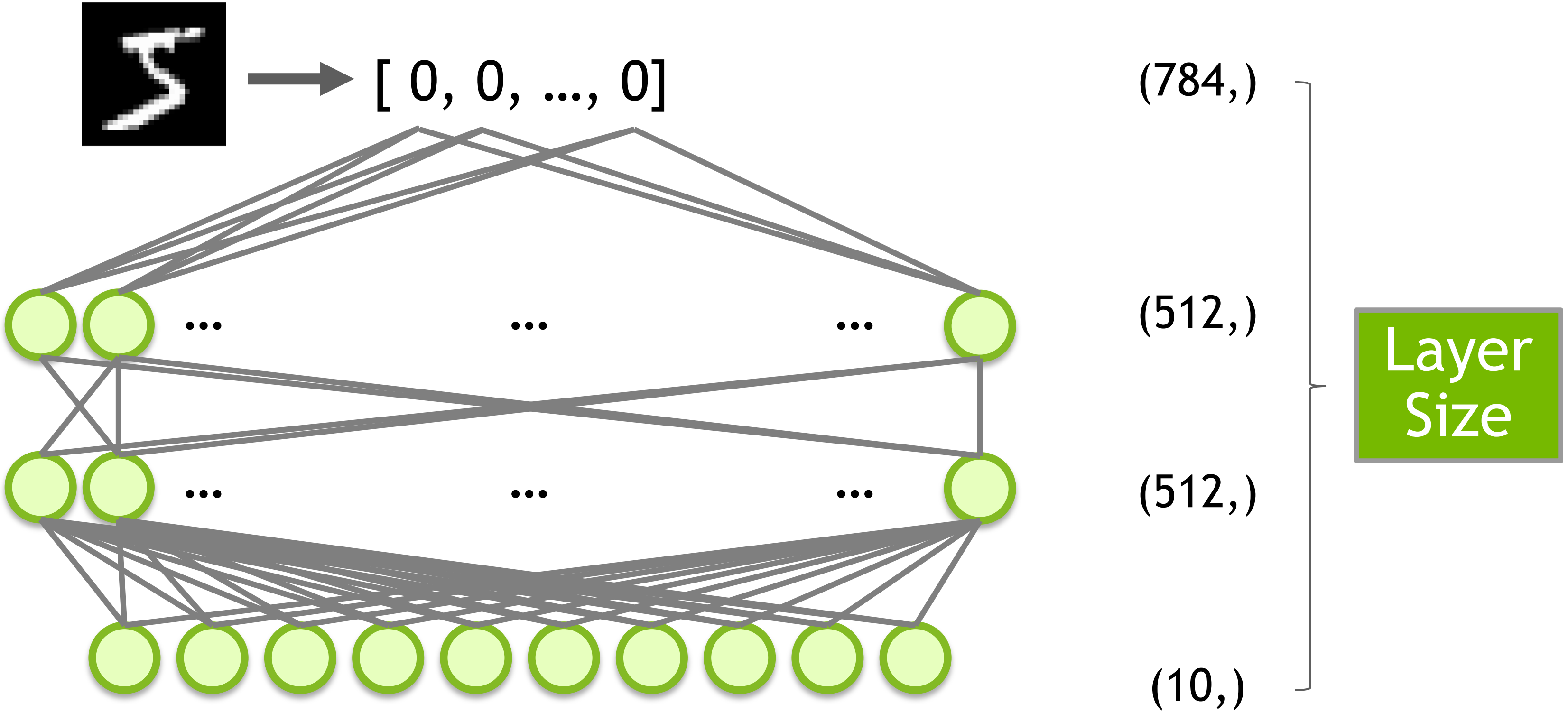
Overfitting

- When model performs well on the training data, but not the validation data (evidence of memorization)
- Ideally the accuracy and loss should be similar between both datasets

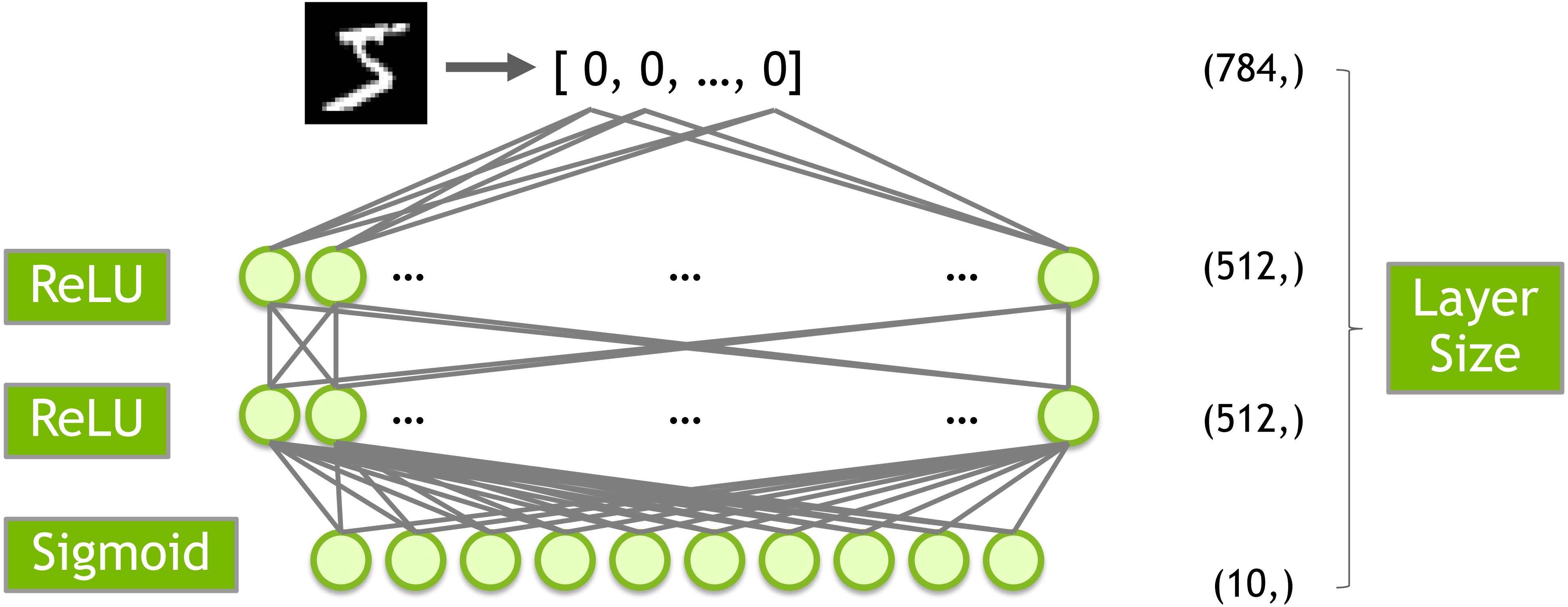


FROM REGRESSION TO CLASSIFICATION

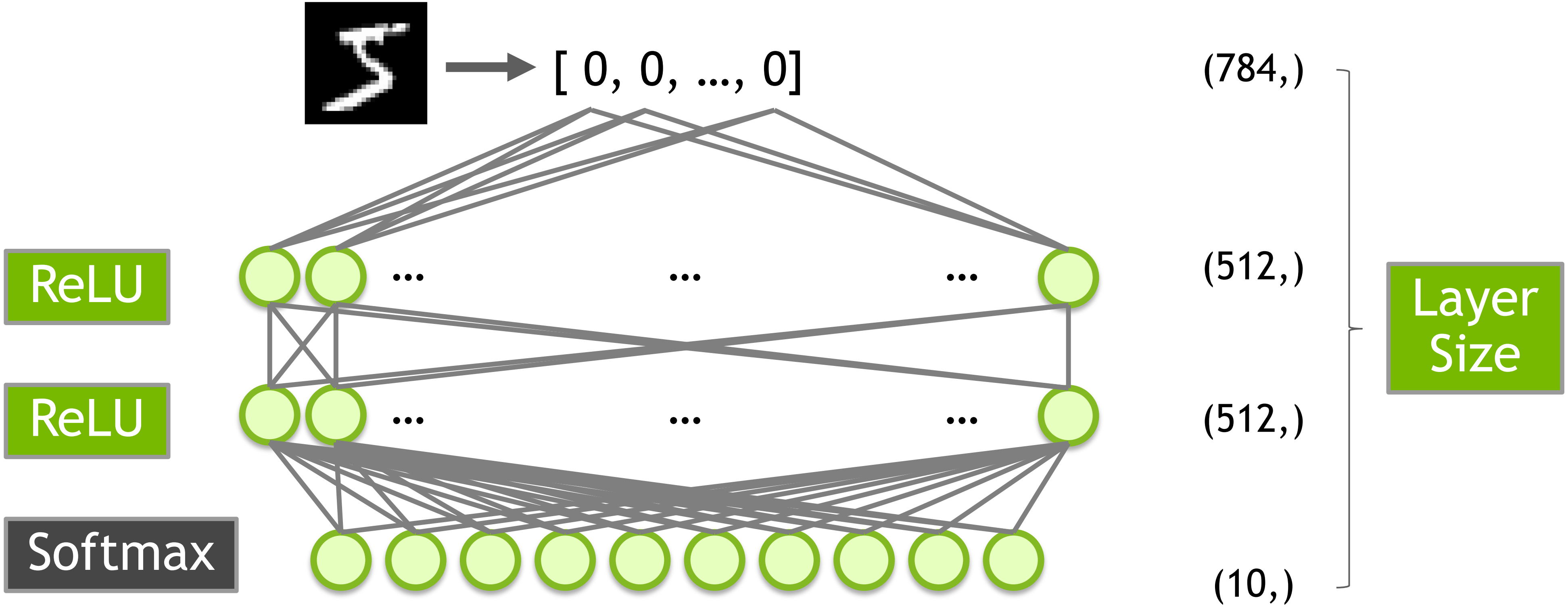
AN MNIST MODEL



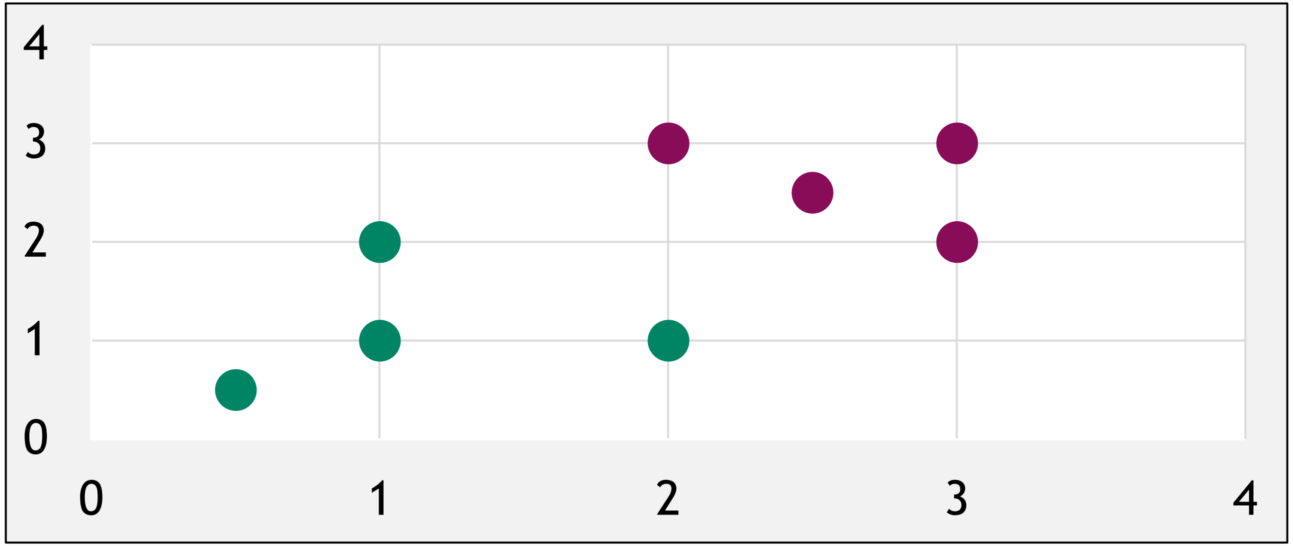
AN MNIST MODEL



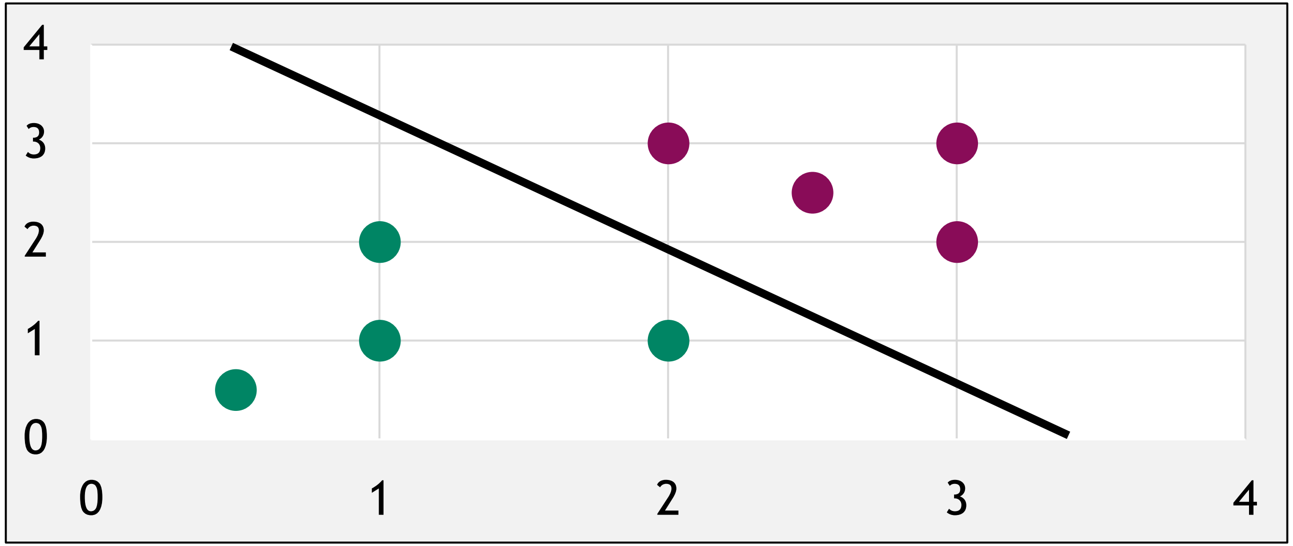
AN MNIST MODEL



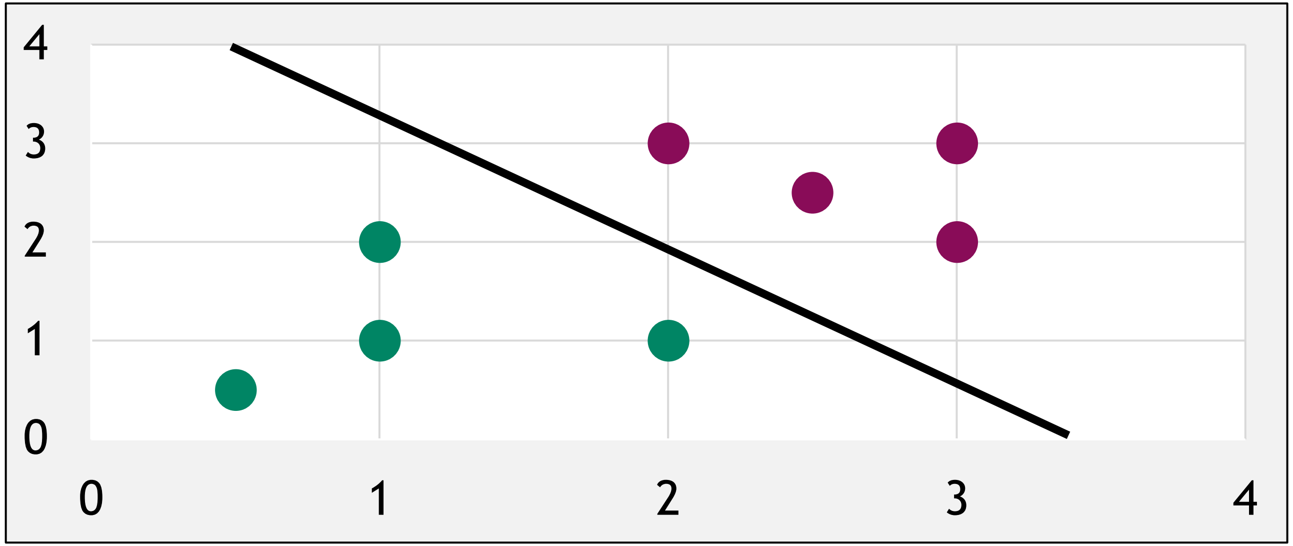
RMSE FOR PROBABILITIES?

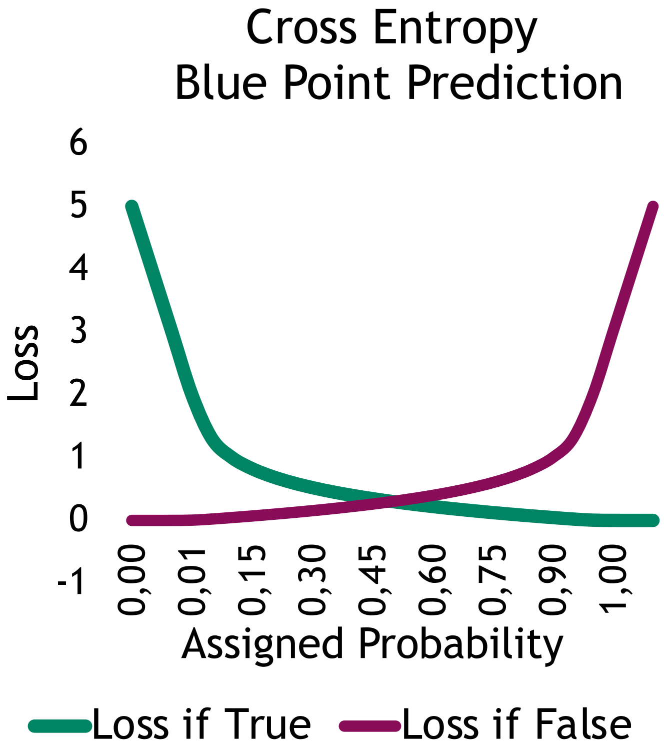


RMSE FOR PROBABILITIES?

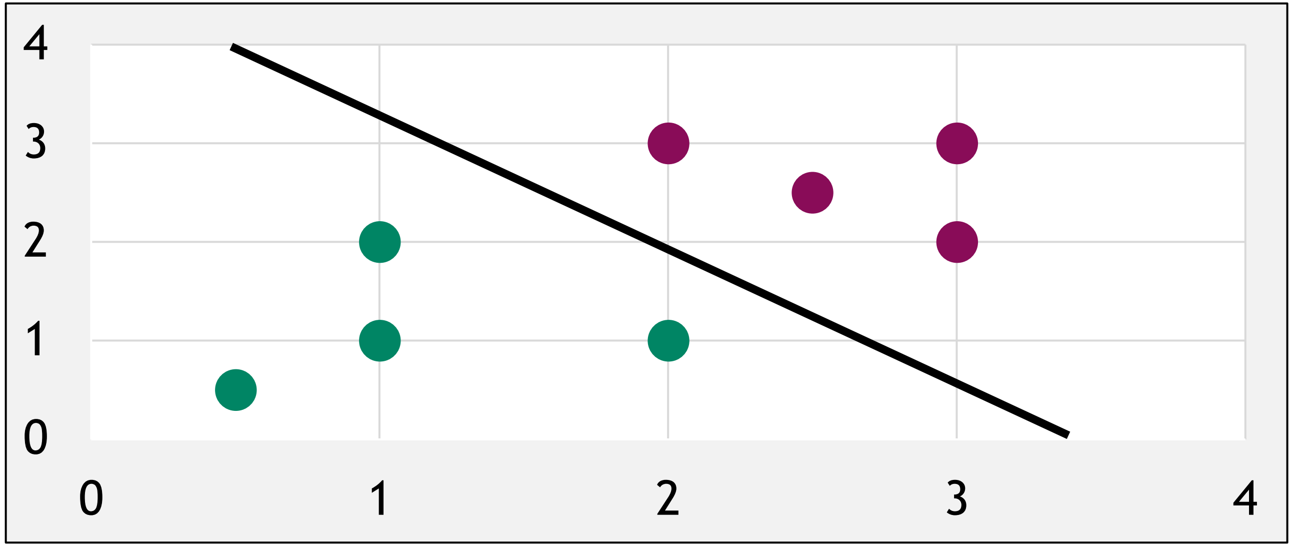


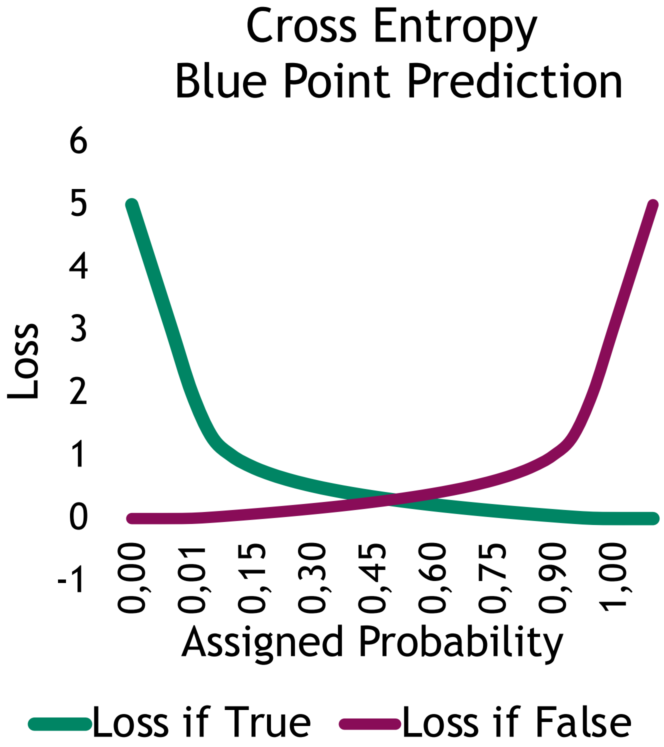
CROSS ENTROPY





CROSS ENTROPY



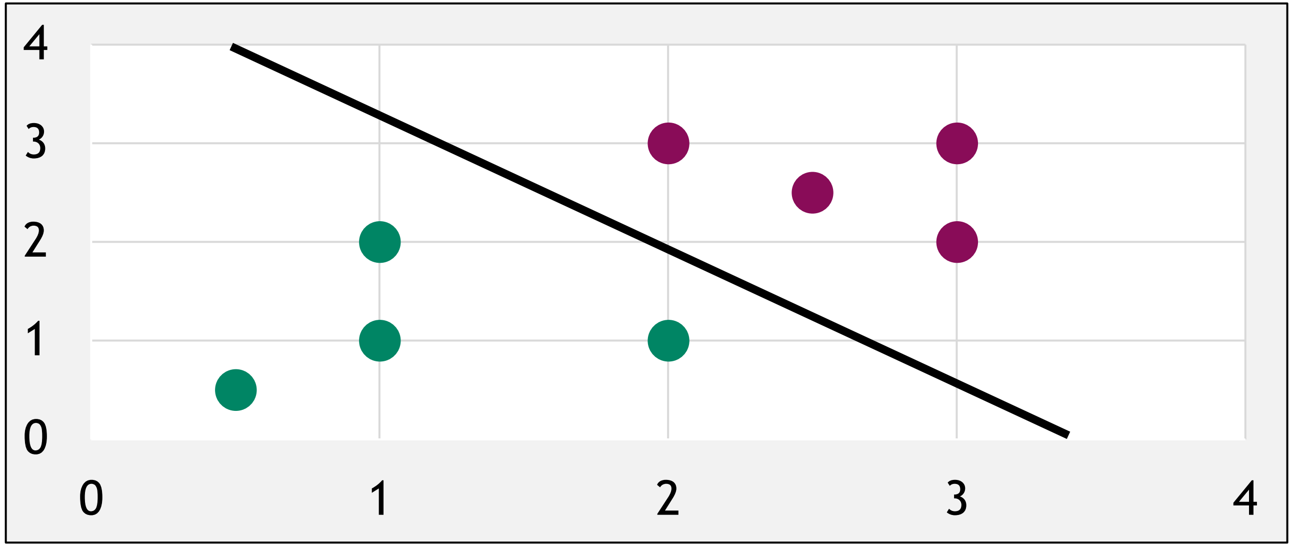


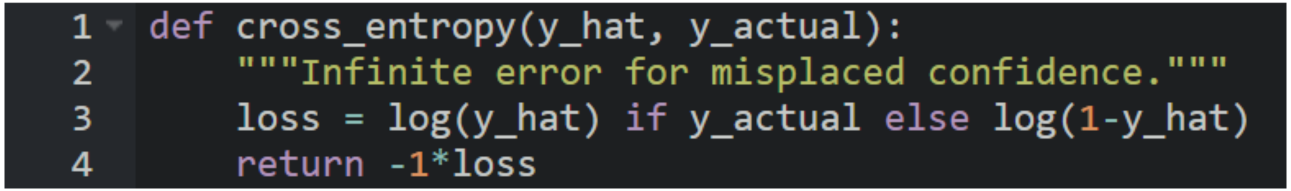
$$Loss = -(t(x) \cdot \log(p(x) + \left(1 - t(x)\right) \cdot \log(1 - p(x)))$$

$$t(x) = target\ (0\ if\ False, 1\ if\ True)$$

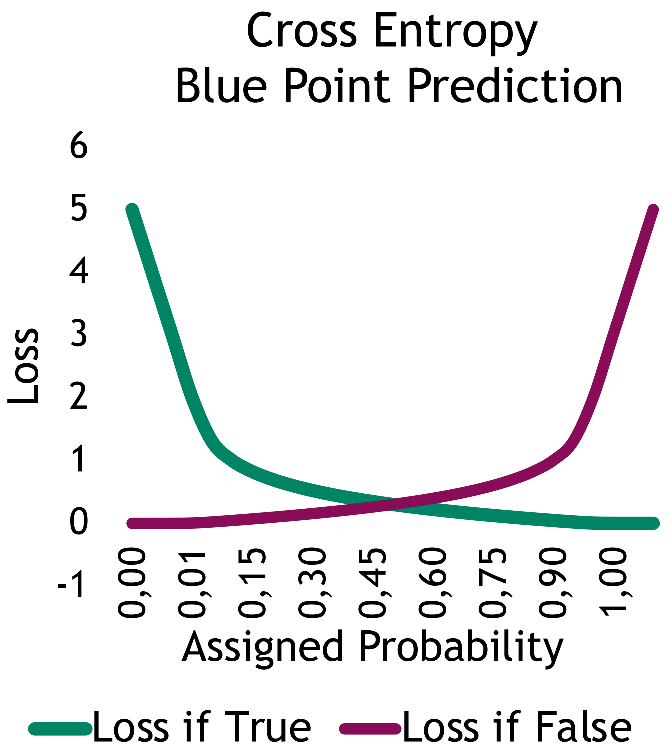
$$p(x) = probability\ prediction\ of\ point\ x$$

CROSS ENTROPY





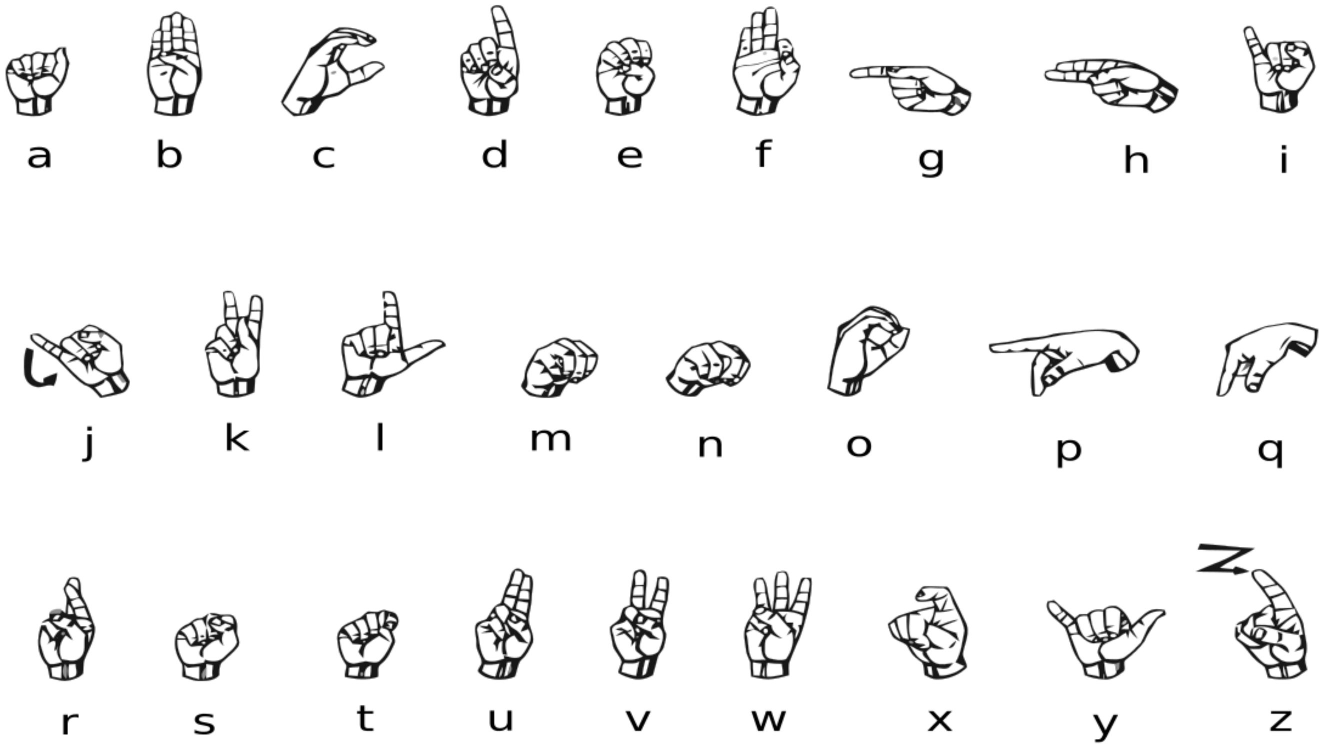
```python
def cross_entropy(y_hat, y_actual):
    """Infinite error for misplaced confidence."""
    loss = log(y_hat) if y_actual else log(1-y_hat)
    return -1*loss
```



BRINGING IT TOGETHER

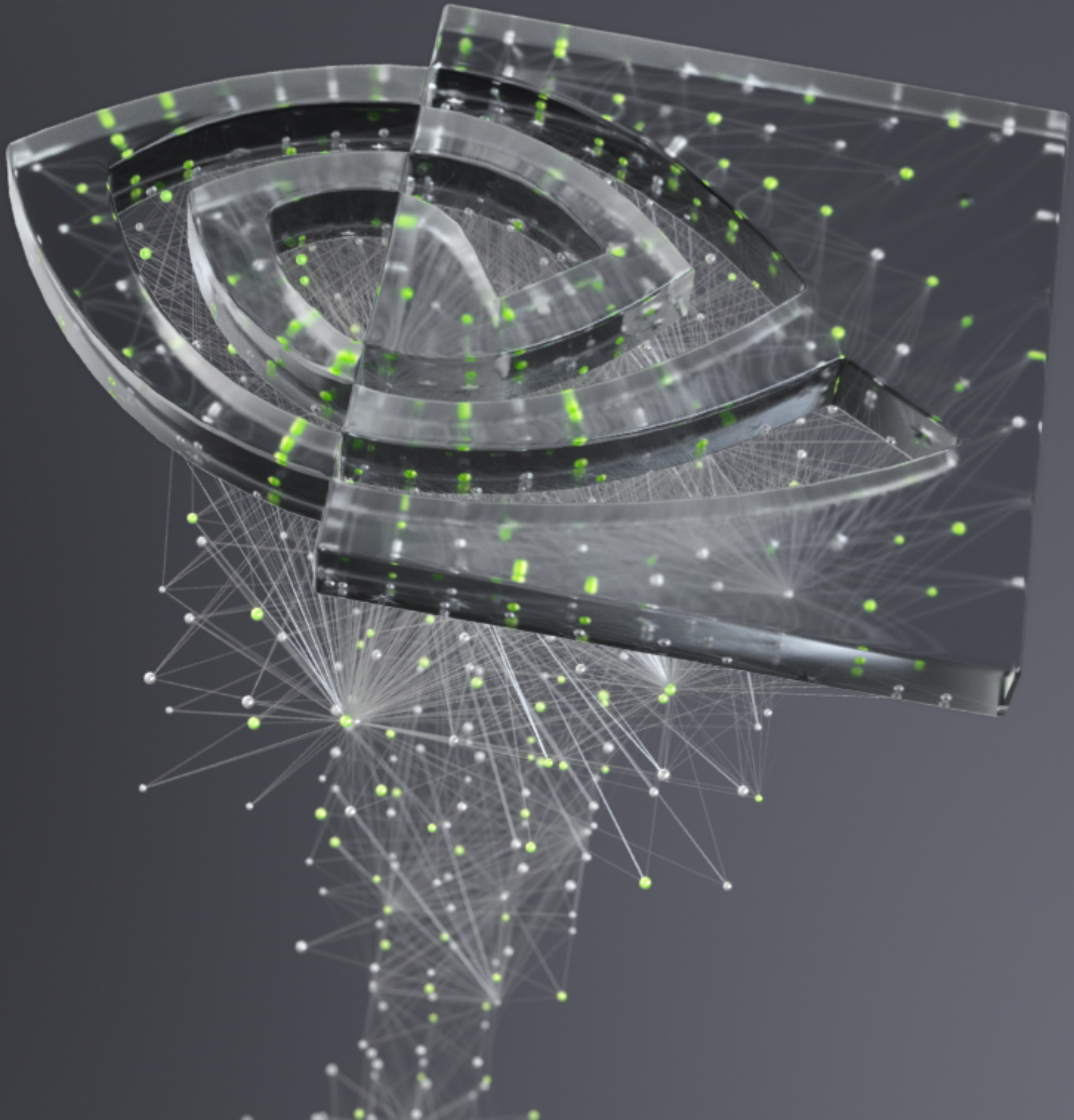
THE NEXT EXERCISE

The American Sign Language Alphabet



LET’S GO!

NVIDIA | DEEP LEARNING INSTITUTE



APPENDIX: GRADIENT DESCENT

HELPING THE COMPUTER CHEAT CALCULUS

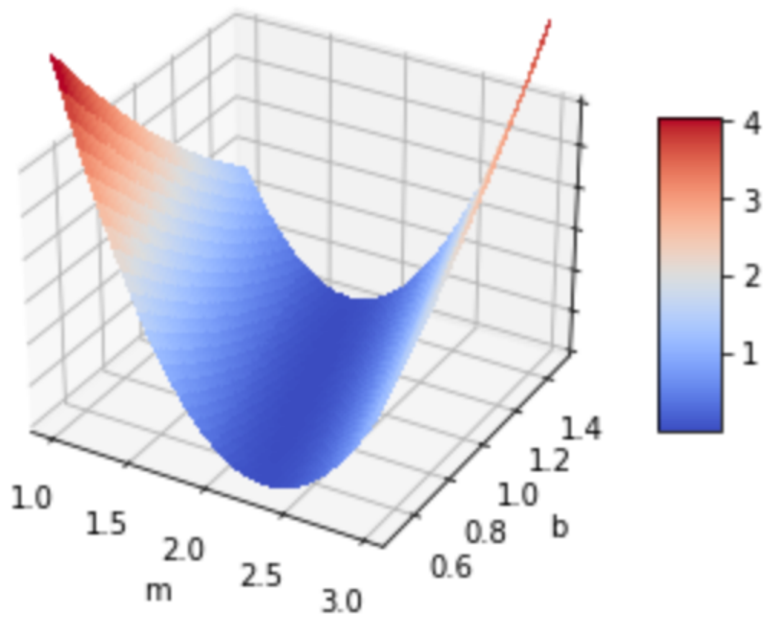
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 = \frac{1}{n}\sum_{i=1}^{n}(y - (mx + b))^2$$

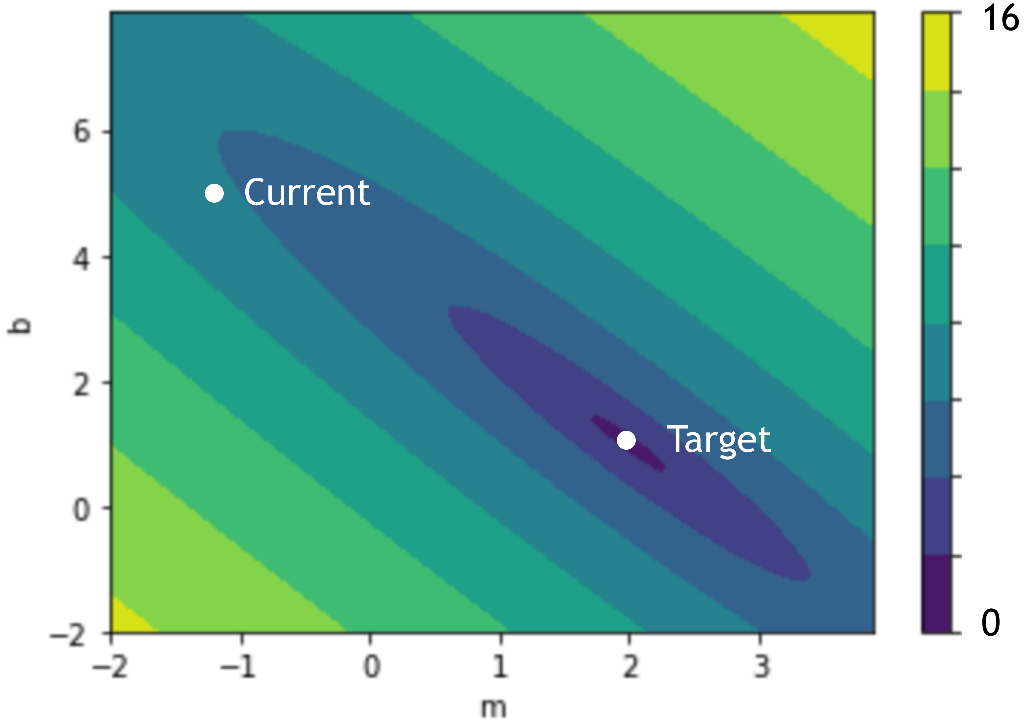
$$MSE = \frac{1}{2}((3 - (m(1) + b))^2 + (5 - (m(2) + b))^2)$$

$$\frac{\partial MSE}{\partial m} = 9m + 5b - 23 \qquad \frac{\partial MSE}{\partial b} = 5m + 3b - 13$$

$$\frac{\partial MSE}{\partial m} = -7 \qquad \frac{\partial MSE}{\partial b} = -3$$

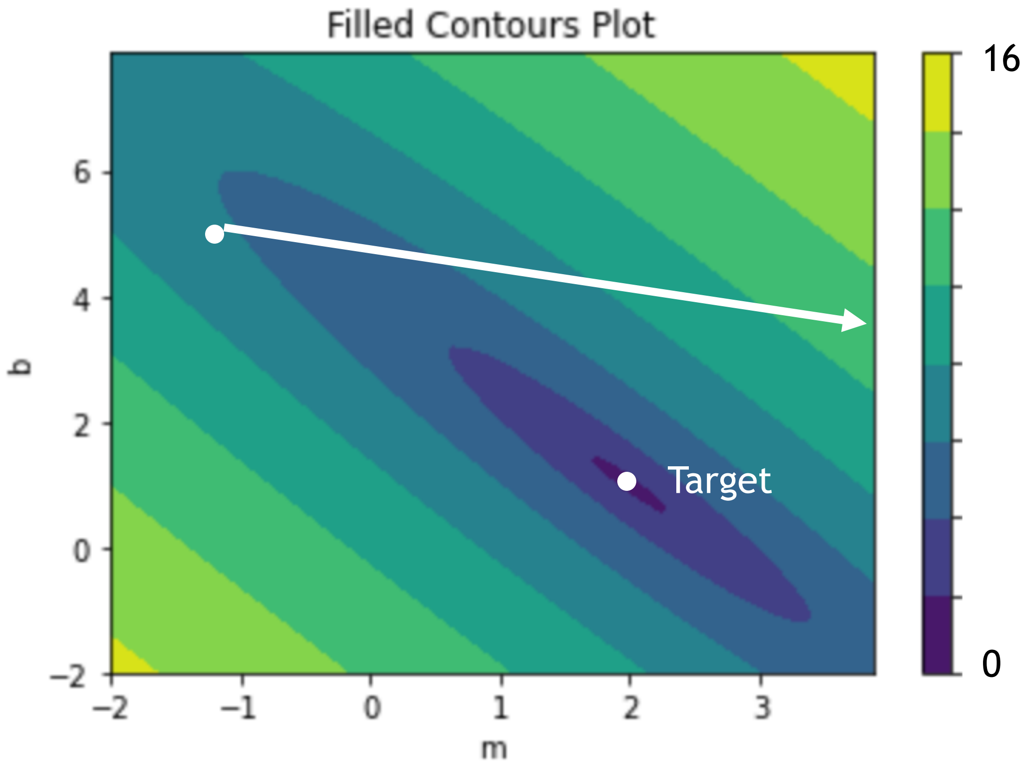
THE LOSS CURVE


Loss Surface


Filled Contours Plot

THE LOSS CURVE

$$\frac{\partial MSE}{\partial m} = -7 \qquad \frac{\partial MSE}{\partial b} = -3$$

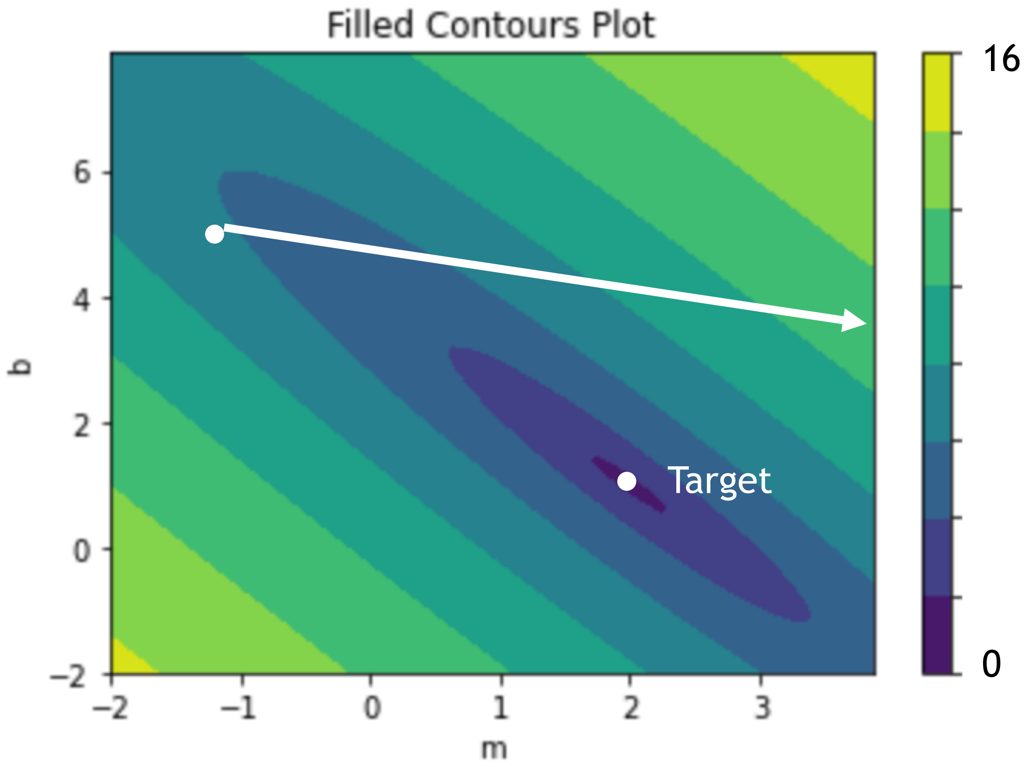


THE LOSS CURVE

$$\frac{\partial MSE}{\partial m} = -7 \qquad \frac{\partial MSE}{\partial b} = -3$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$



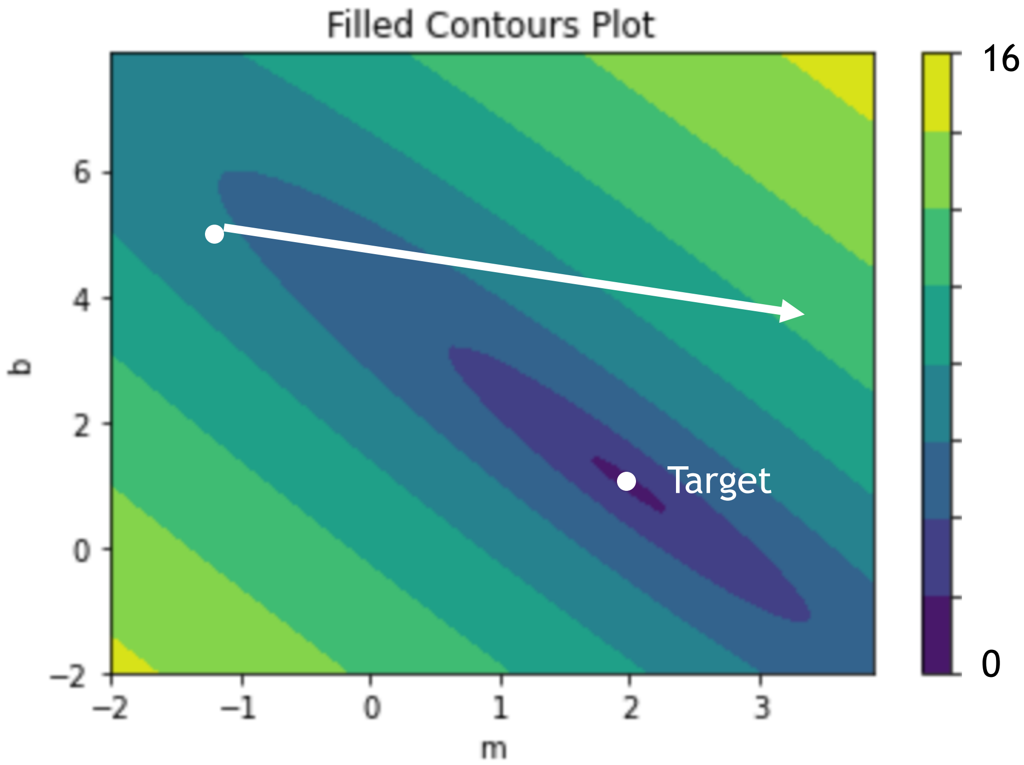
THE LOSS CURVE

$$\frac{\partial MSE}{\partial m} = -7 \qquad \frac{\partial MSE}{\partial b} = -3$$

$$\mathrm{m} := \mathrm{m} \ - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = .6$$

$$\mathrm{b} \ := b \ - \lambda \frac{\partial MSE}{\partial b}$$



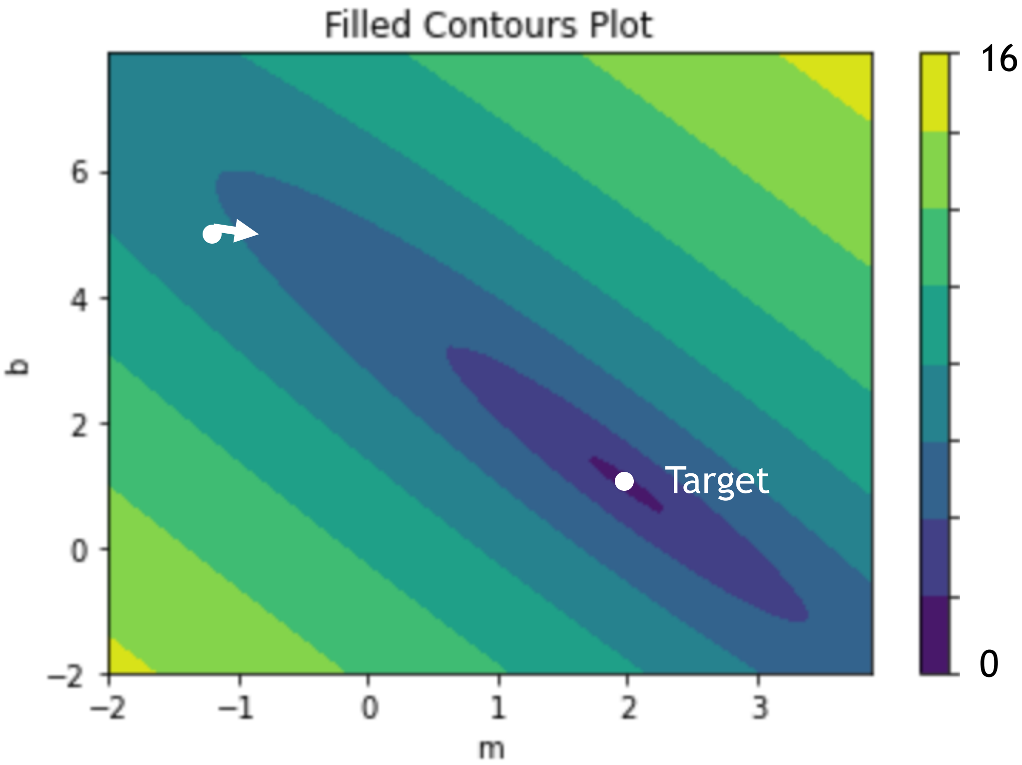
THE LOSS CURVE

$$\frac{\partial MSE}{\partial m} = -7 \qquad \frac{\partial MSE}{\partial b} = -3$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = .005$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$

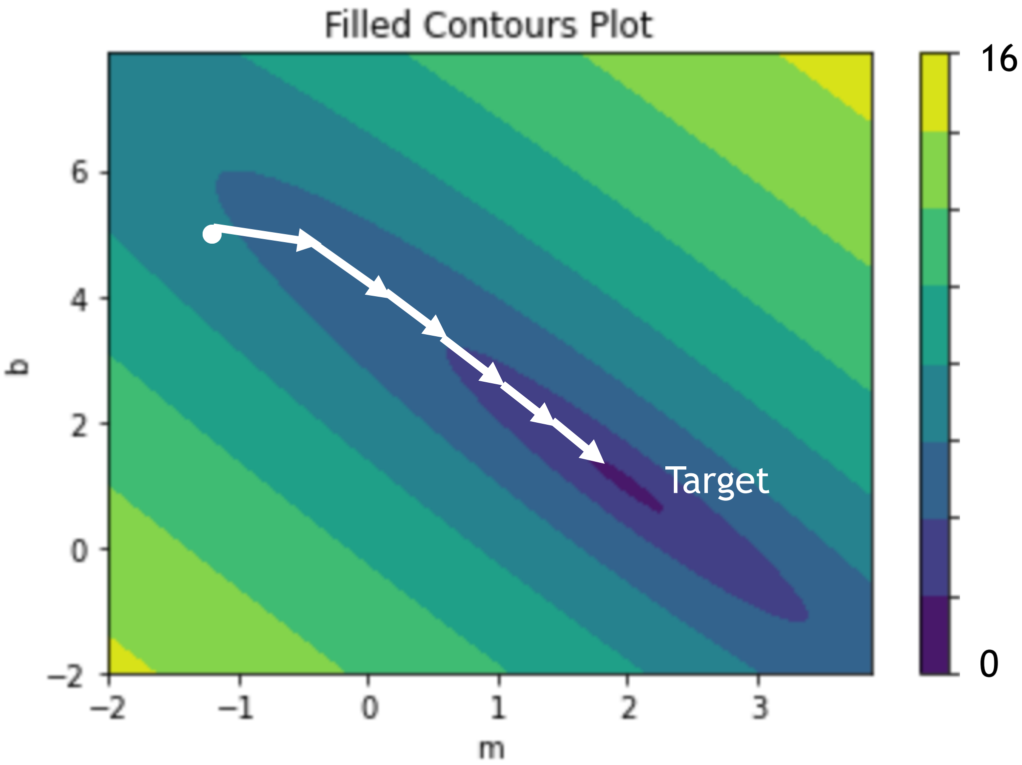


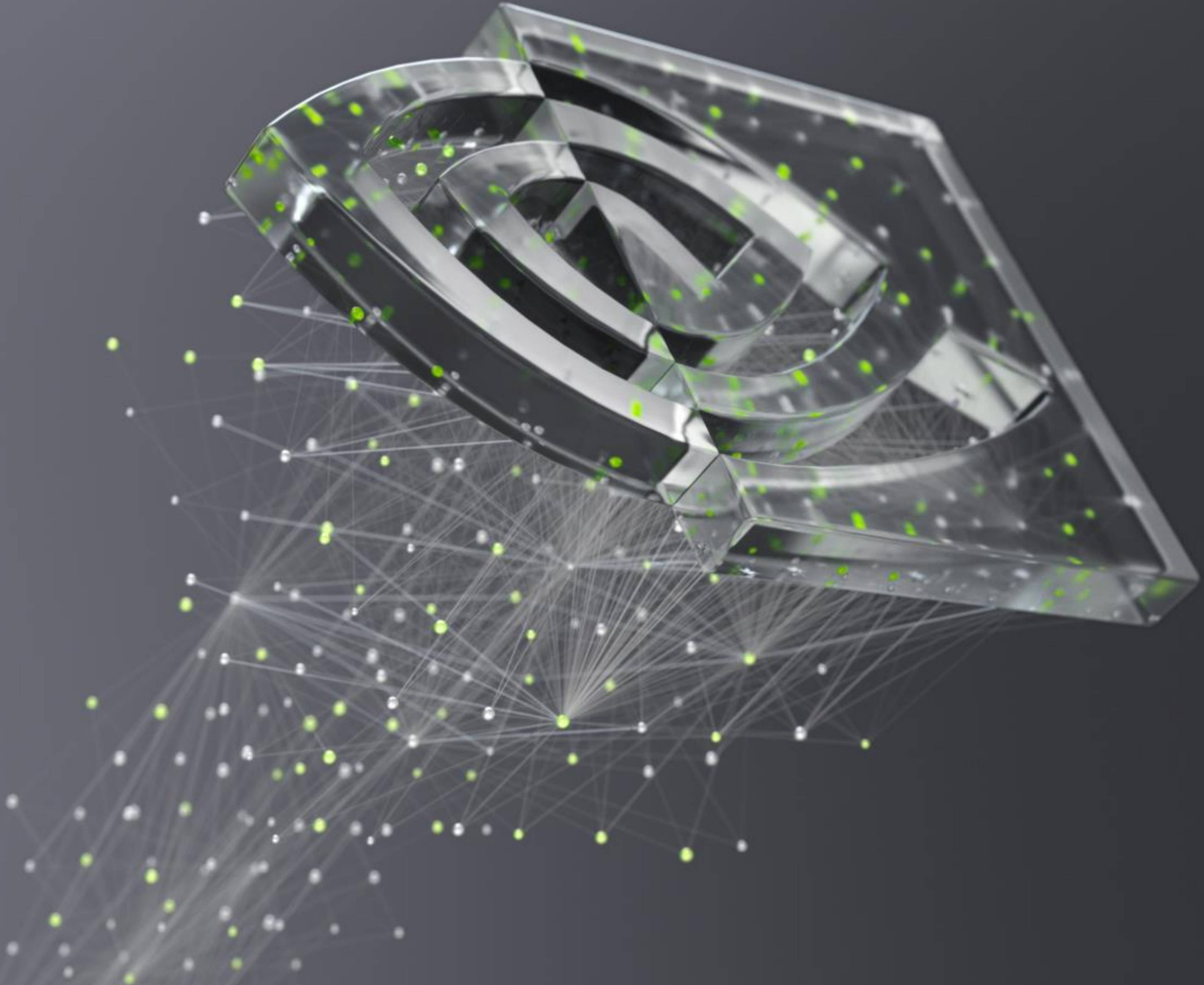
THE LOSS CURVE

$$\lambda = .1$$

$$m := -1 + 7\,\lambda = -0.3$$

$$b := 5 + 3\,\lambda = 4.7$$





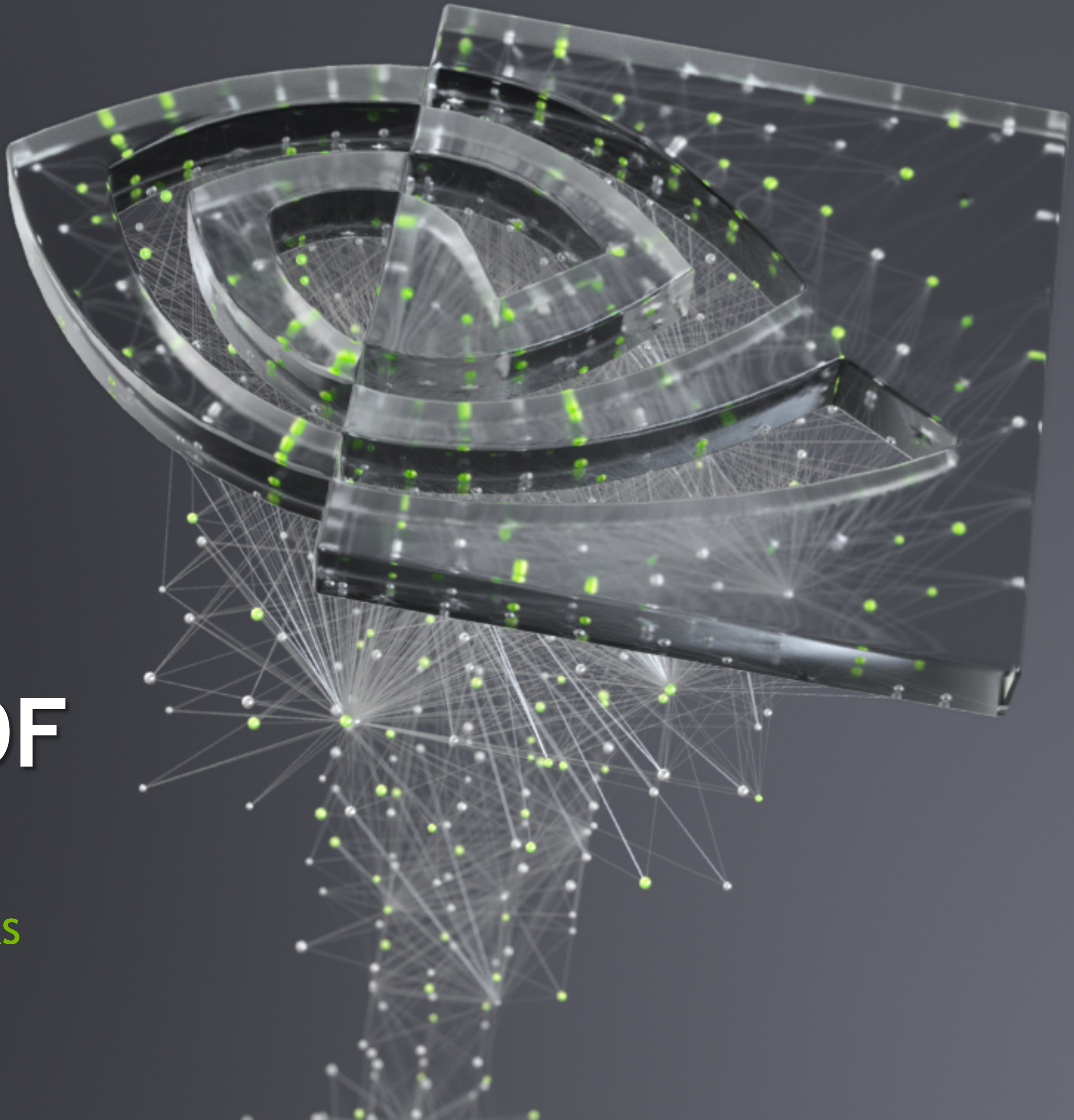
NVIDIA

DEEP
LEARNING
INSTITUTE

NVIDIA | DEEP LEARNING INSTITUTE

FUNDAMENTALS OF DEEP LEARNING

Part 3: Convolutional Neural Networks



Part 1: An Introduction to Deep Learning

Part 2: How a Neural Network Trains

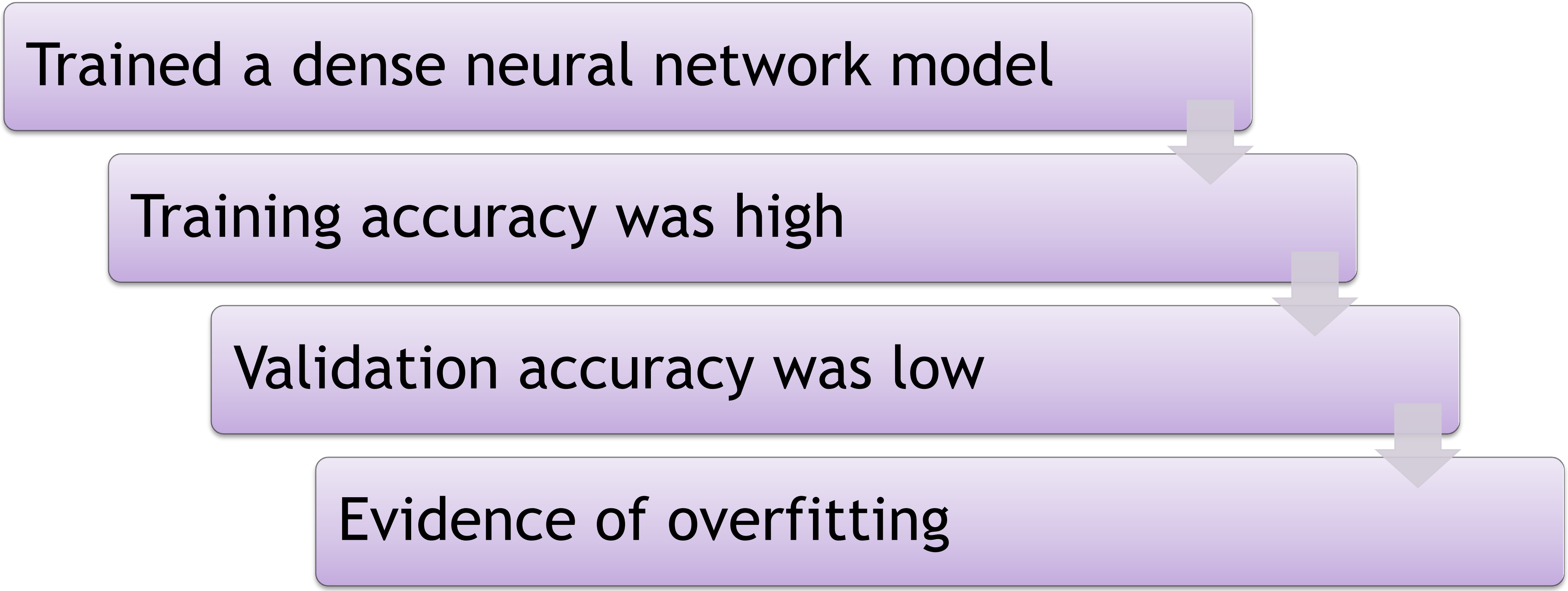
Part 3: Convolutional Neural Networks

Part 4: Data Augmentation and Deployment

Part 5: Pre-trained Models

Part 6: Advanced Architectures

RECAP OF THE EXERCISE



Trained a dense neural network model

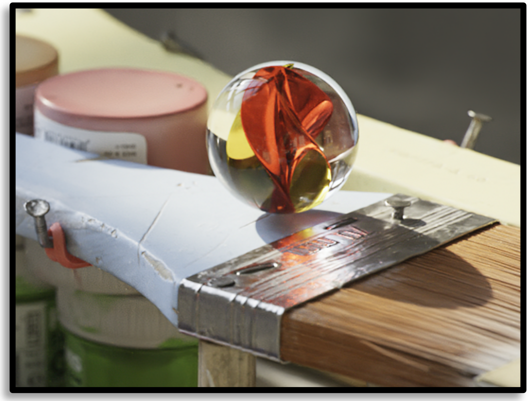
Training accuracy was high

Validation accuracy was low

Evidence of overfitting

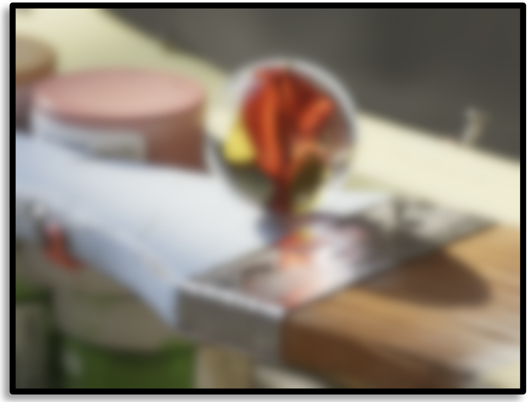
KERNELS AND CONVOLUTION

KERNELS AND CONVOLUTION

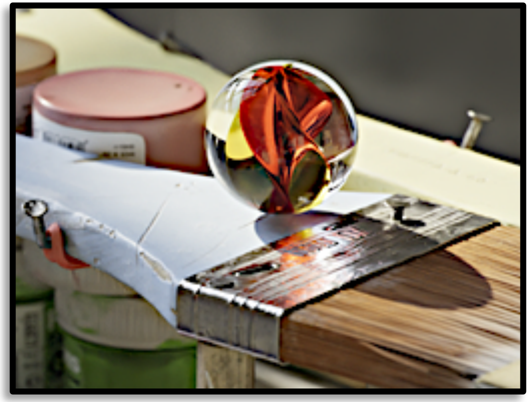


Original Image

Blur



Sharpen



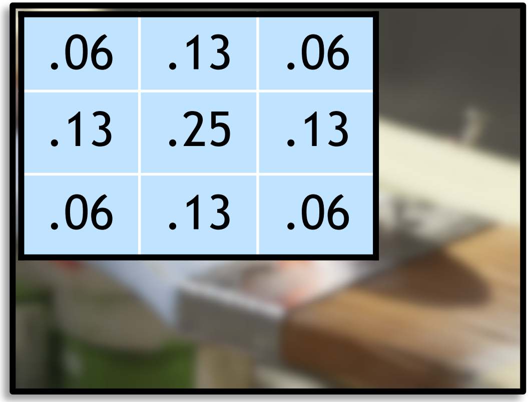
Brighten

Darken

KERNELS AND CONVOLUTION

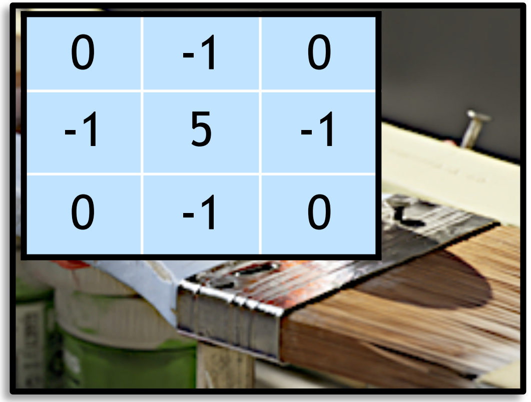
Original Image

Blur



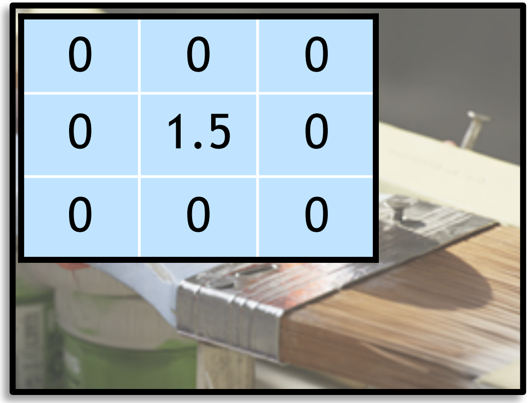
.06	.13	.06
.13	.25	.13
.06	.13	.06

Sharpen



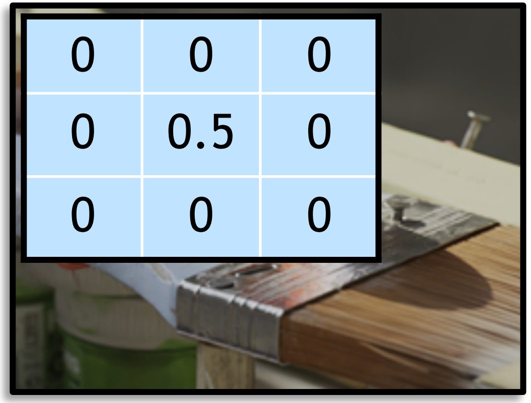
0	-1	0
-1	5	-1
0	-1	0

Brighten



0	0	0
0	1.5	0
0	0	0

Darken



0	0	0
0	0.5	0
0	0	0

KERNELS AND CONVOLUTION

Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

*

Original Image

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

=

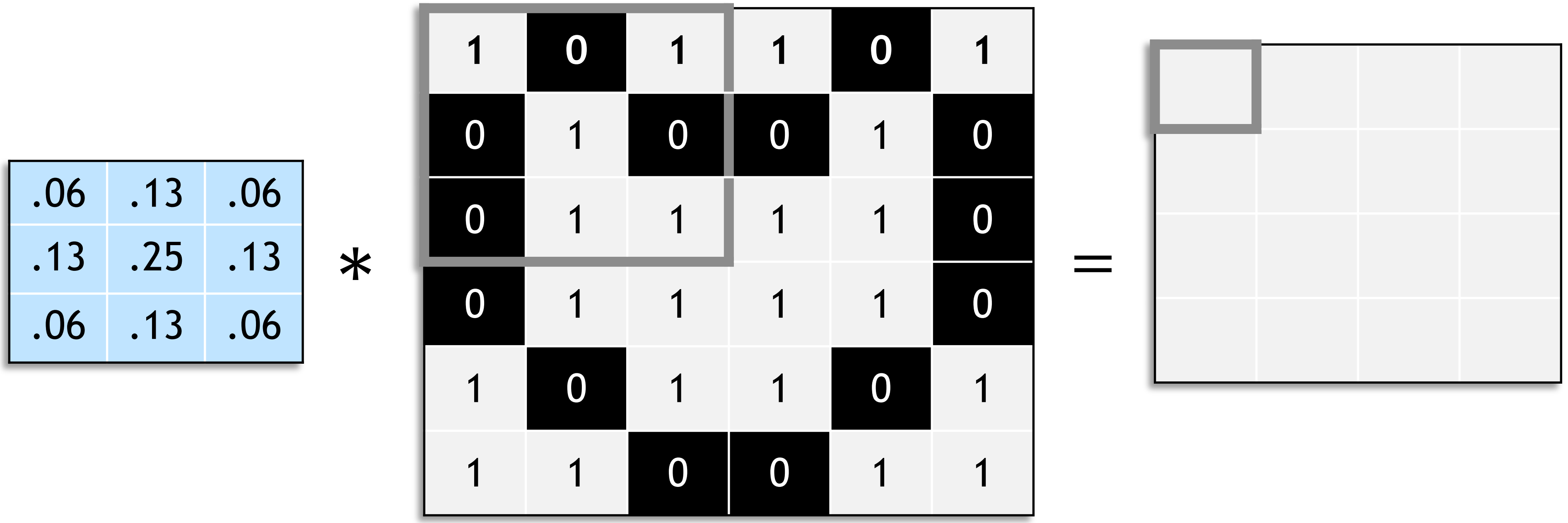
Convolved Image

KERNELS AND CONVOLUTION

Blur Kernel

Original Image

Convolved Image



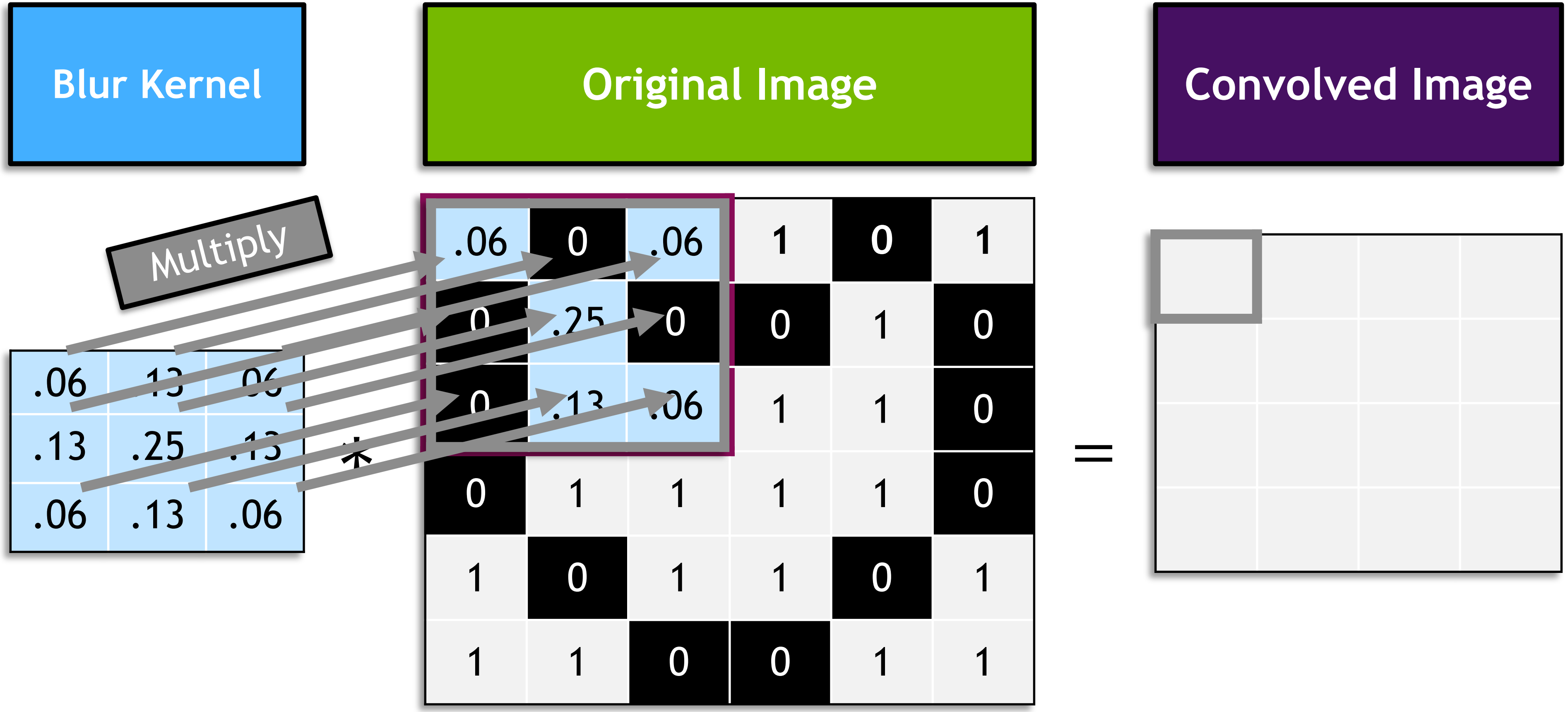
Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

Original Image

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

KERNELS AND CONVOLUTION



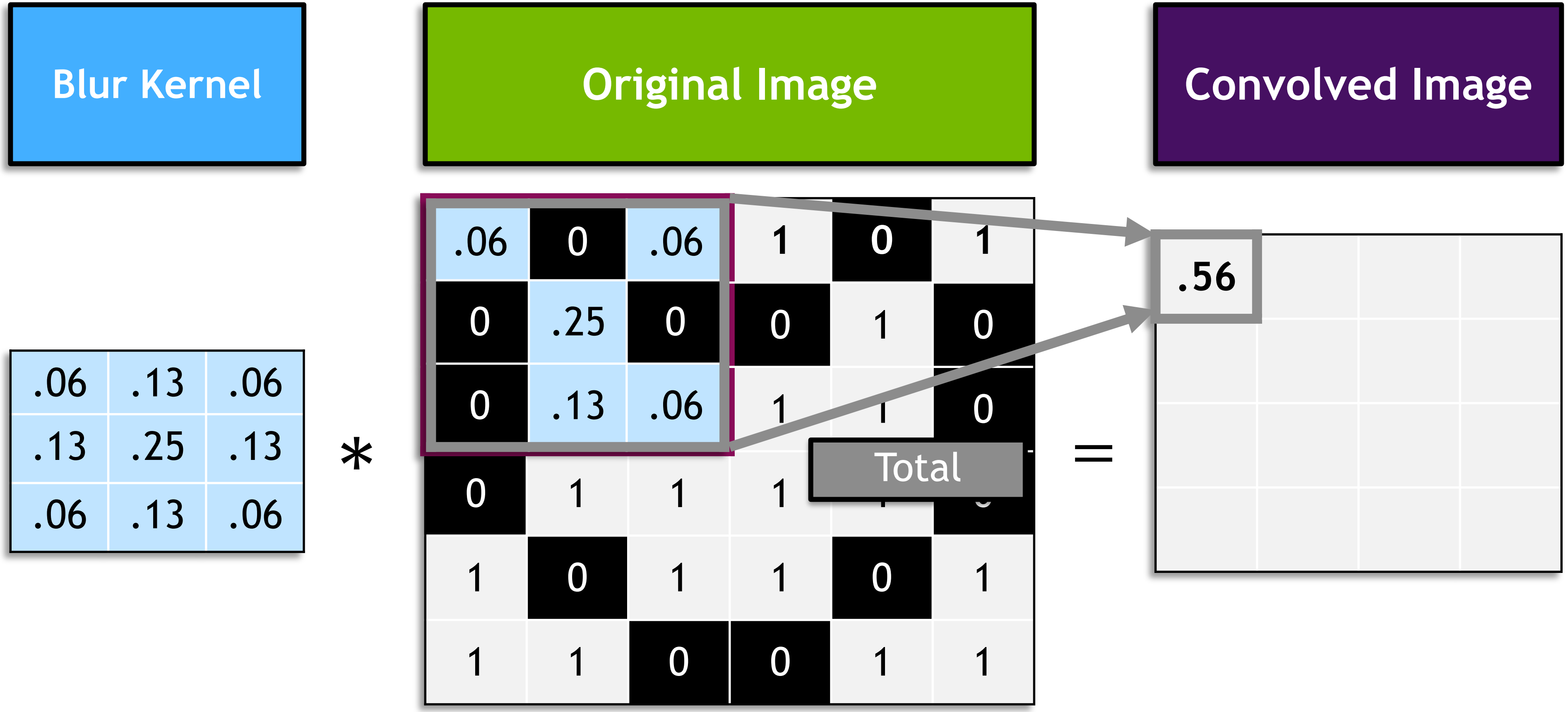
Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

Original Image

.06	0	.06	1	0	1
0	.25	0	0	1	0
0	.13	.06	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

KERNELS AND CONVOLUTION



Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

*

Original Image

.06	0	.06	1	0	1
0	.25	0	0	1	0
0	.13	.06	1	1	0
0	1	1	1	[illegible]	[illegible]
1	0	1	1	0	1
1	1	0	0	1	1

Total

=

Convolved Image

.56			

KERNELS AND CONVOLUTION

Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

*

Original Image

1	0	.13	.06	0	1
0	.13	0	0	1	0
0	.06	.13	.06	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

=

Convolved Image

.56	.57		

KERNELS AND CONVOLUTION

Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

*

Original Image

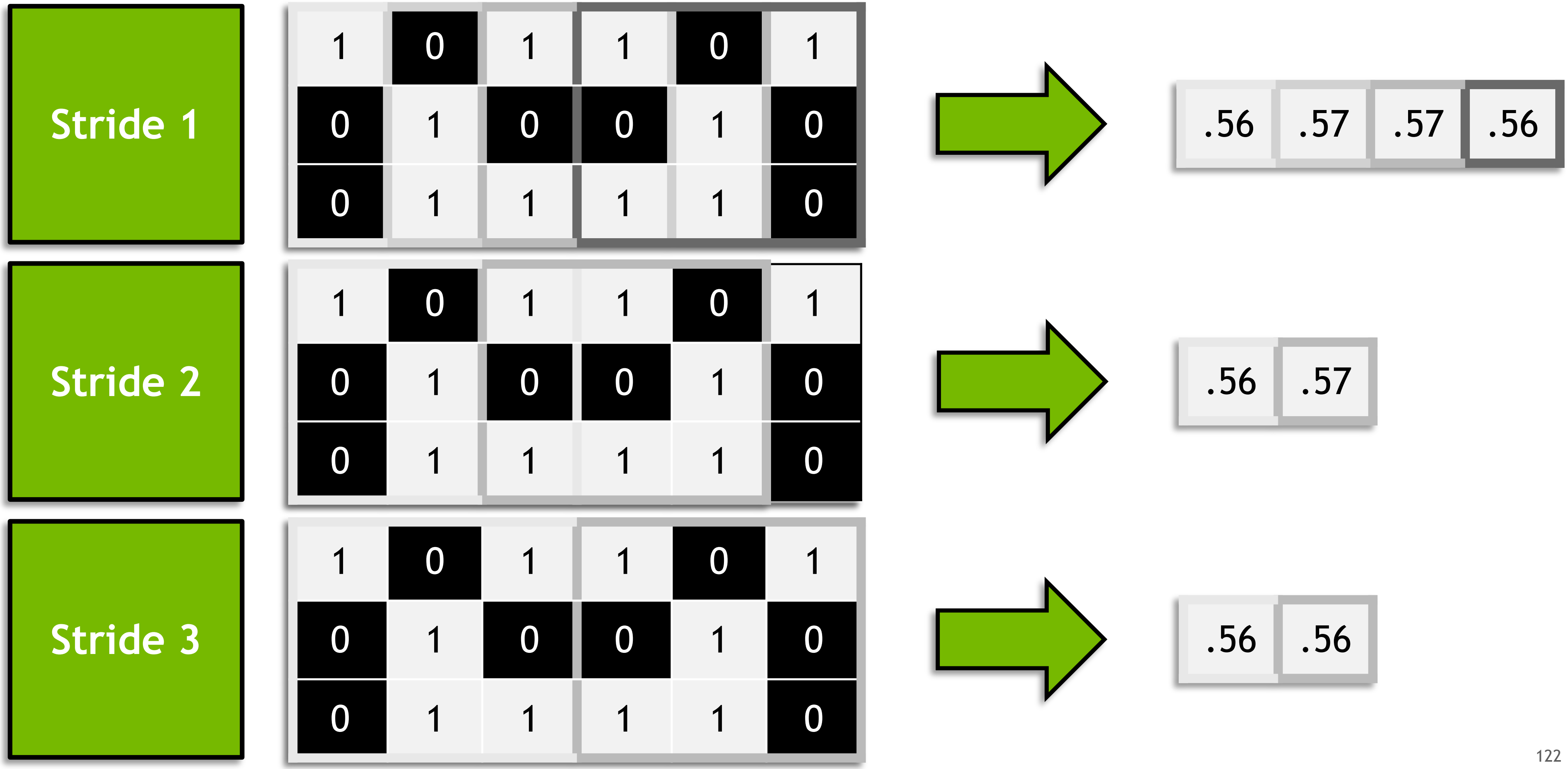
1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

=

Convolved Image

.56	.57	.57	.56
.7	.82	.82	.7
.69	.95	.95	.69
.64	.69	.69	.64

STRIDE



Stride 1

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

.56	.57	.57	.56

Stride 2

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

.56	.57

Stride 3

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

.56	.56

PADDING

Original Image

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

Zero Padding

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

PADDING

Original Image

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

Mirror Padding

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

KERNELS AND NEURAL NETWORKS

KERNELS AND NEURAL NETWORKS

Kernel

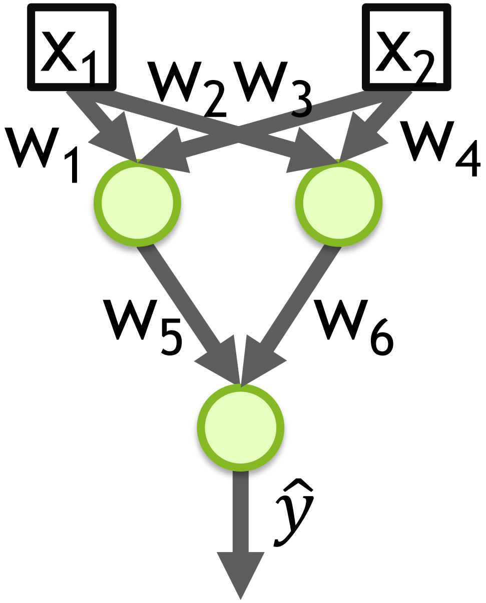
W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

KERNELS AND NEURAL NETWORKS

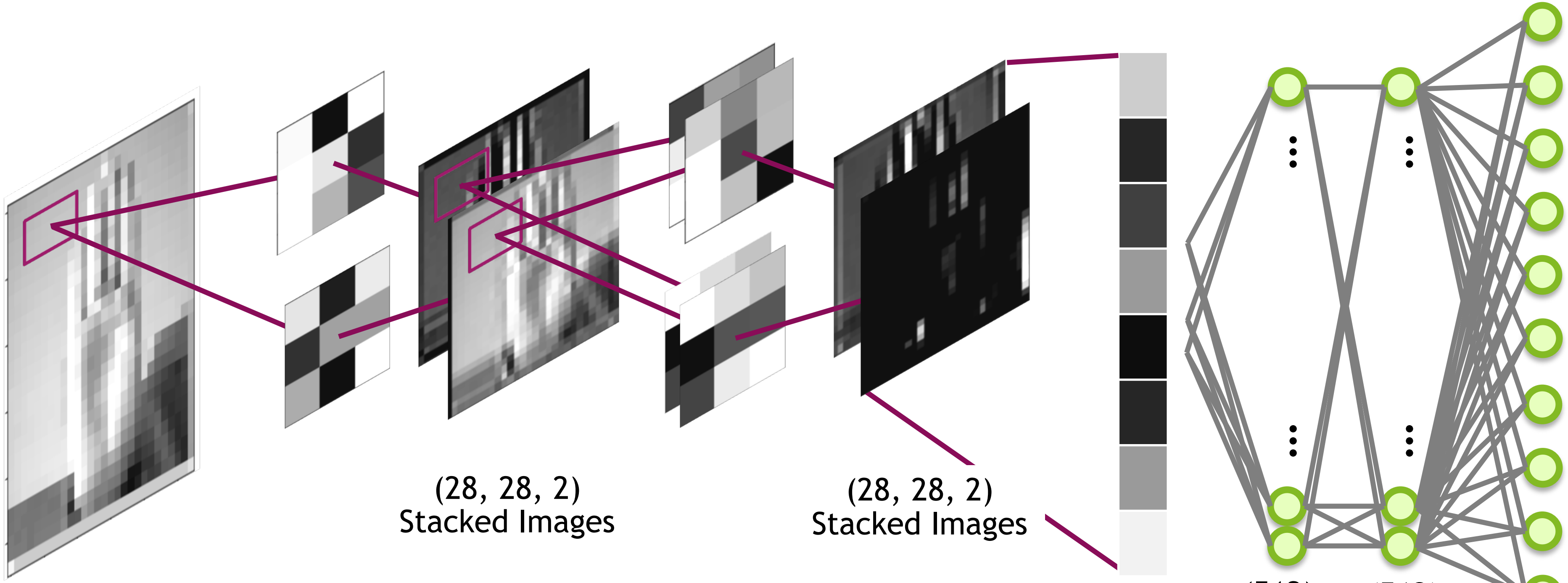
Kernel

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Neuron



KERNELS AND NEURAL NETWORKS



(28, 28, 1)
Image Input

(3, 3, 1, 2)
Kernels

(28, 28, 2)
Stacked Images

(3, 3, 2, 2)
Kernels

(28, 28, 2)
Stacked Images

(1568)
Flattened Image
Vector

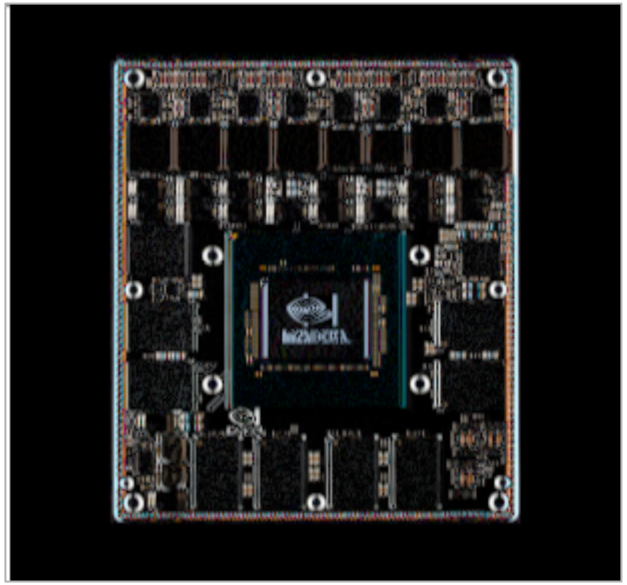
(512)
Dense

(512)
Dense

(24)
Output Prediction

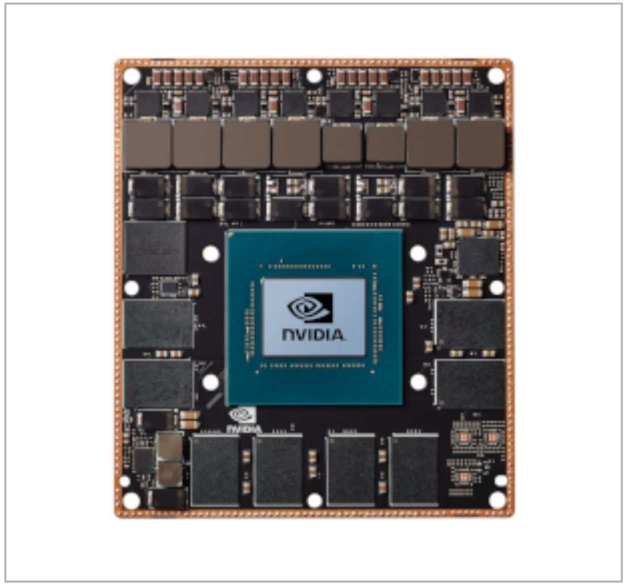
FINDING EDGES

Vertical Edges



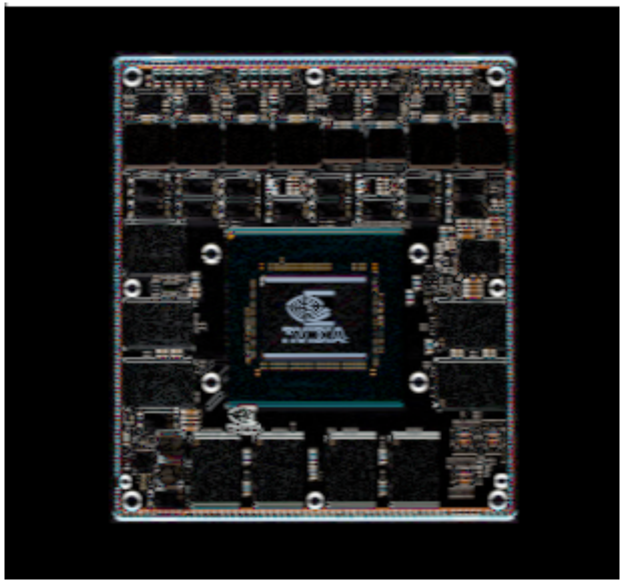
1	0	-1
2	0	-2
1	0	-1

Original Image



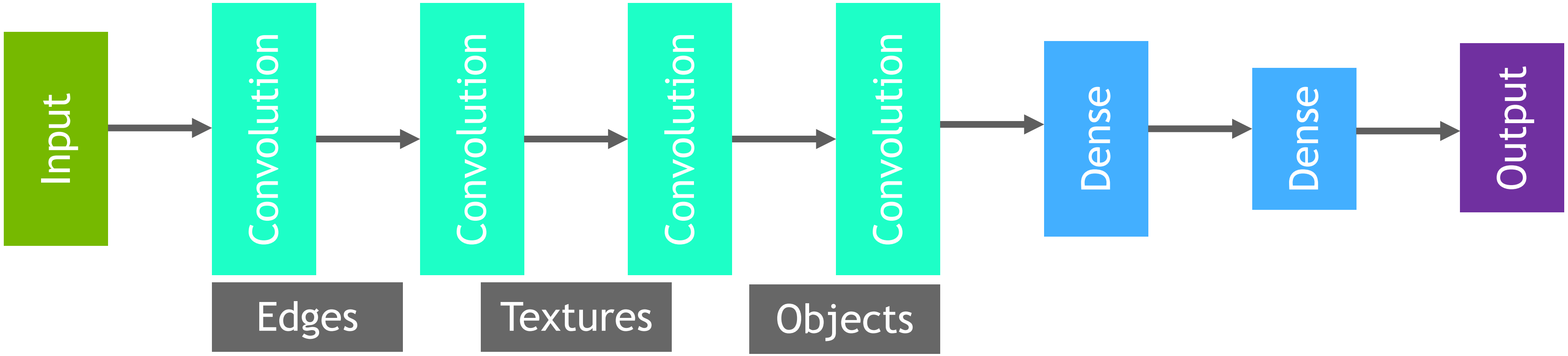
0	0	0
0	1	0
0	0	0

Horizontal Edges



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

NEURAL NETWORK PERCEPTION



Input

Convolution

Convolution

Convolution

Convolution

Dense

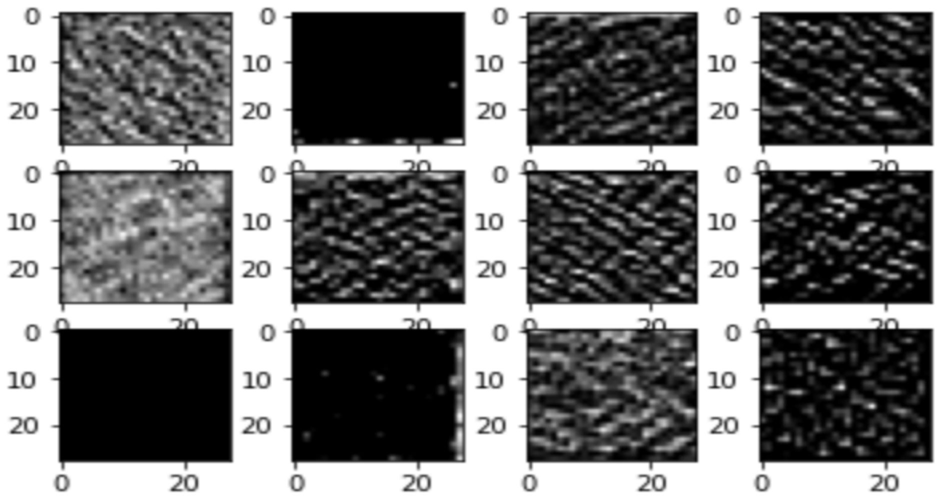
Dense

Output

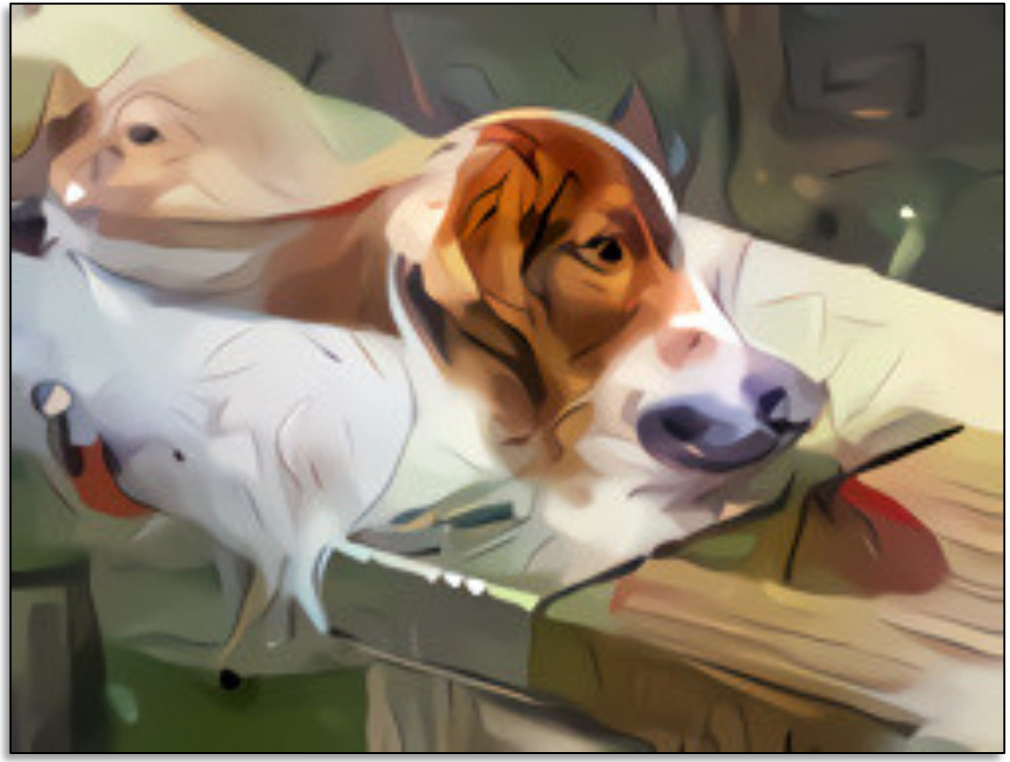
Edges

Textures

Objects

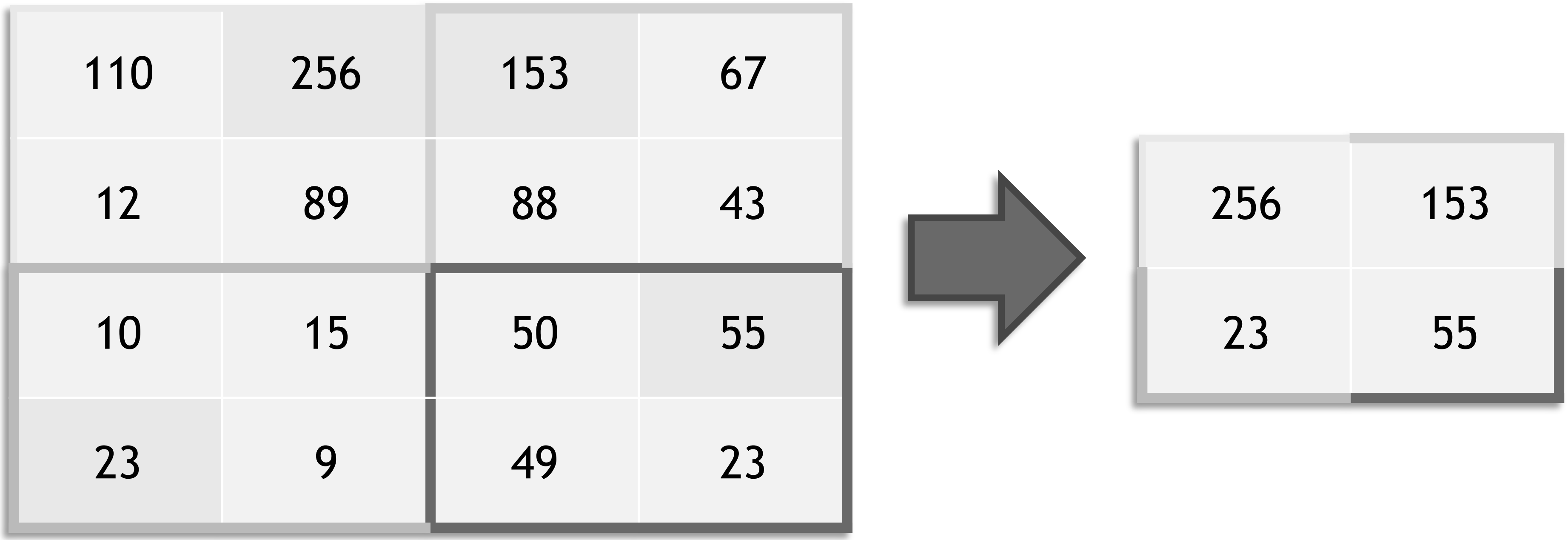


NEURAL NETWORK PERCEPTION



OTHER LAYERS IN THE MODEL

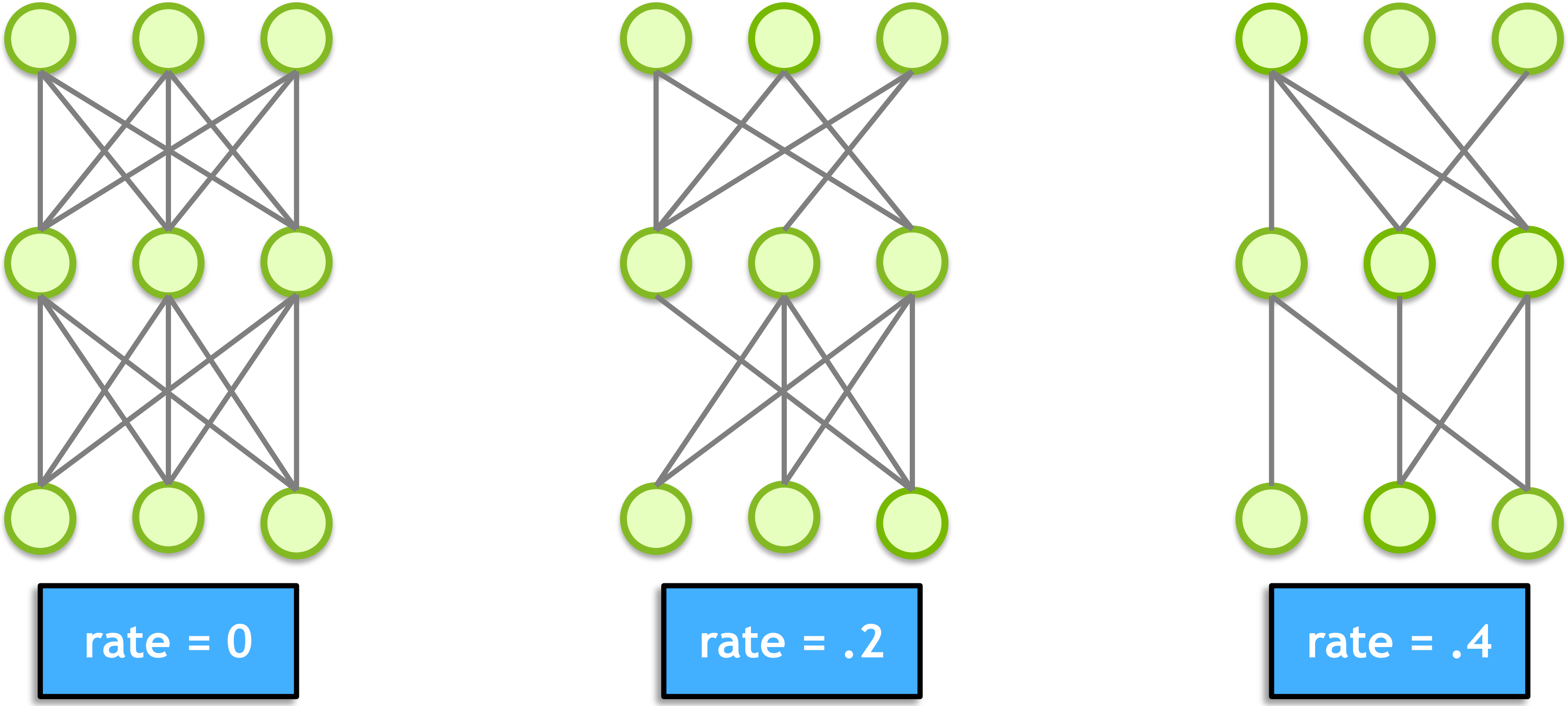
MAX POOLING



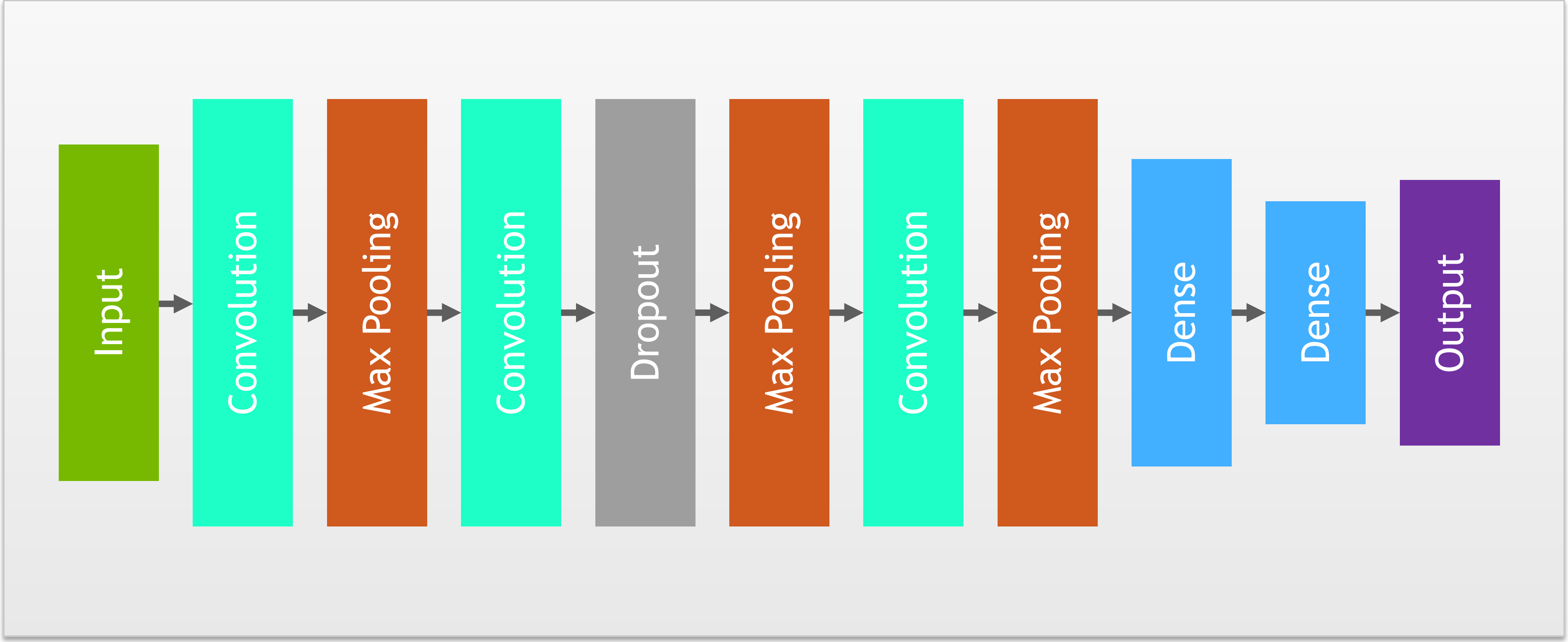
110	256	153	67
12	89	88	43
10	15	50	55
23	9	49	23

256	153
23	55

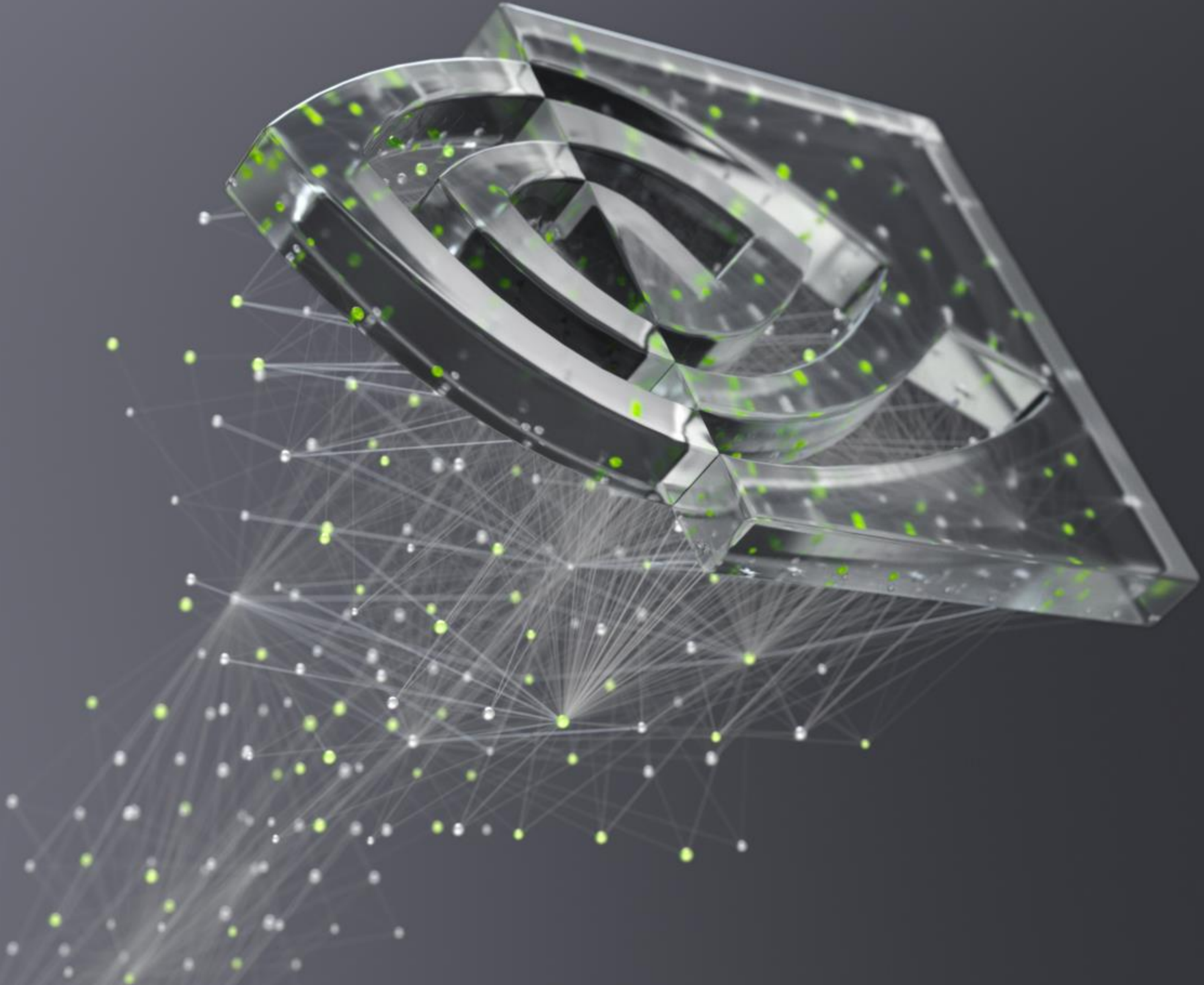
DROPOUT



WHOLE ARCHITECTURE



LET’S GO!



NVIDIA

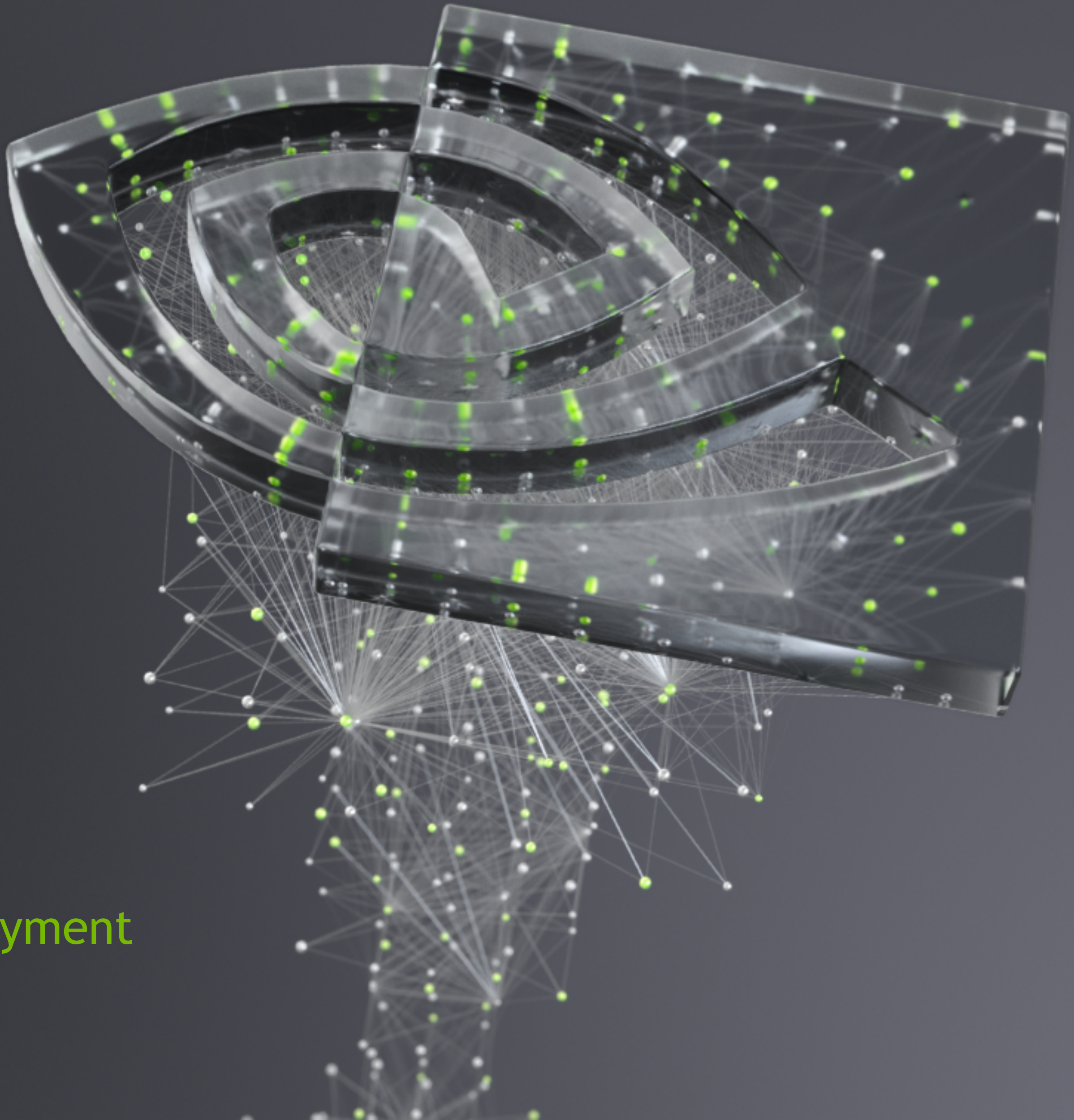
DEEP
LEARNING
INSTITUTE

NVIDIA

DEEP LEARNING INSTITUTE

FUNDAMENTALS OF DEEP LEARNING

Part 4: Data Augmentation and Deployment



- Part 1: An Introduction to Deep Learning
- Part 2: How a Neural Network Trains
- Part 3: Convolutional Neural Networks
- Part 4: Data Augmentation and Deployment
- Part 5: Pre-trained Models
- Part 6: Advanced Architectures

RECAP OF THE EXERCISE

Analysis

- CNN increased validation accuracy
- Still seeing training accuracy higher than validation

Solution

- Clean data provides better examples
- Dataset variety helps the model generalize

DATA AUGMENTATION

DATA AUGMENTATION

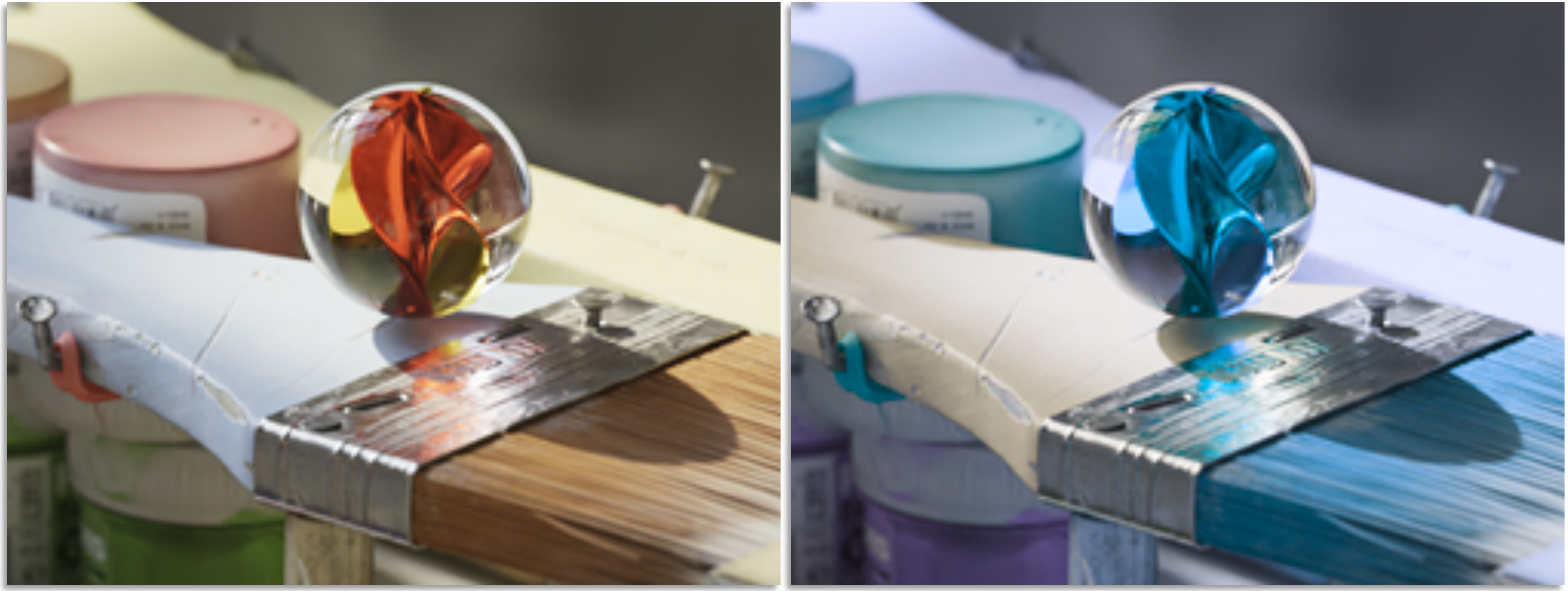
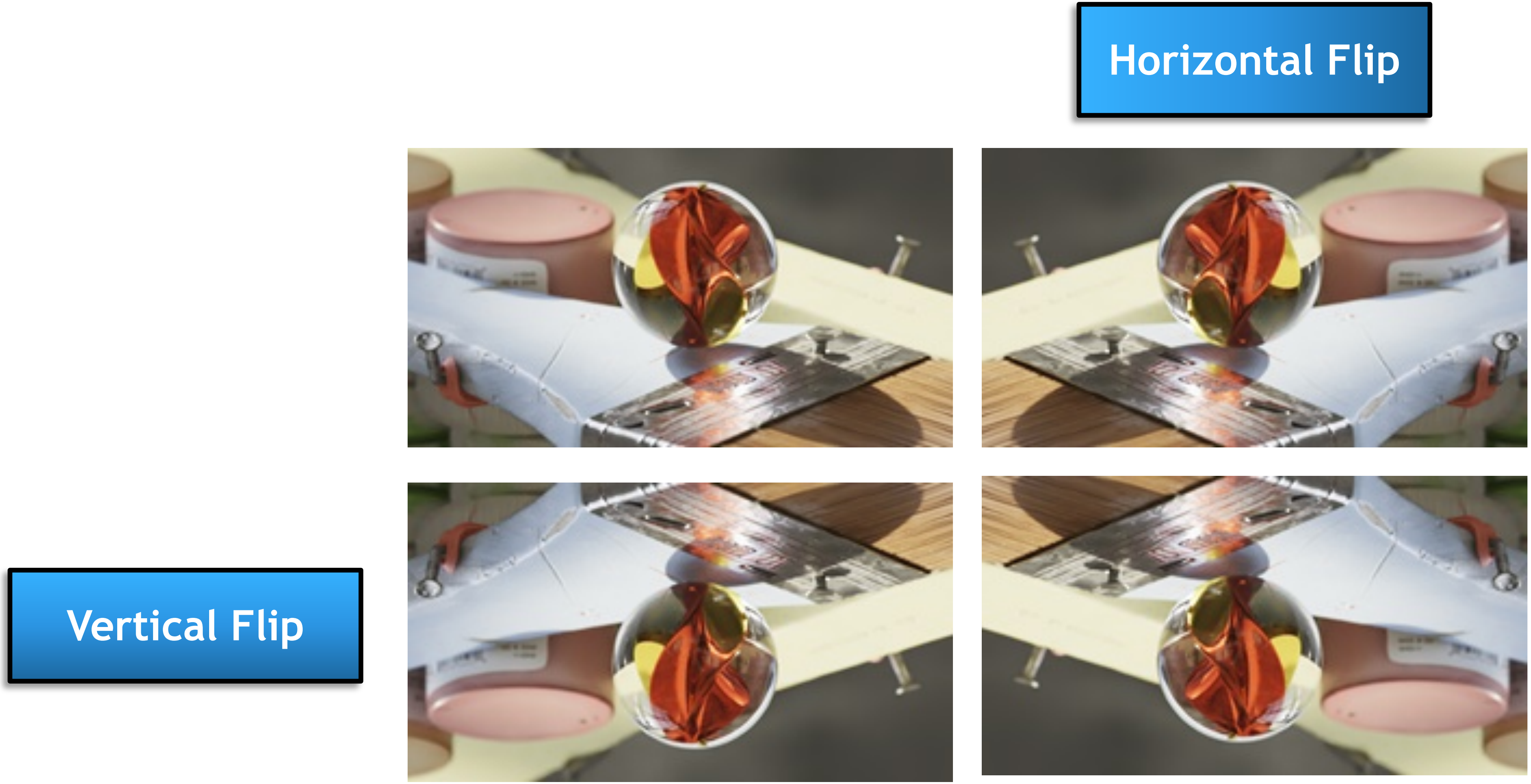


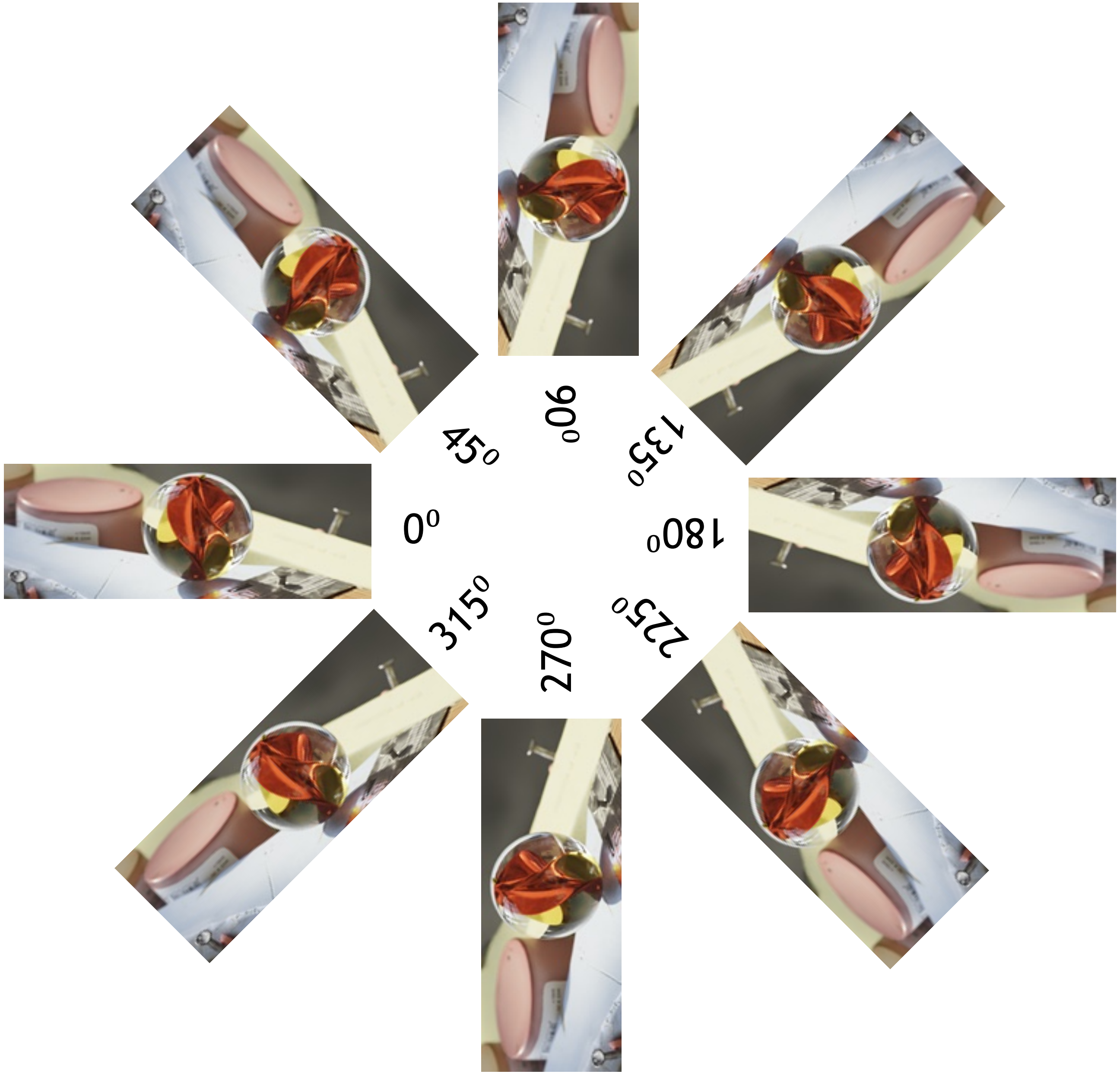
IMAGE FLIPPING



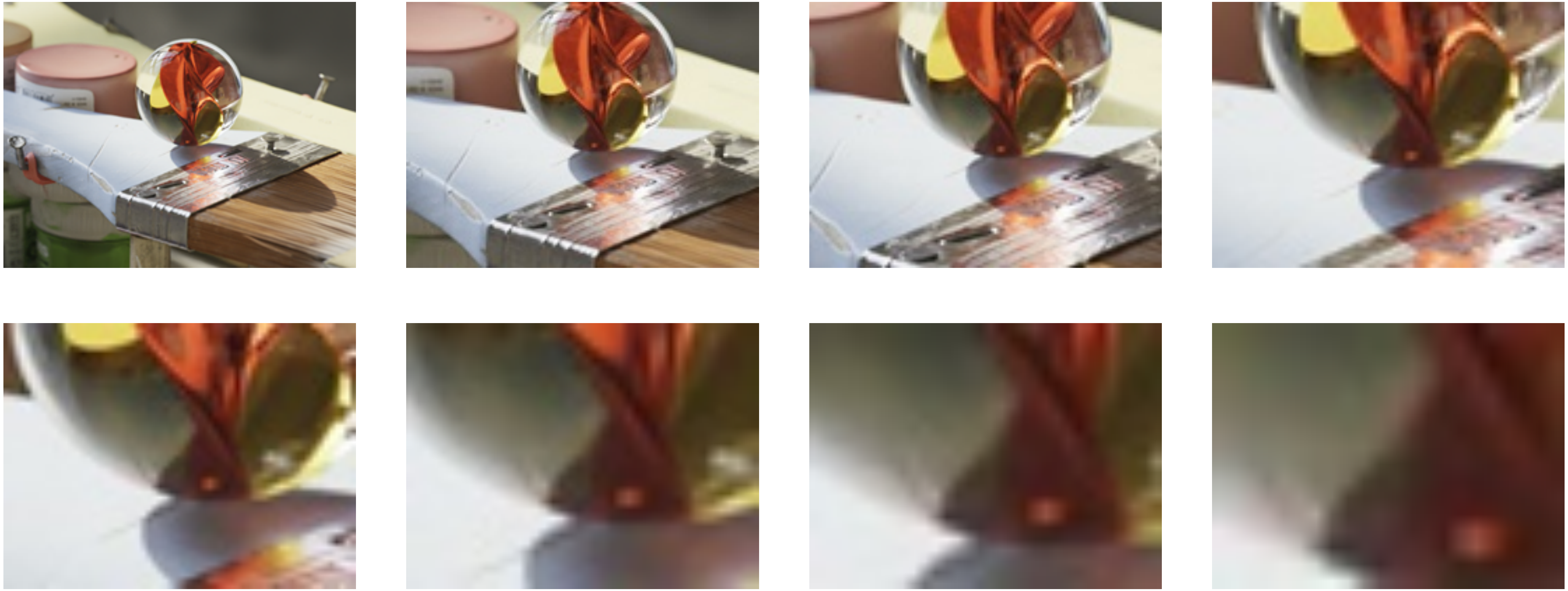
Horizontal Flip

Vertical Flip

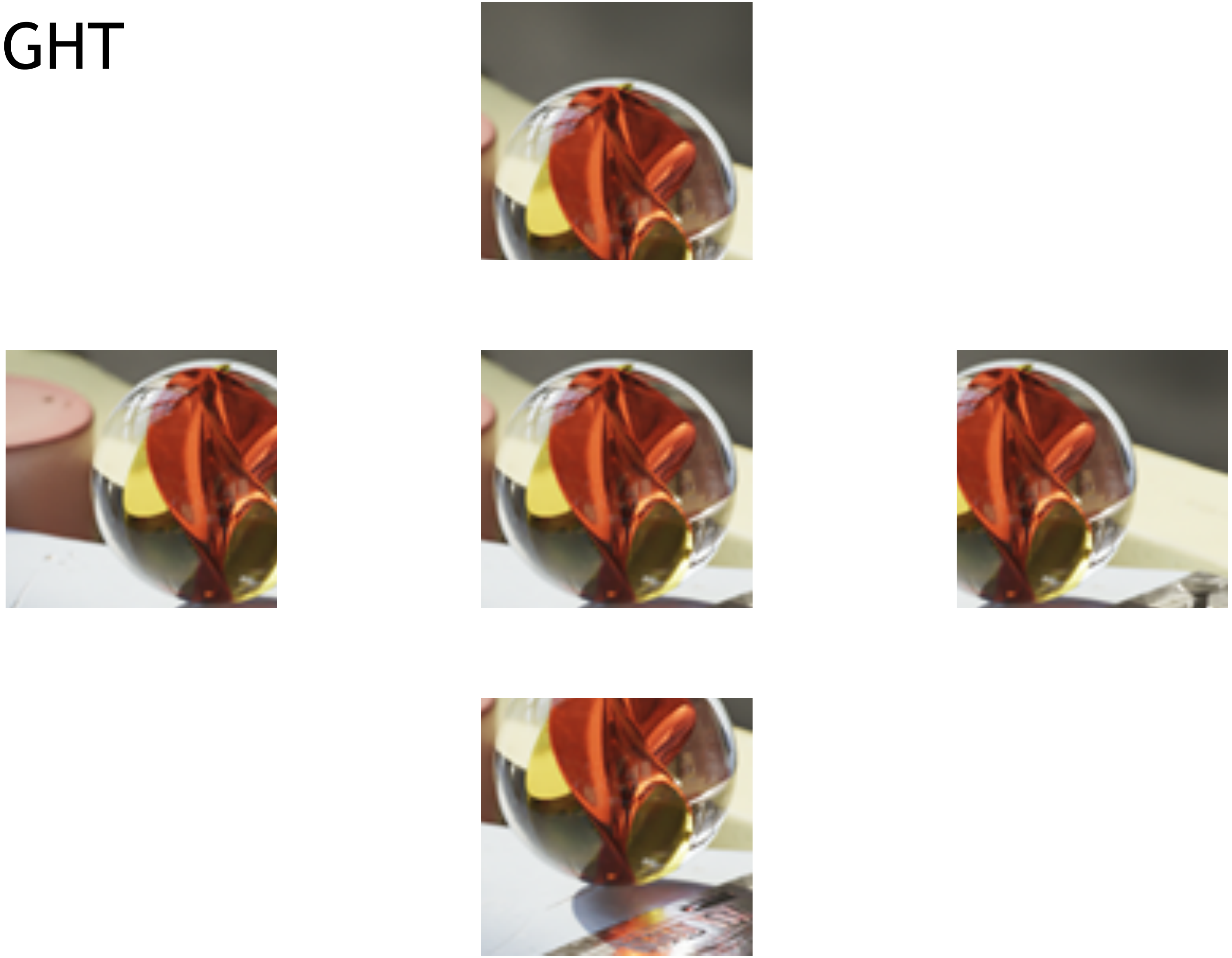
ROTATION



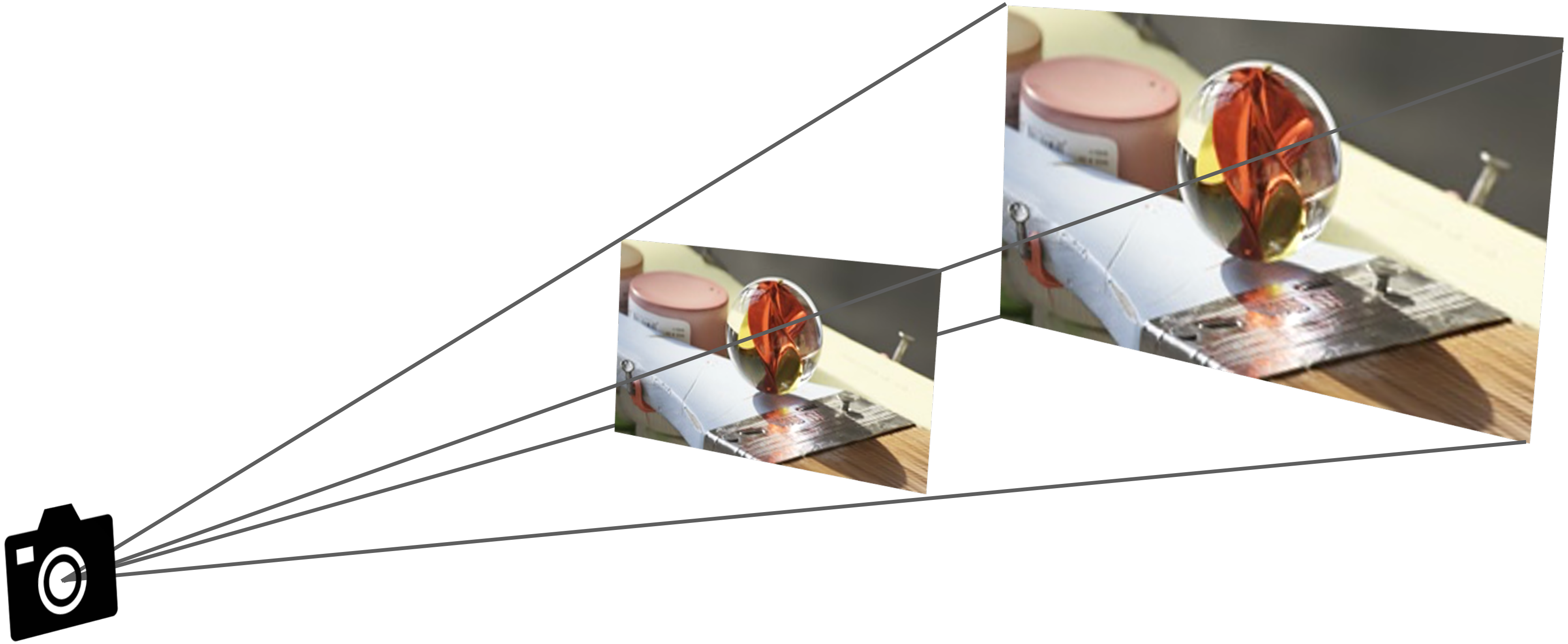
ZOOMING



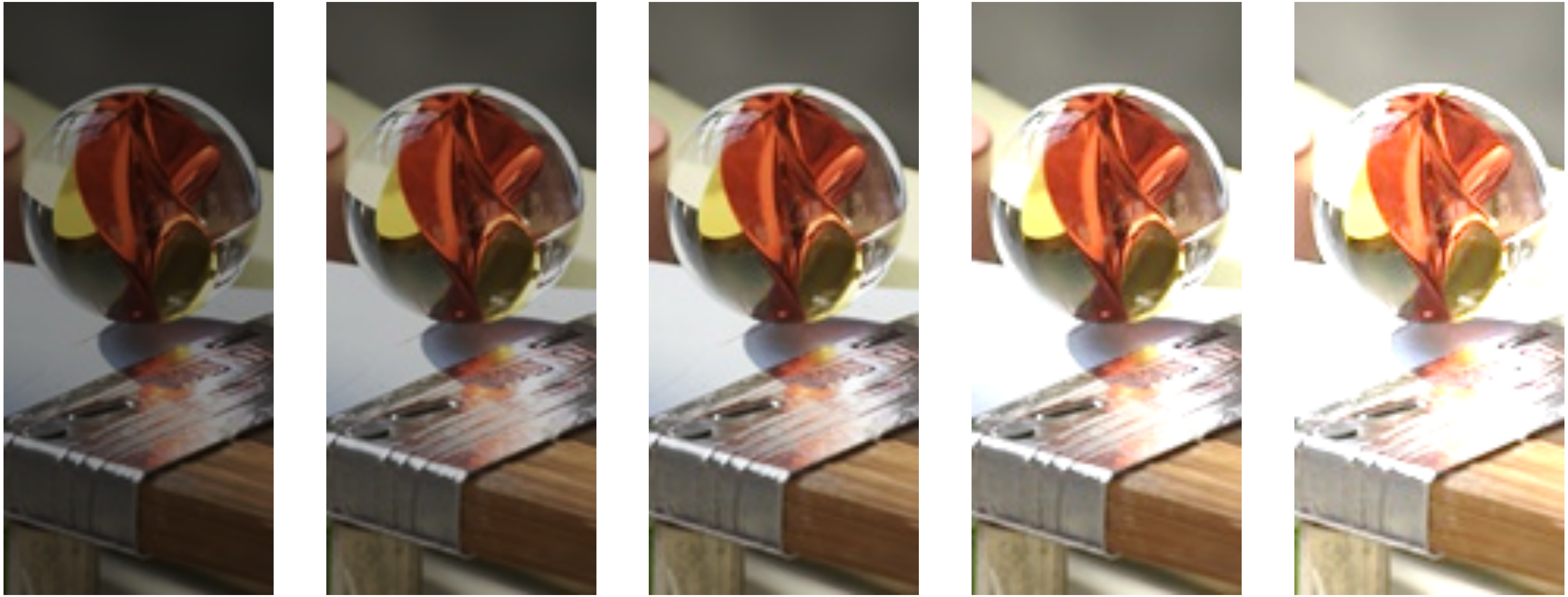
WIDTH AND HEIGHT SHIFTING



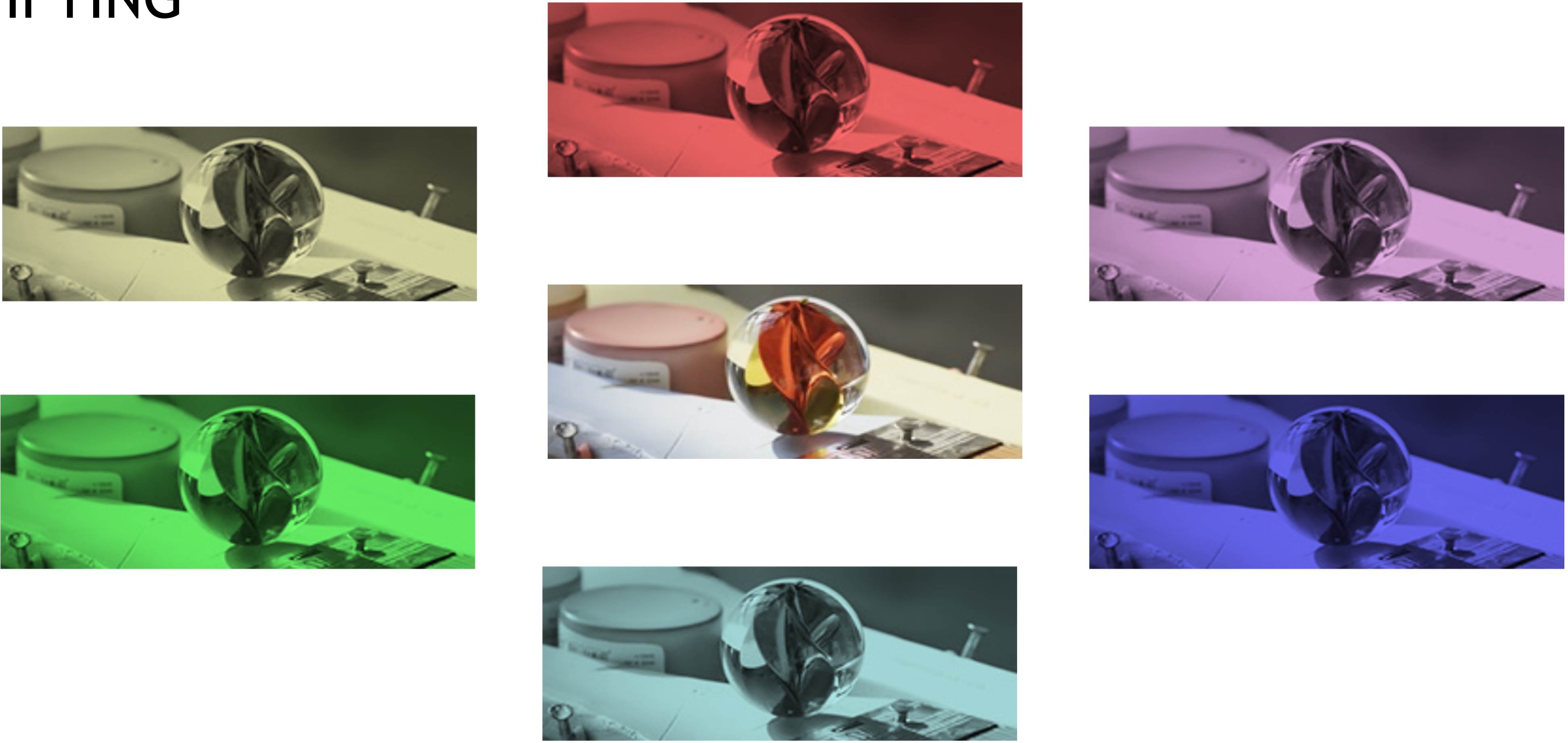
HOMOGRAPHY



BRIGHTNESS

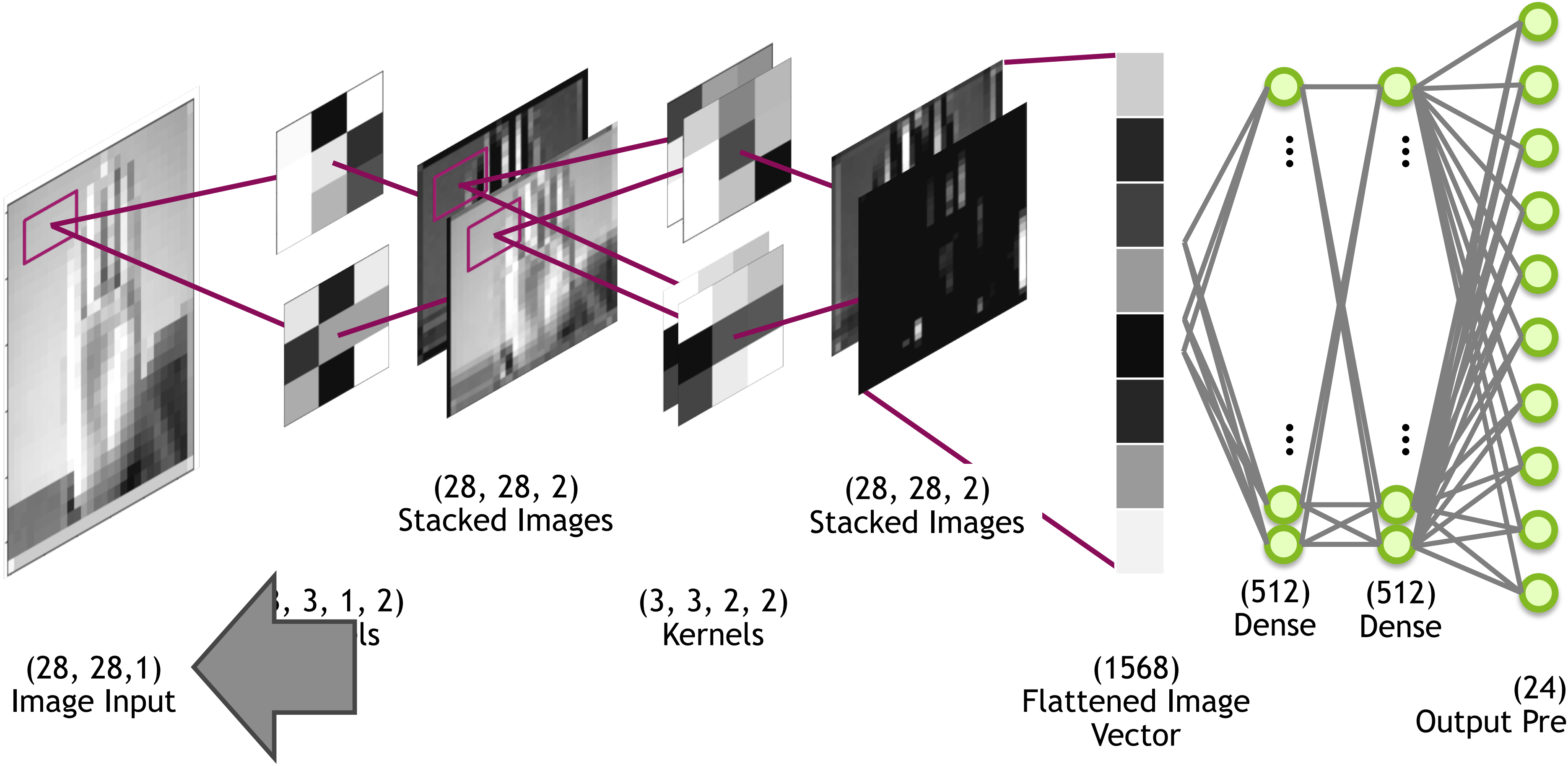


CHANNEL SHIFTING



MODEL DEPLOYMENT

MODEL DEPLOYMENT



(28, 28, 2)
Stacked Images

(28, 28, 2)
Stacked Images

3, 3, 1, 2)
ls

(3, 3, 2, 2)
Kernels

(512)
Dense

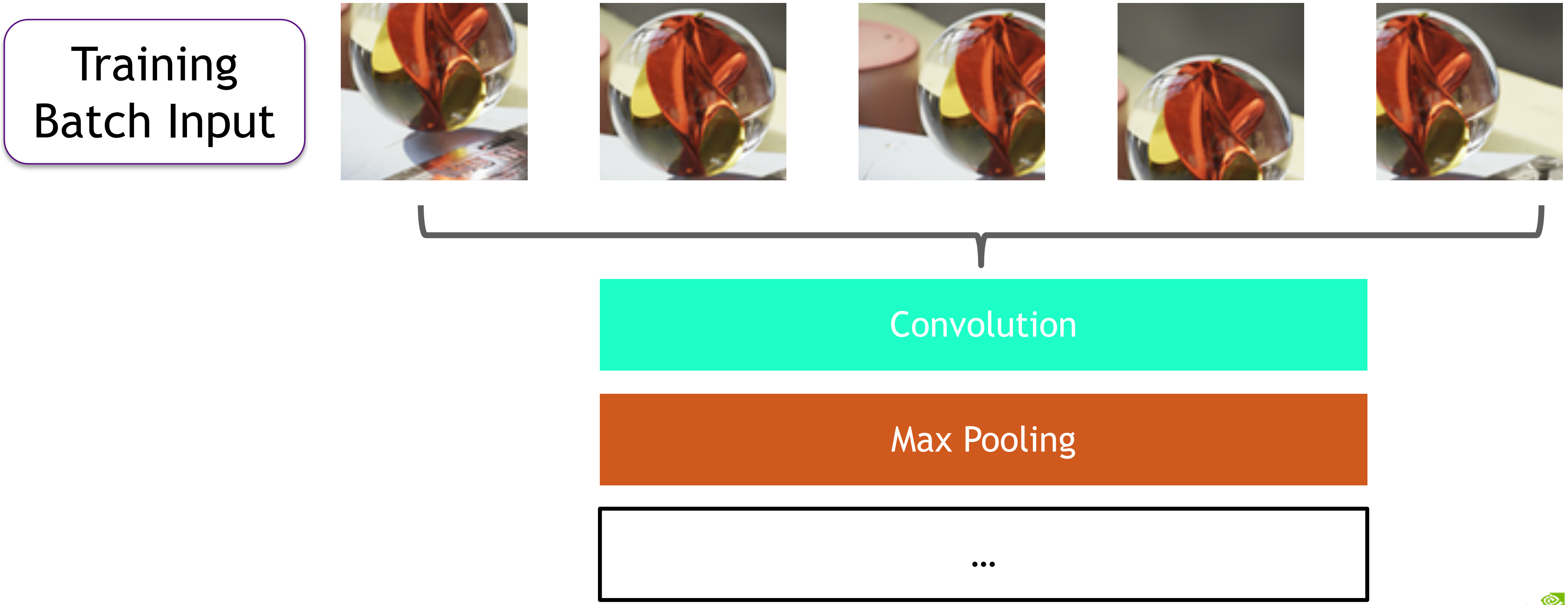
(512)
Dense

(28, 28,1)
Image Input

(1568)
Flattened Image
Vector

(24)
Output Prediction

MODEL DEPLOYMENT



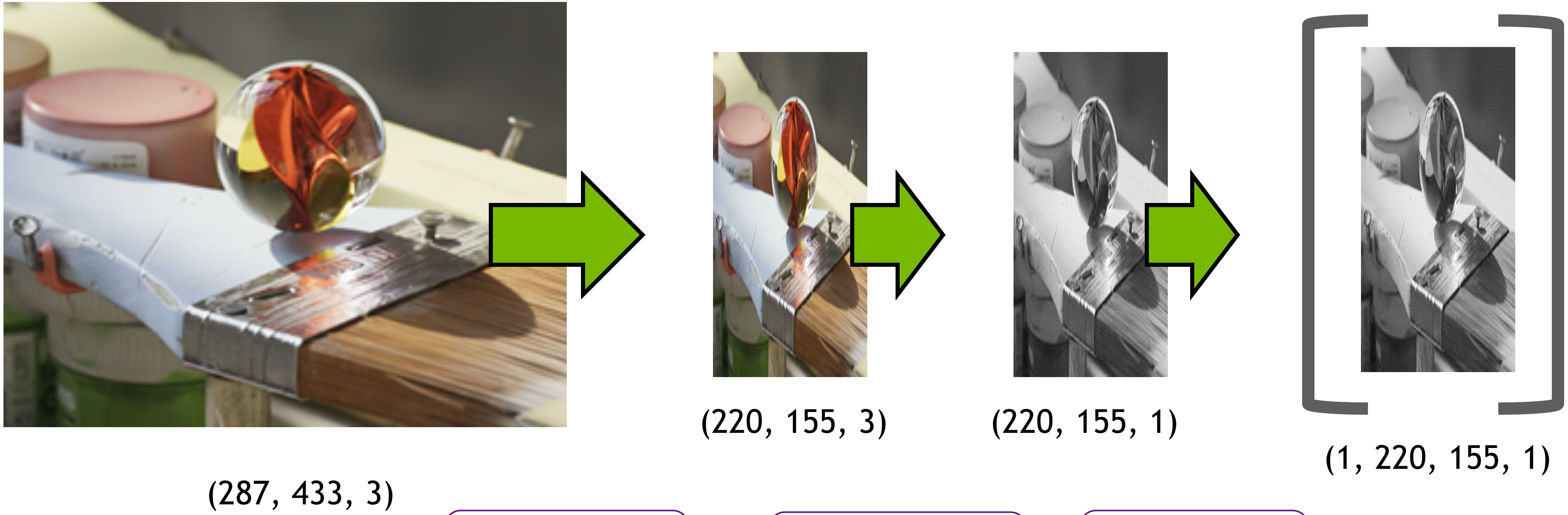
Training Batch Input

Convolution

Max Pooling

...

MODEL DEPLOYMENT

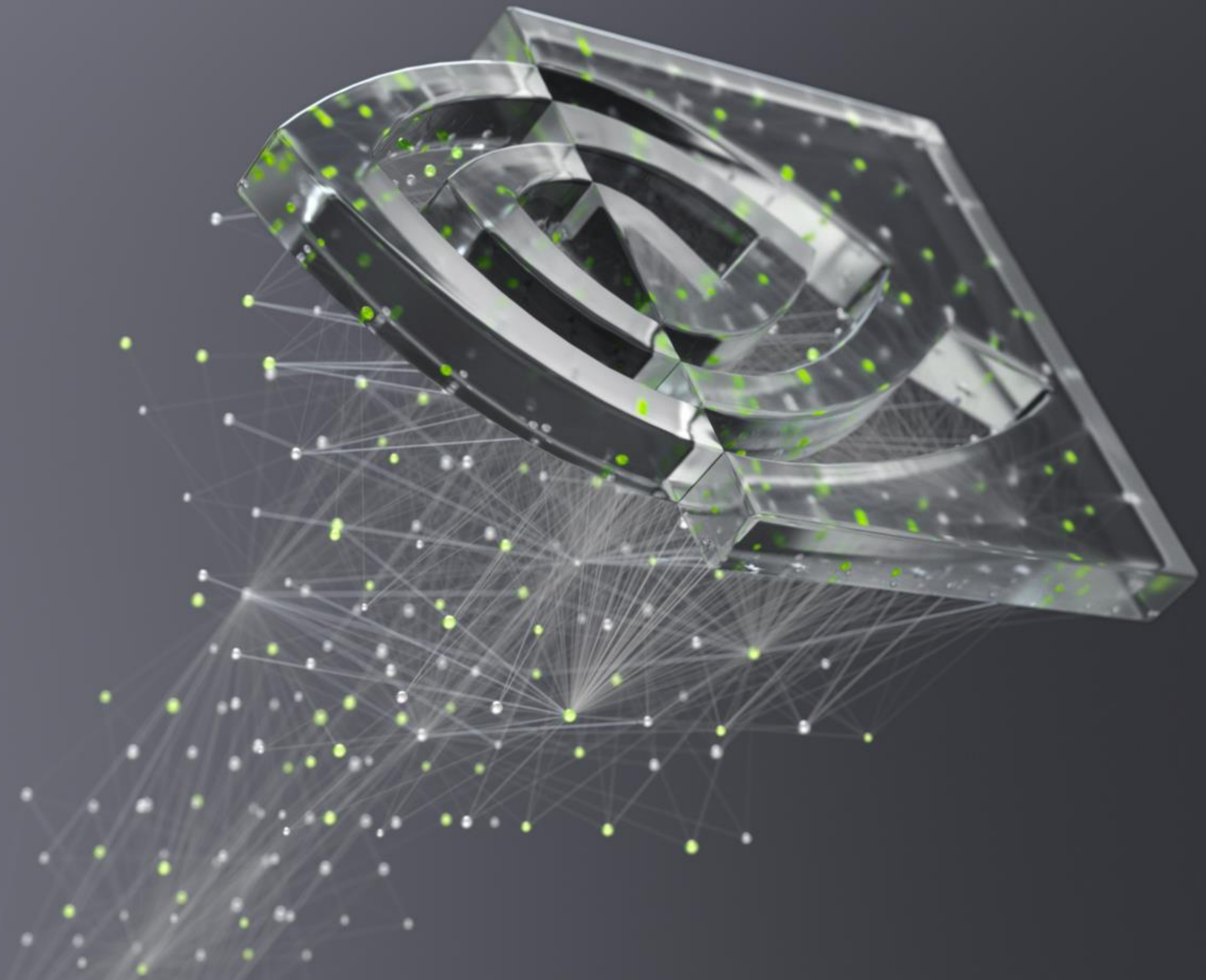


Resize

Greyscale

"Batch"

LET’S TRY IT OUT!



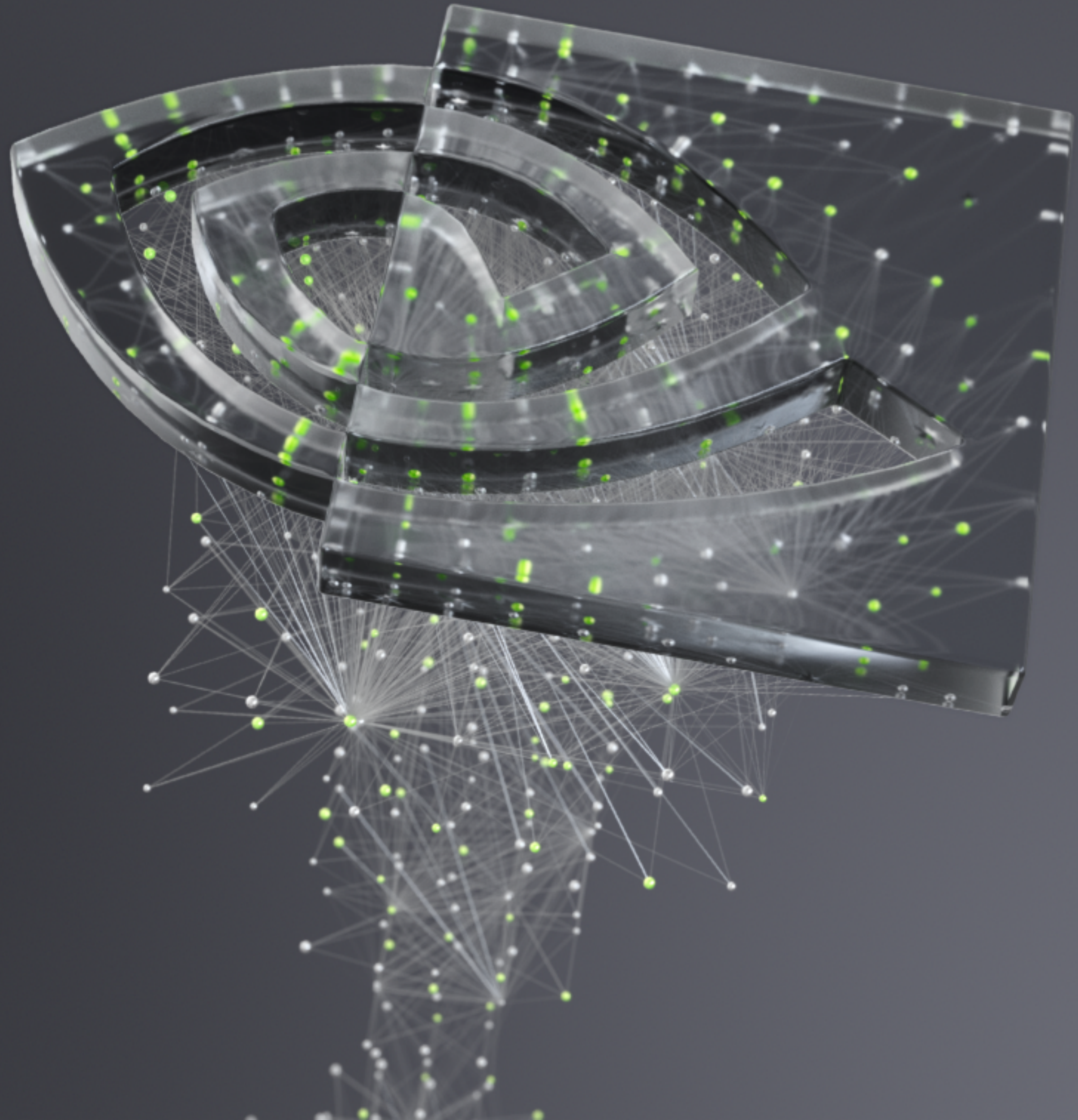
NVIDIA

DEEP
LEARNING
INSTITUTE

NVIDIA | DEEP LEARNING INSTITUTE

FUNDAMENTALS OF DEEP LEARNING

Part 5: Pre-trained Models



Part 1: An Introduction to Deep Learning

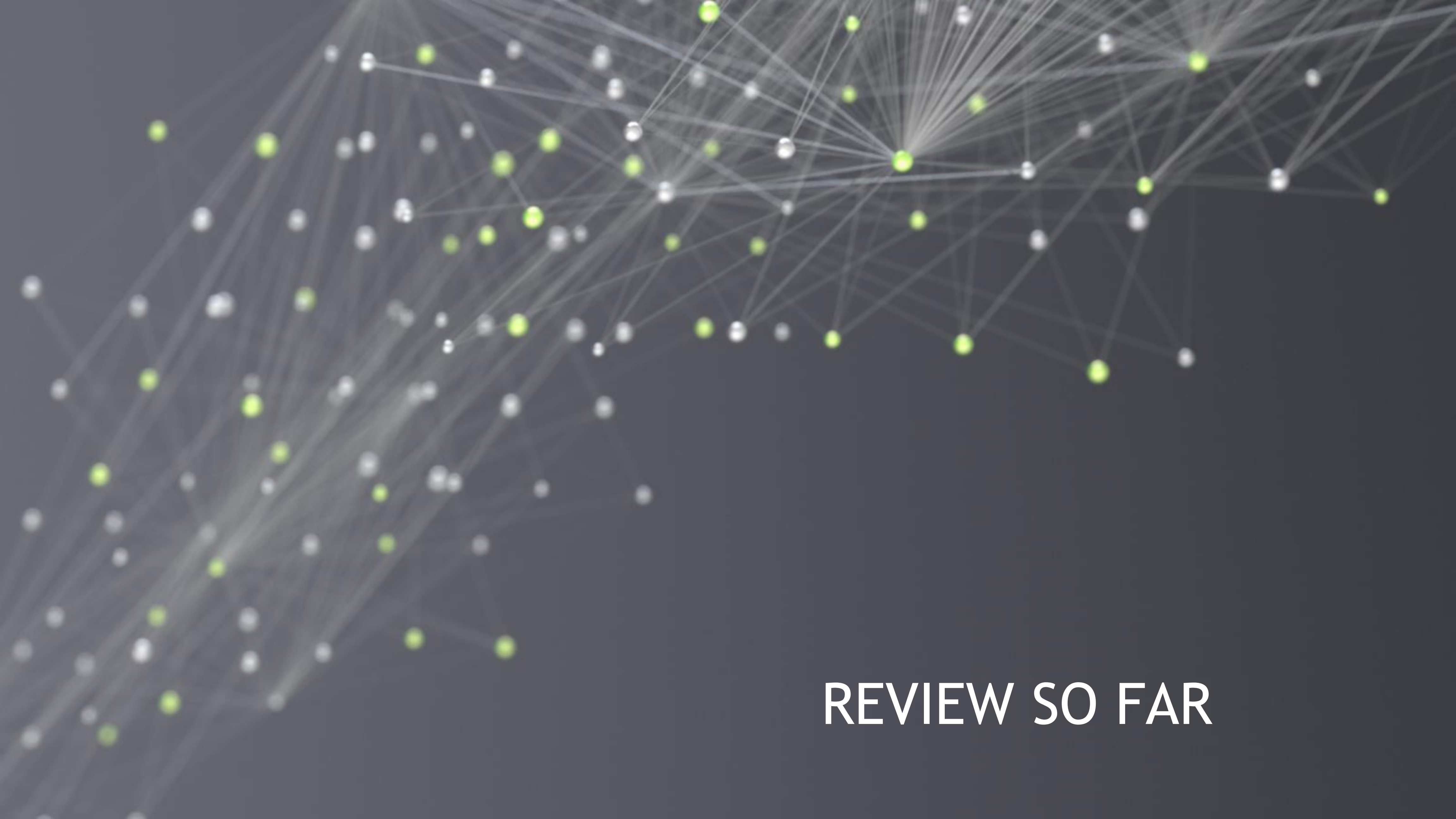
Part 2: How a Neural Network Trains

Part 3: Convolutional Neural Networks

Part 4: Data Augmentation and Deployment

Part 5: Pre-trained Models

Part 6: Advanced Architectures



REVIEW SO FAR

REVIEW SO FAR

- Learning Rate
- Number of Layers
- Neurons per Layer
- Activation Functions
- Dropout
- Data

PRE-TRAINED MODELS

PRE-TRAINED MODELS

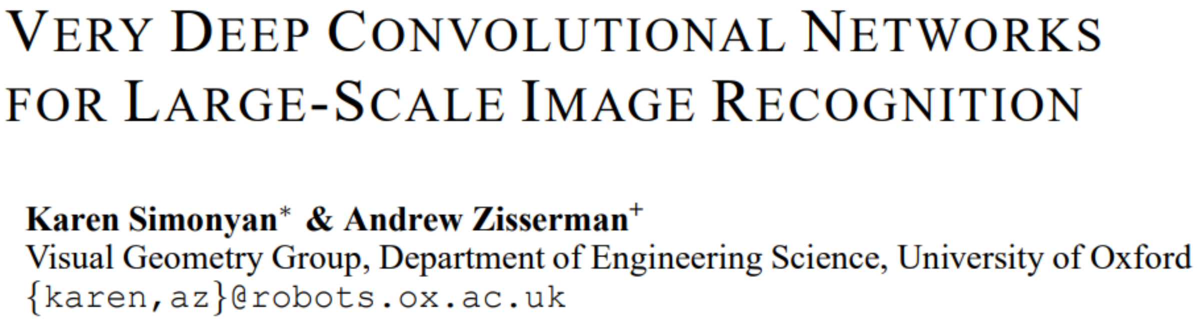
TensorFlow Hub

K Keras

NVIDIA. NGC

PYTORCH HUB

PRE-TRAINED MODELS



VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan[*] **& Andrew Zisserman**[+]
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

IMAGENET

THE NEXT CHALLENGE

An Automated Doggy Door

✓

✗

TRANSFER LEARNING

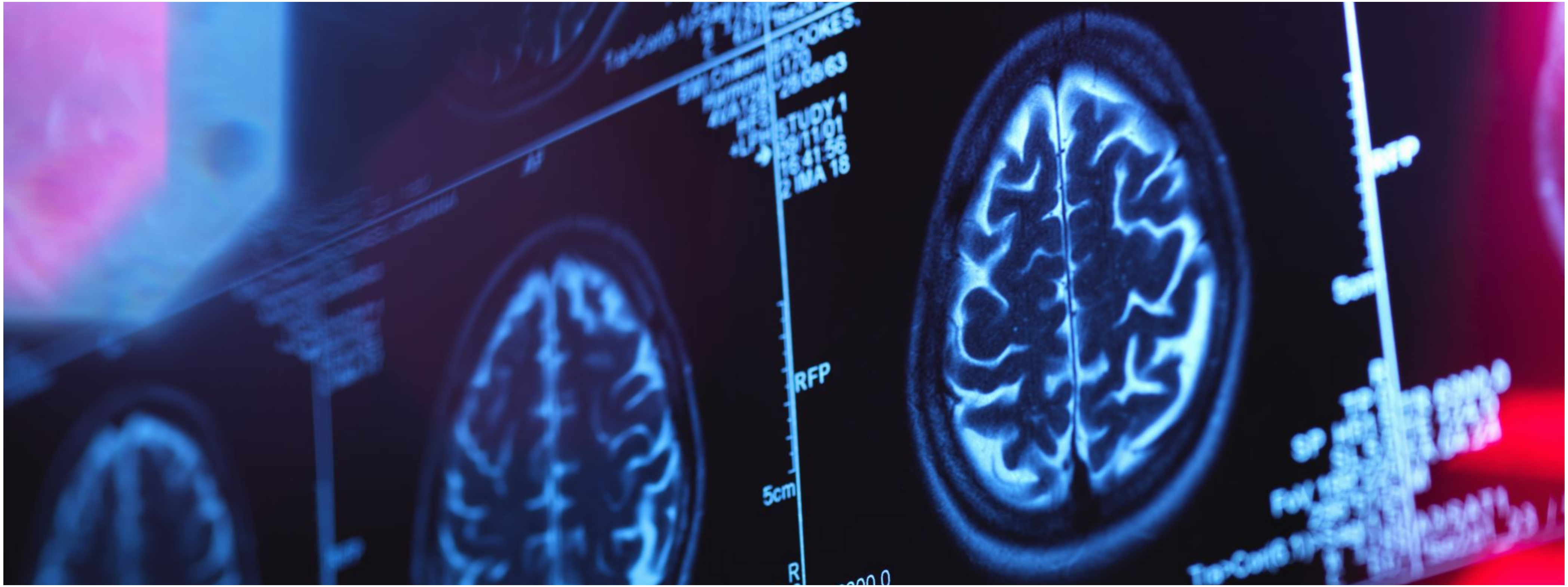
THE CHALLENGE AFTER

An Automated Presidential Doggy Door

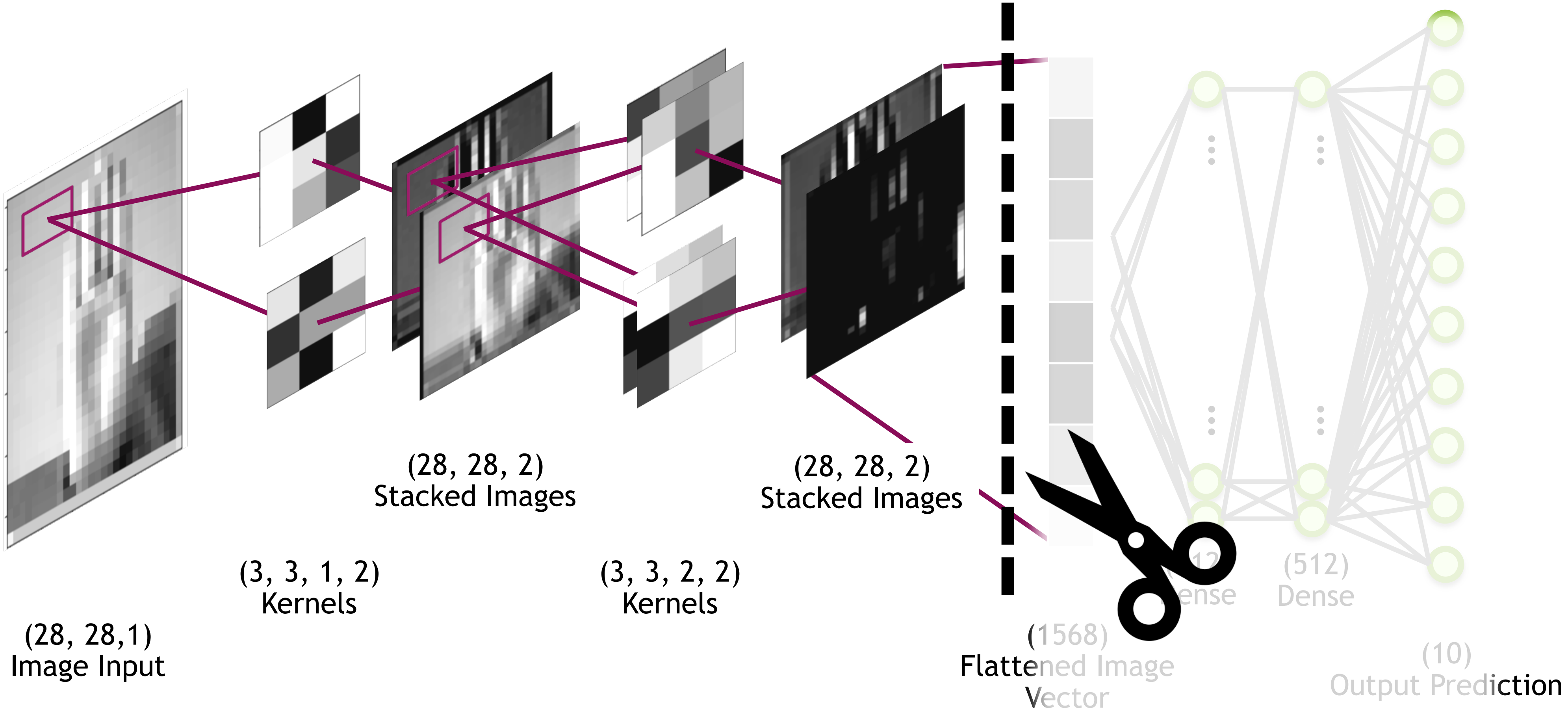
✓

✗

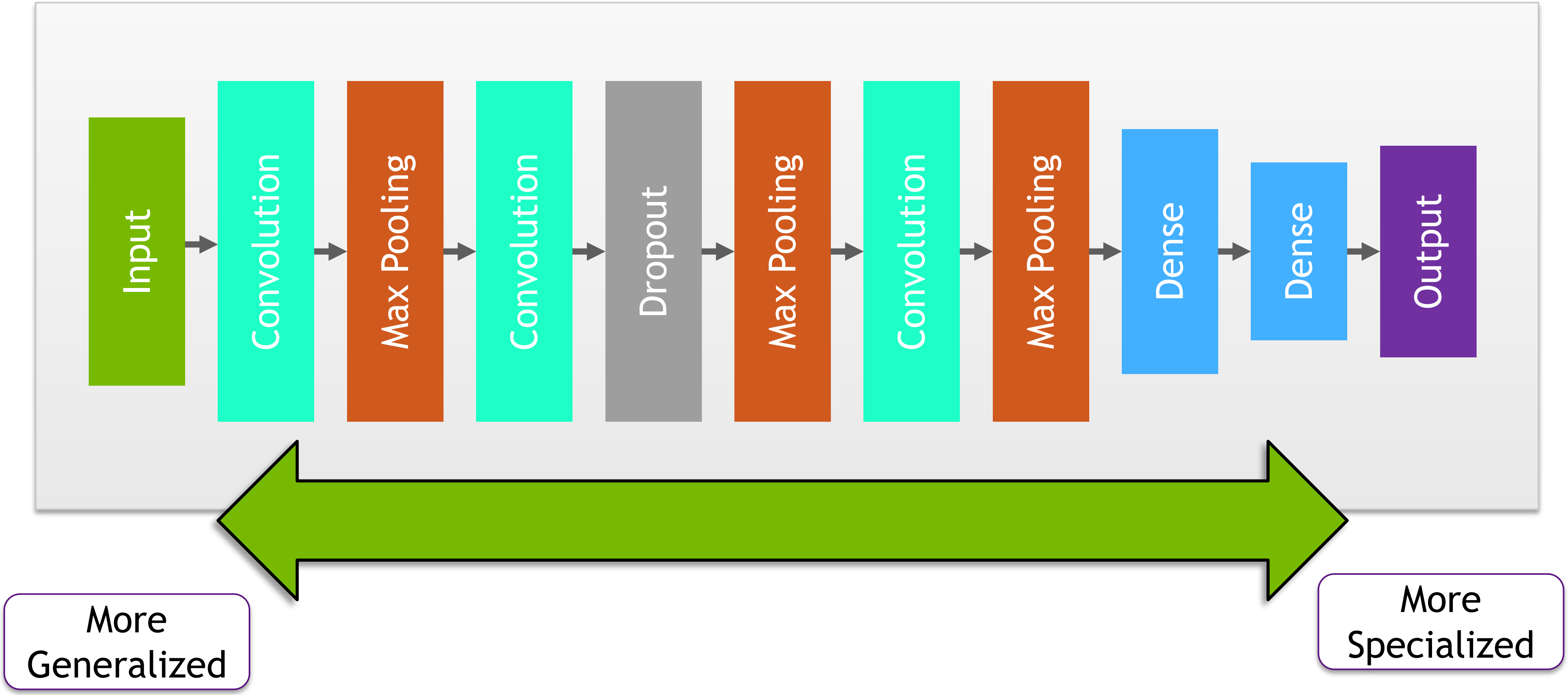
TRANSFER LEARNING



TRANSFER LEARNING



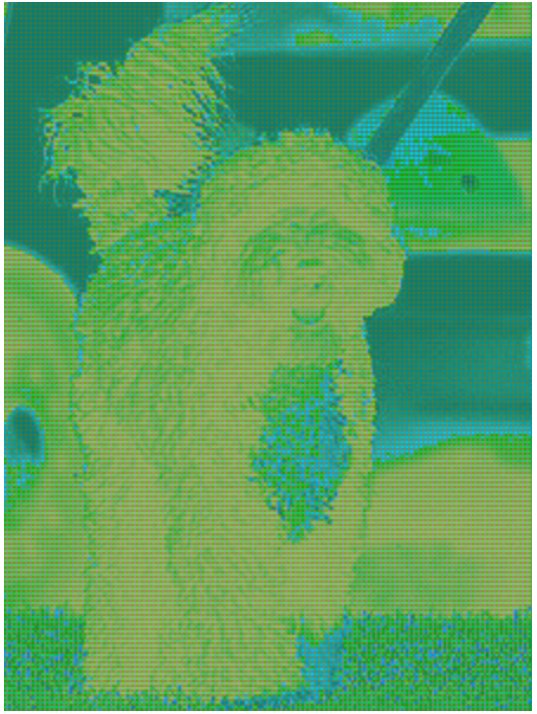
TRANSFER LEARNING

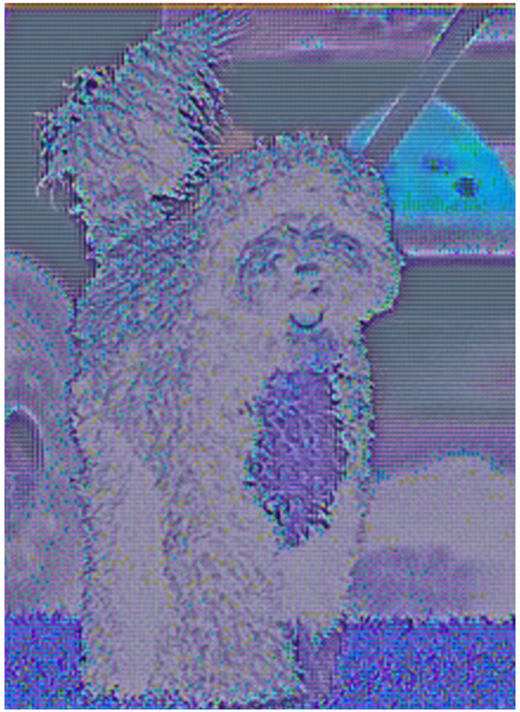


TRANSFER LEARNING

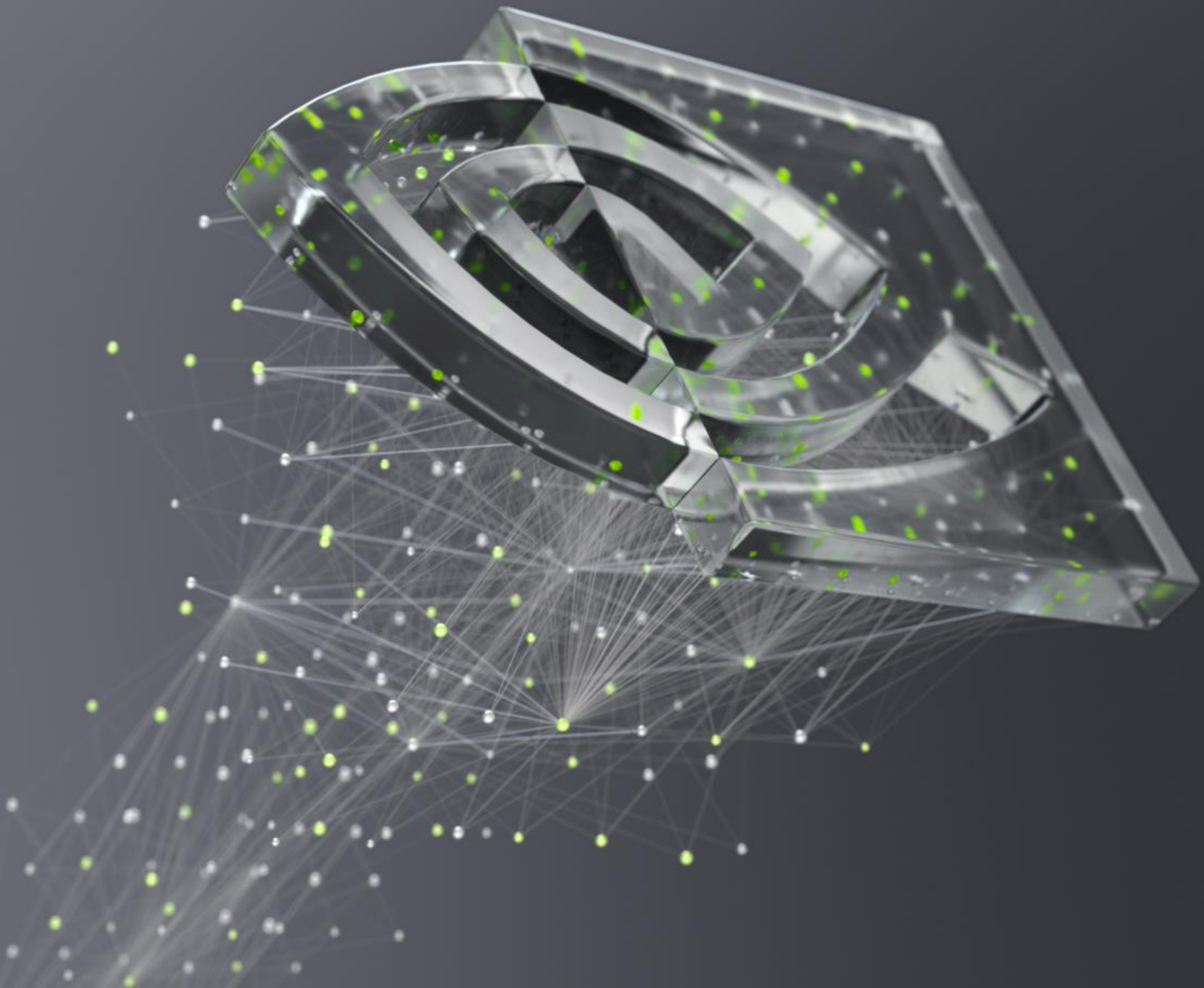
Freezing the Model?

TRANSFER LEARNING





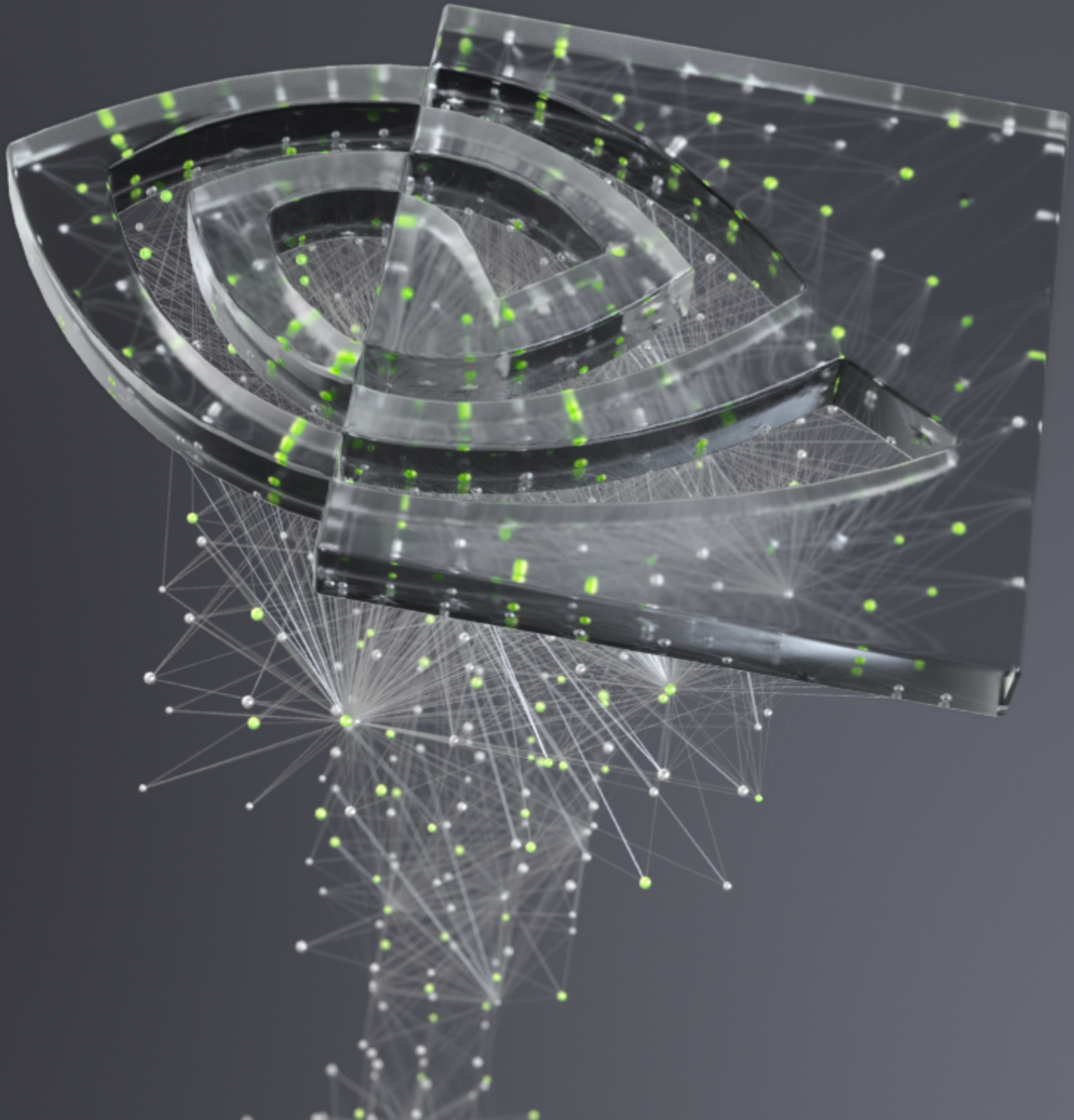
LET’S GET STARTED!



NVIDIA

DEEP
LEARNING
INSTITUTE

NVIDIA | DEEP LEARNING INSTITUTE



FUNDAMENTALS OF DEEP LEARNING

Part 6: Advanced Architectures

Part 1: An Introduction to Deep Learning

Part 2: How a Neural Network Trains

Part 3: Convolutional Neural Networks

Part 4: Data Augmentation and Deployment

Part 5: Pre-trained Models

Part 6: Advanced Architectures

MOVING FORWARD

FIELDS OF AI

Computer Vision

- Optometry

Natural Language Processing

- Linguistics

Reinforcement Learning

- Game Theory
- Psychology

Anomaly Detection

- Security
- Medicine

FIELDS OF AI

Computer Vision

- Optometry

Natural Language Processing

- Linguistics

Reinforcement Learning

- Game Theory
- Psychology

Anomaly Detection

- Security
- Medicine

FIELDS OF AI

Computer Vision

- Optometry

Natural Language Processing

- Linguistics

Reinforcement Learning

- Game Theory
- Psychology

Anomaly Detection

- Security
- Medicine

NATURAL LANGUAGE PROCESSING

FROM WORDS TO NUMBERS

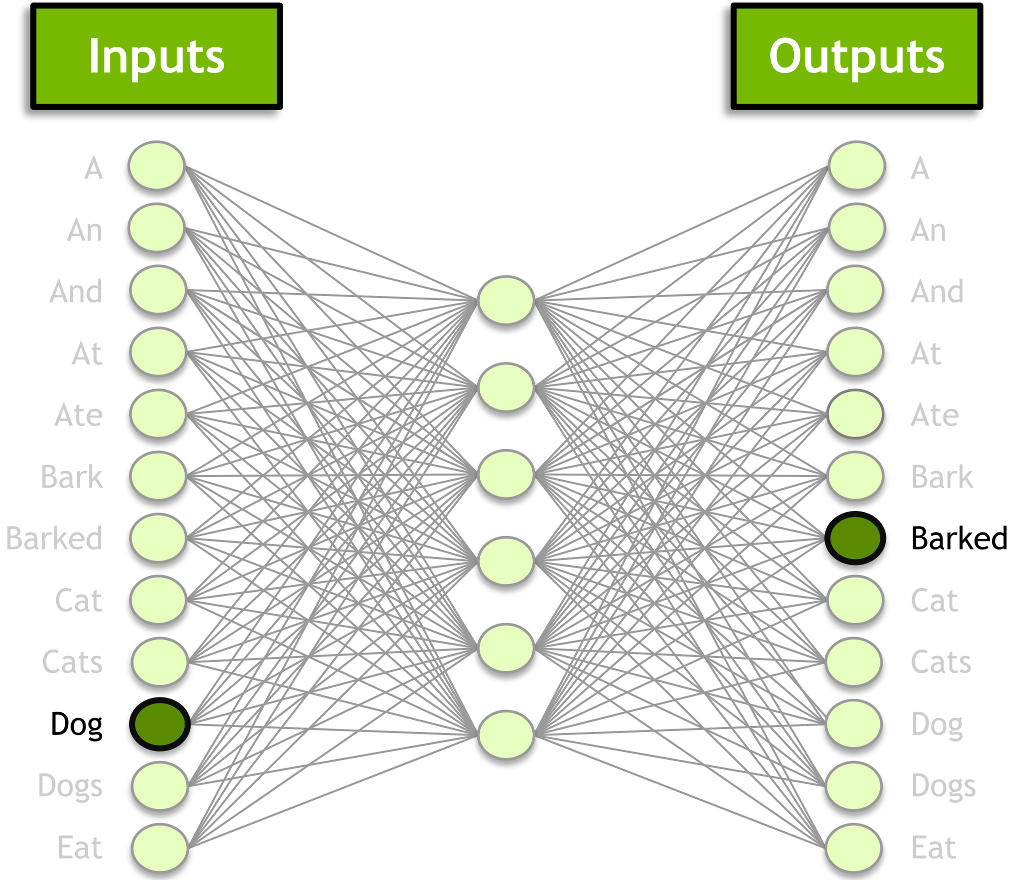
"A dog barked at a cat."

[1, 10, 7, 4, 1, 8]

Dictionary

1. A
2. An
3. And
4. At
5. Ate
6. Bark
7. Barked
8. Cat
9. Cats
10. Dog
11. Dogs
12. Eat

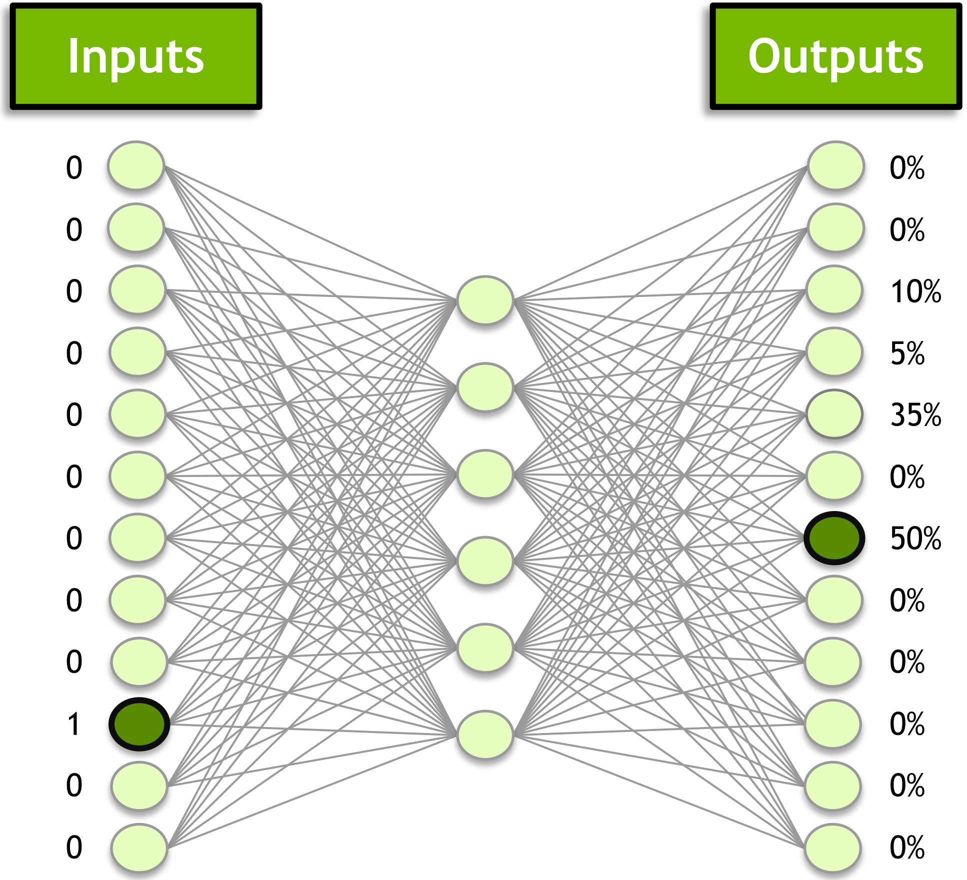
FROM WORDS TO NUMBERS



Dictionary

1. A
2. An
3. And
4. At
5. Ate
6. Bark
7. Barked
8. Cat
9. Cats
10. Dog
11. Dogs
12. Eat

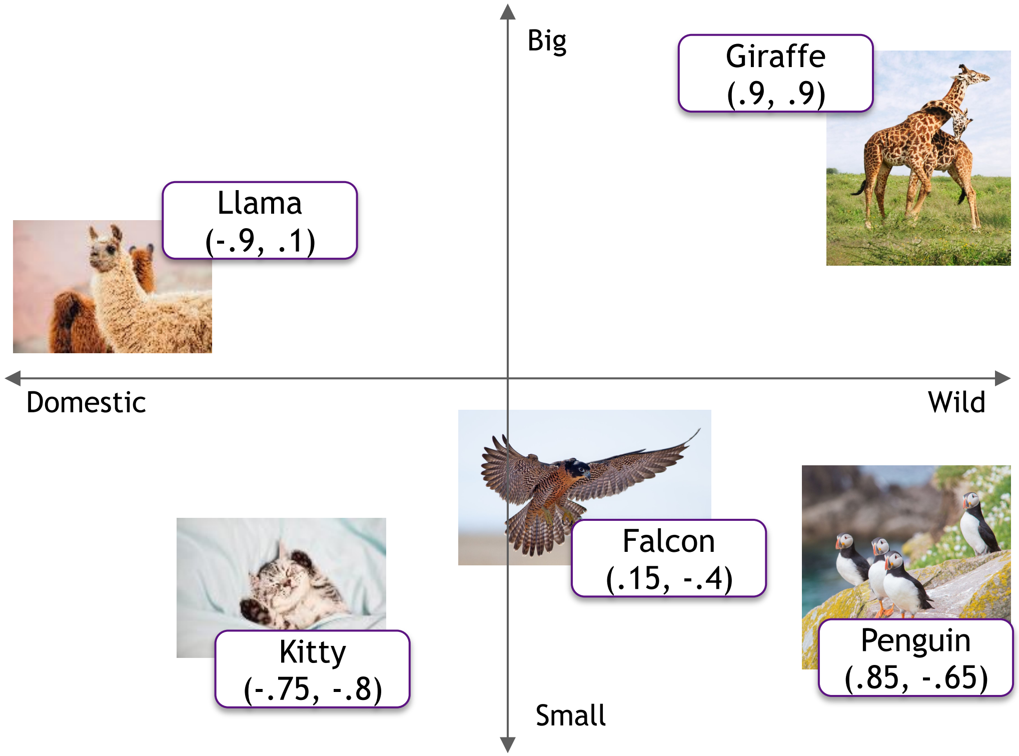
FROM WORDS TO NUMBERS



Dictionary

1. A
2. An
3. And
4. At
5. Ate
6. Bark
7. Barked
8. Cat
9. Cats
10. Dog
11. Dogs
12. Eat

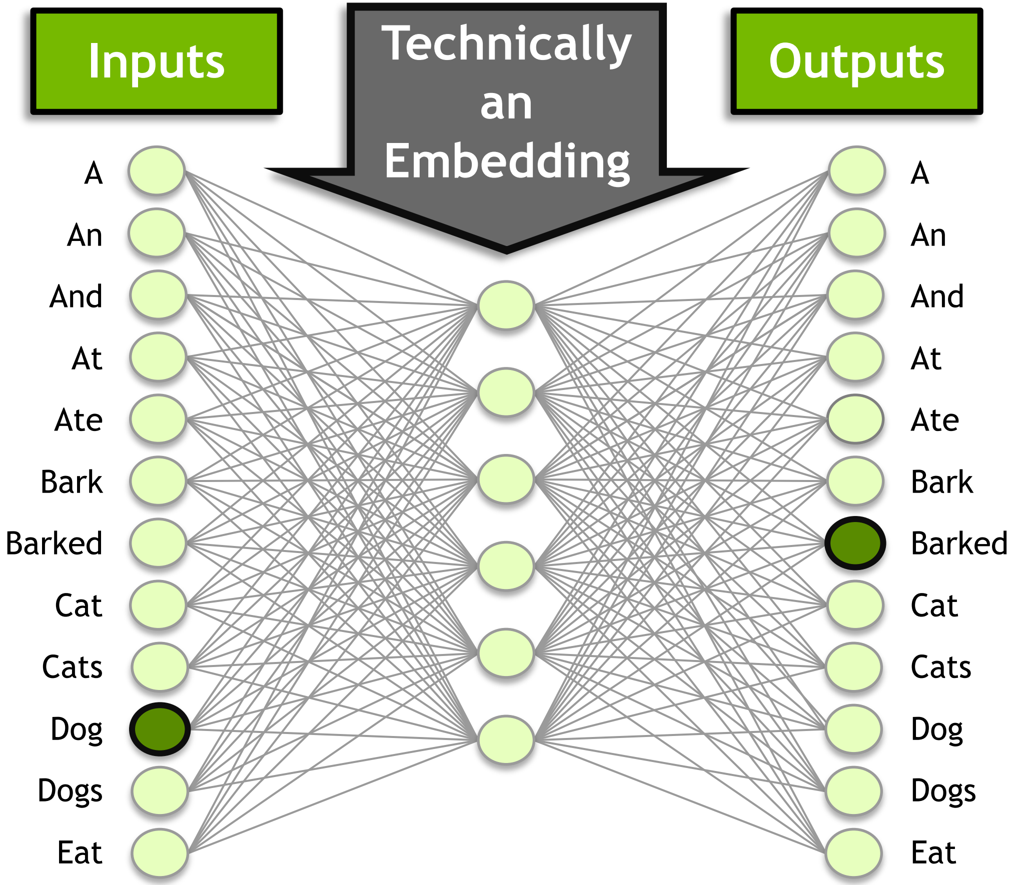
FROM WORDS TO NUMBERS



Bigger Dictionary

1.	A	31.	Ate	61.	Cats
2.	An	32.	Bark	62.	Dog
3.	And	33.	Barked	63.	Dogs
4.	At	34.	Cat	64.	Eat
5.	Ate	35.	Cats	65.	Eaten
6.	Bark	36.	Dog	66.	A
7.	Barked	37.	Dogs	67.	An
8.	Cat	38.	Eat	68.	And
9.	Cats	39.	Eaten	69.	At
10.	Dog	40.	A	70.	Ate
11.	Dogs	41.	An	71.	Bark
12.	Eat	42.	And	72.	Barked
13.	Eaten	43.	At	73.	Cat
14.	A	44.	Ate	74.	Cats
15.	An	45.	Bark	75.	Dog
16.	And	46.	Barked	76.	Dogs
17.	At	47.	Cat	77.	Eat
18.	Ate	48.	Cats	78.	Eaten
19.	Bark	49.	Dog	79.	…
20.	Barked	50.	Dogs	80.	…
21.	Cat	51.	Eat	81.	…
22.	Cats	52.	Eaten	82.	…
23.	Dog	53.	A		
24.	Dogs	54.	An		
25.	Eat	55.	And		
26.	Eaten	56.	At		
27.	A	57.	Ate		
28.	An	58.	Bark		
29.	And	59.	Barked		
30.	At	60.	Cat		

FROM WORDS TO NUMBERS



Inputs

Technically an Embedding

Outputs

Dictionary

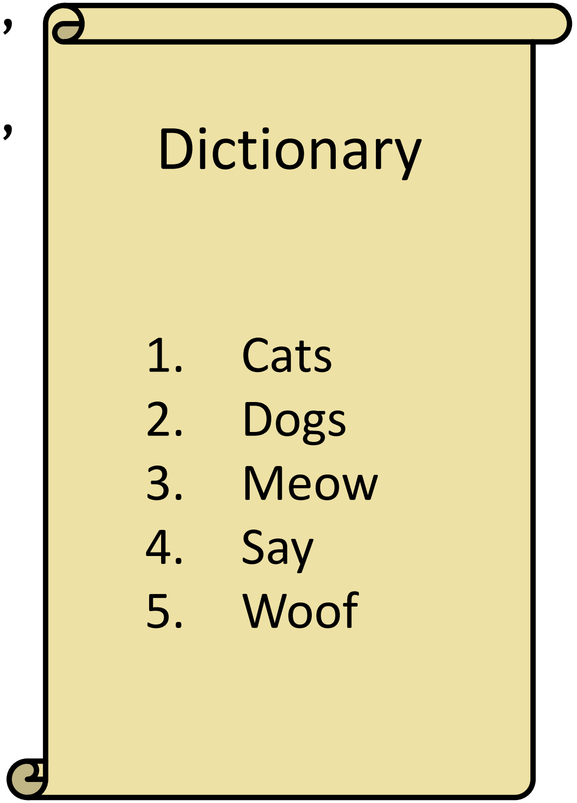
1. A
2. An
3. And
4. At
5. Ate
6. Bark
7. Barked
8. Cat
9. Cats
10. Dog
11. Dogs
12. Eat

RECURRENT NEURAL NETWORKS

RECURRENT NEURAL NETWORKS

“Cats say ____.”

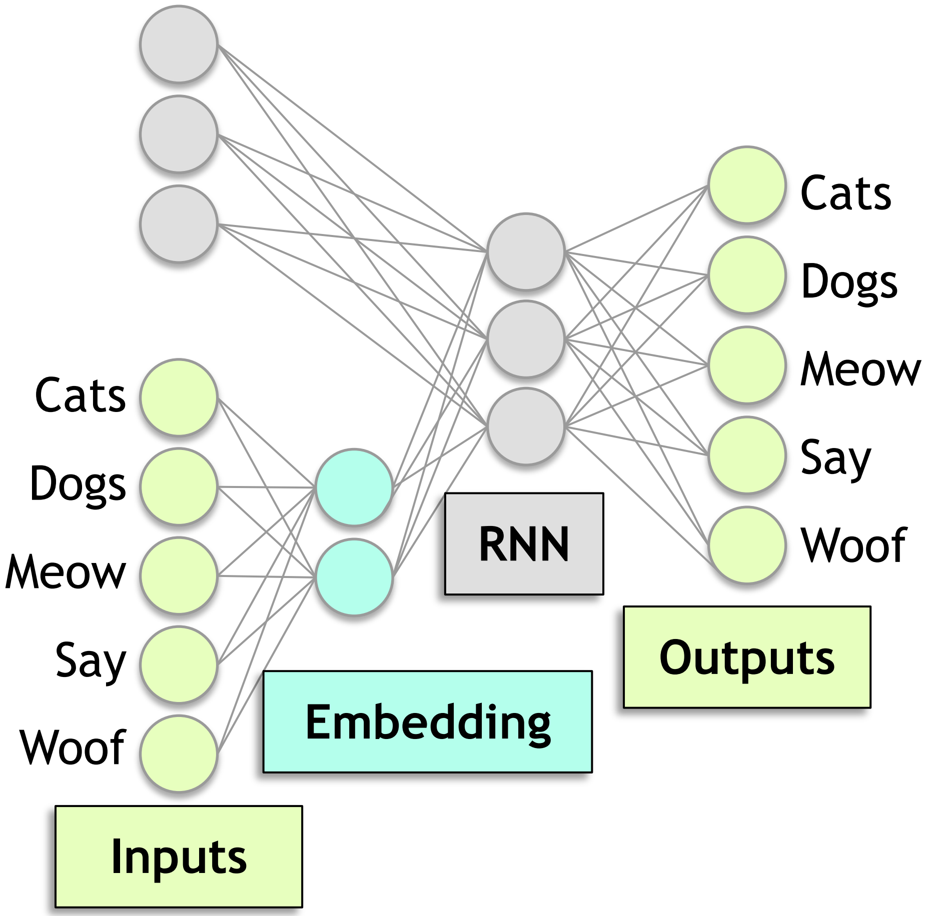
“Dogs say ____.”



Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



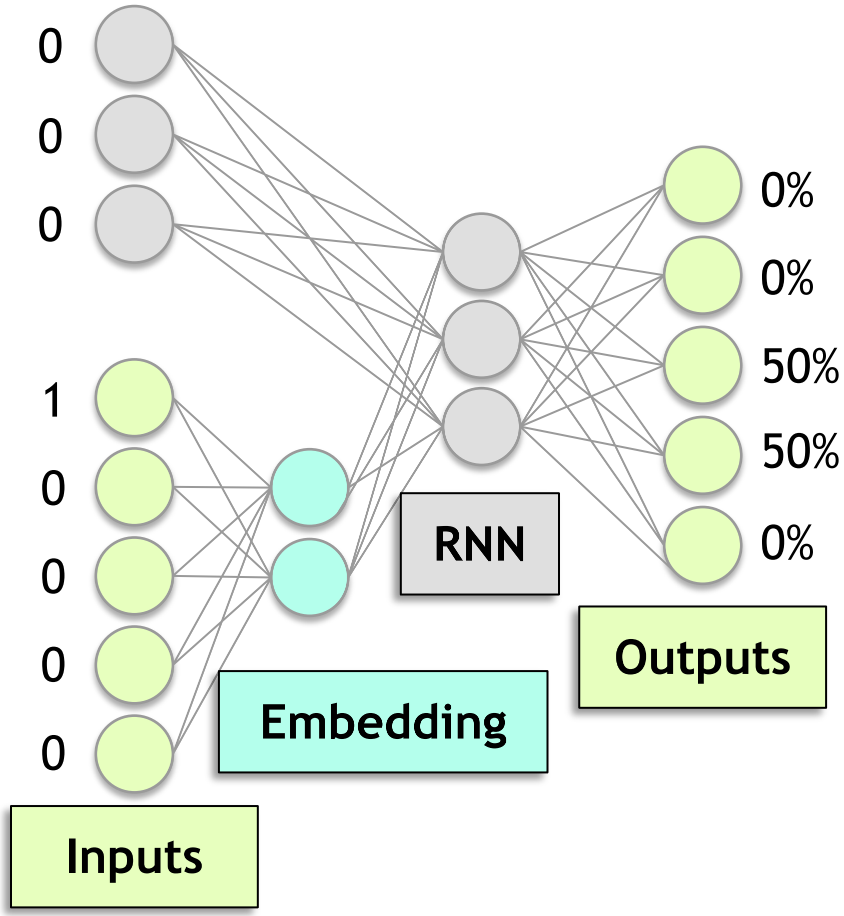
"Cats say ____."

"Dogs say ____."

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



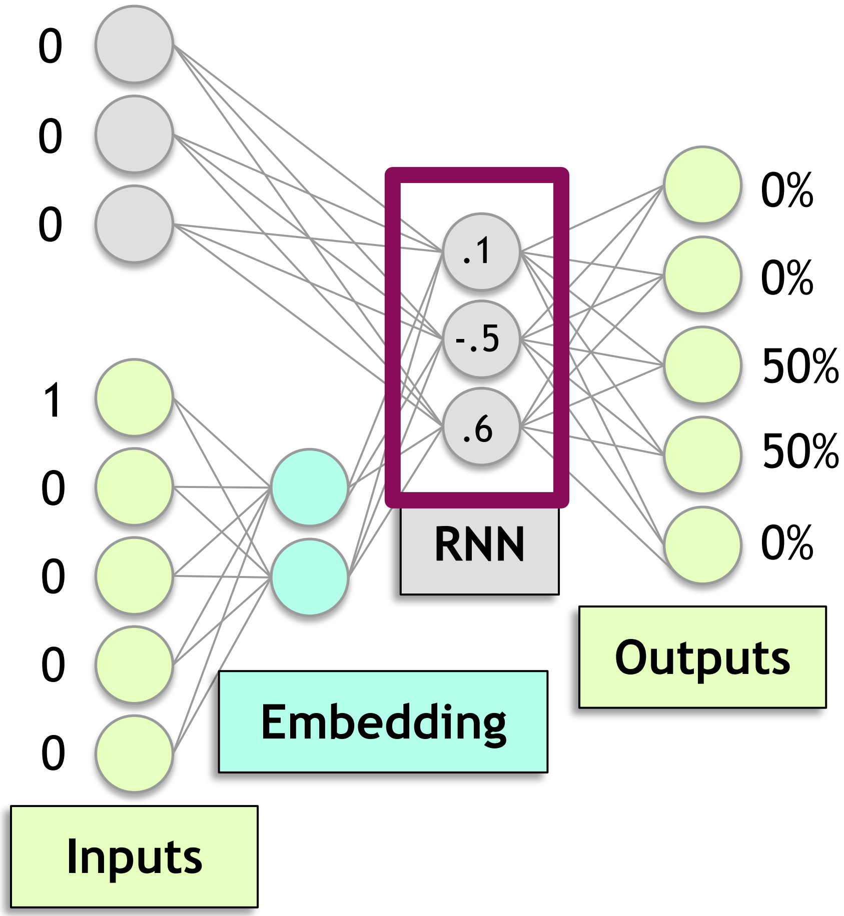
“Cats say ____.”

“Dogs say ____.”

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



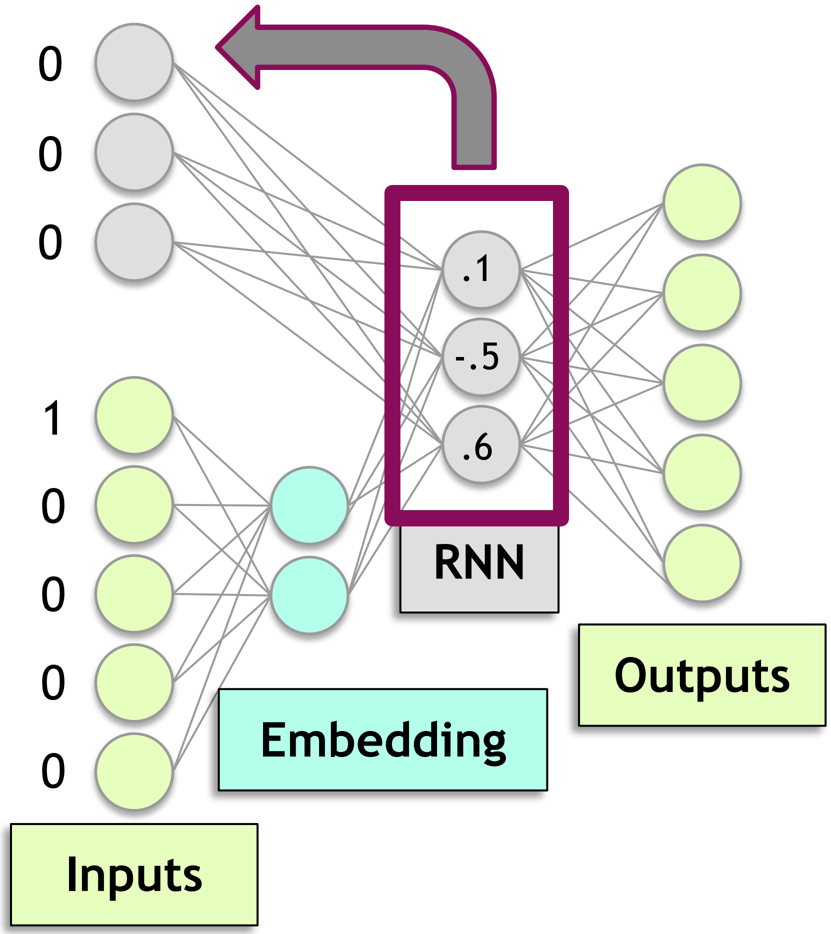
“Cats say ____.”

“Dogs say ____.”

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



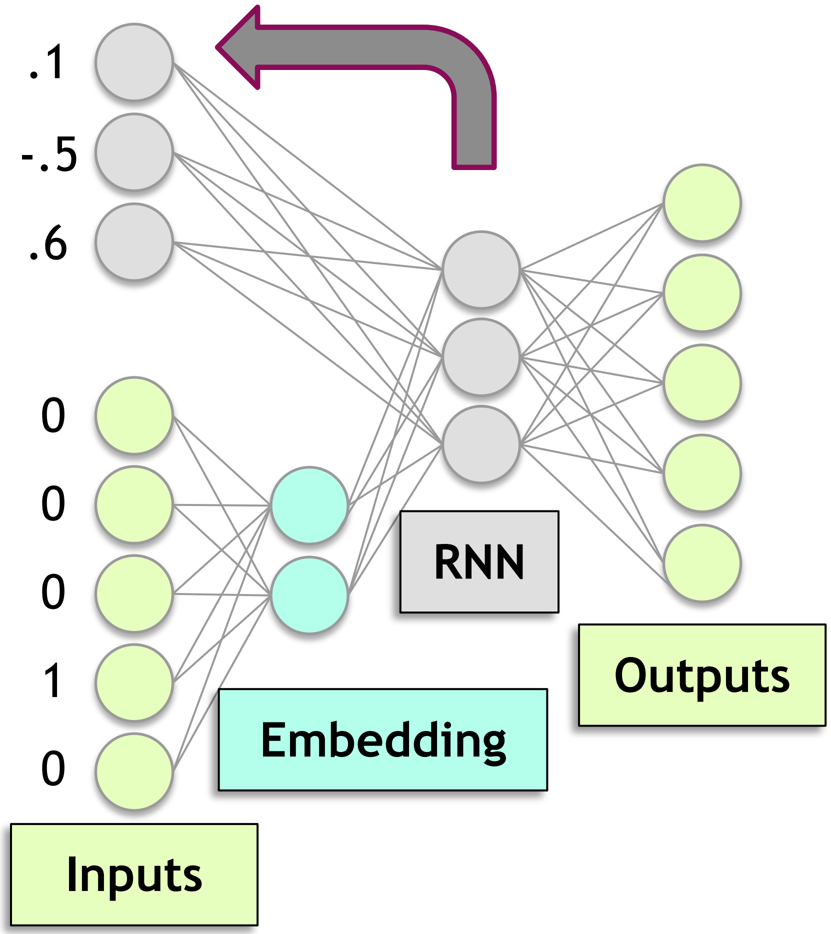
“Cats say ____.”

“Dogs say ____.”

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



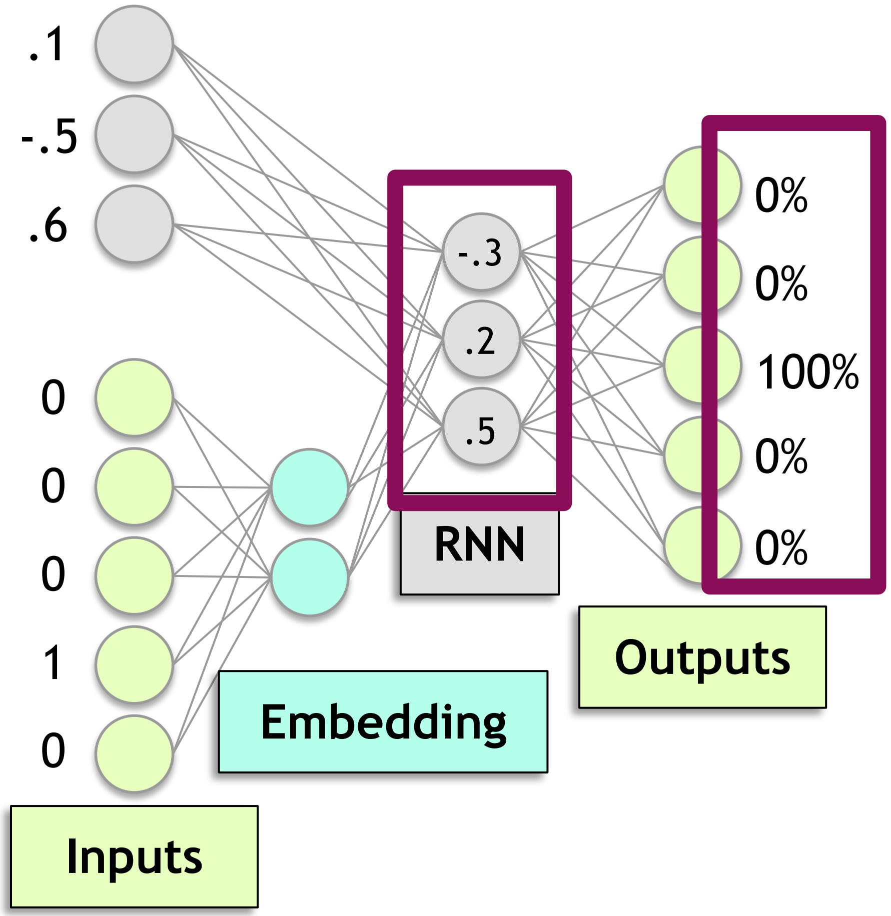
"Cats say ____."

"Dogs say ____."

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

RECURRENT NEURAL NETWORKS



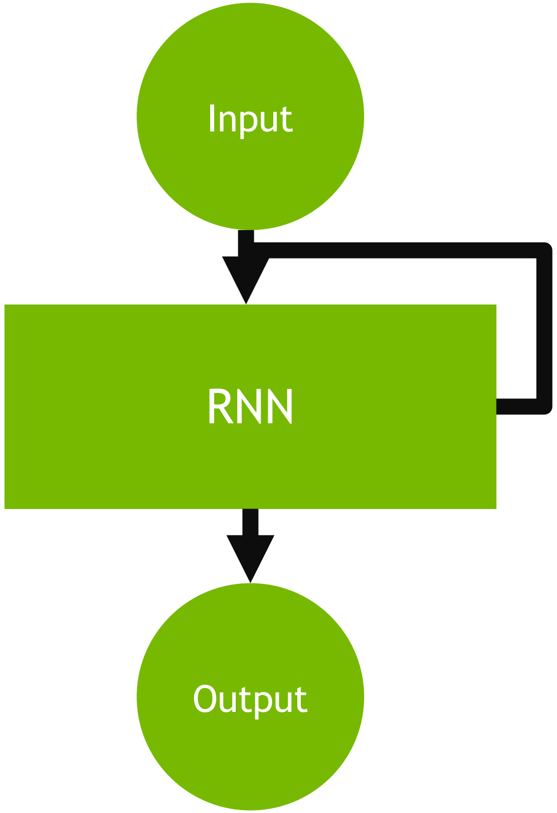
"Cats say ____."

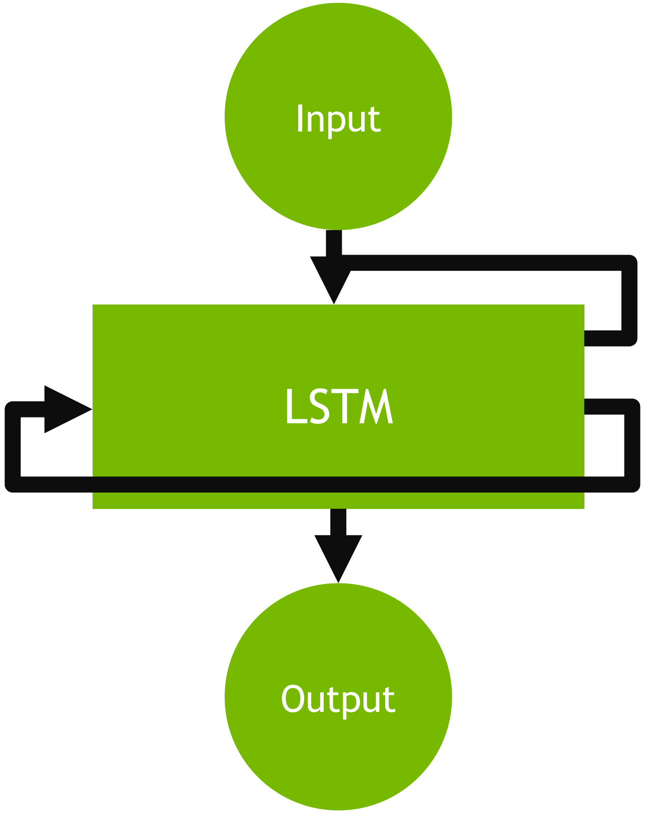
"Dogs say ____."

Dictionary

1. Cats
2. Dogs
3. Meow
4. Say
5. Woof

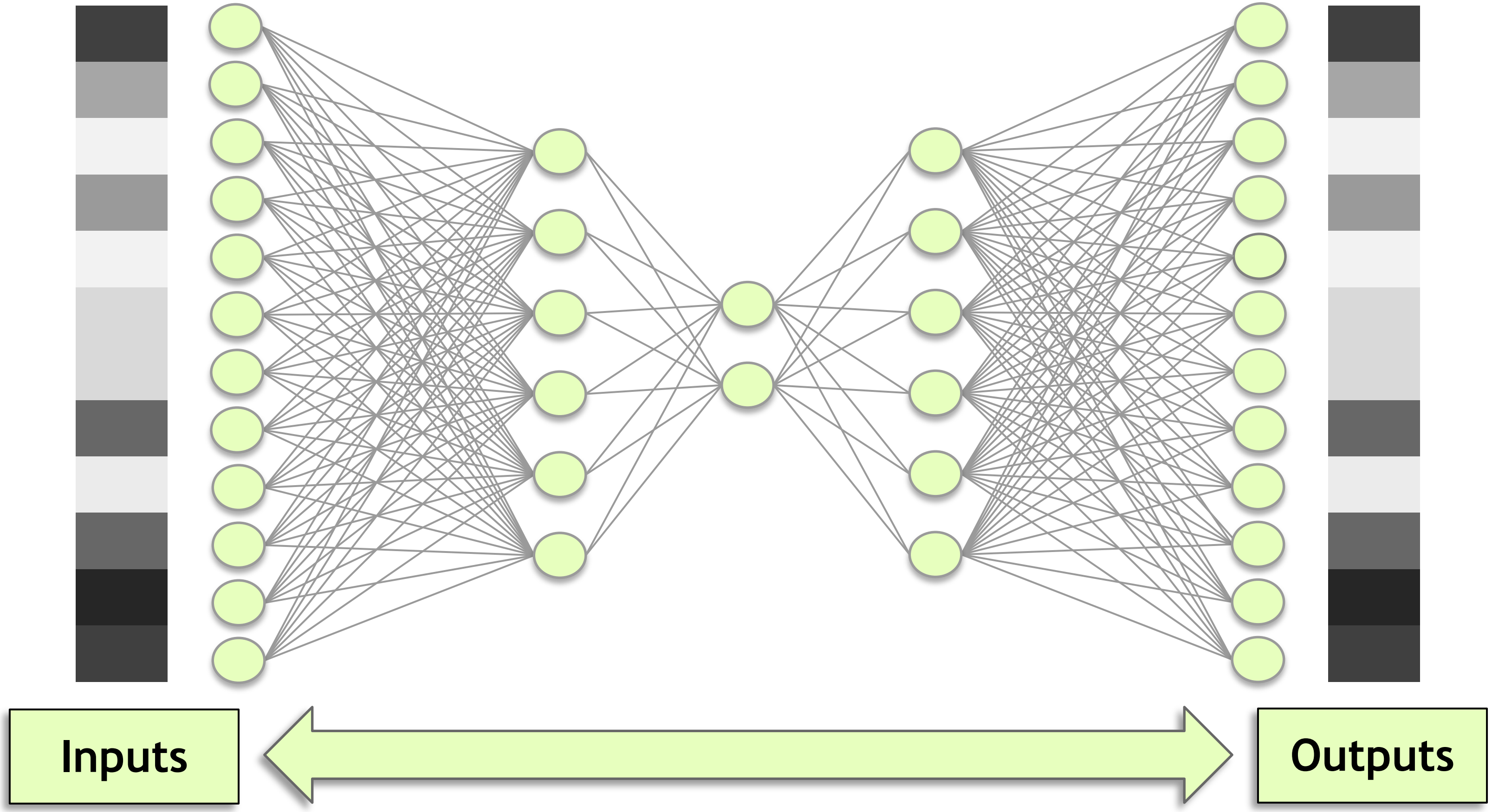
RECURRENT NEURAL NETWORKS





OTHER ARCHITECTURES

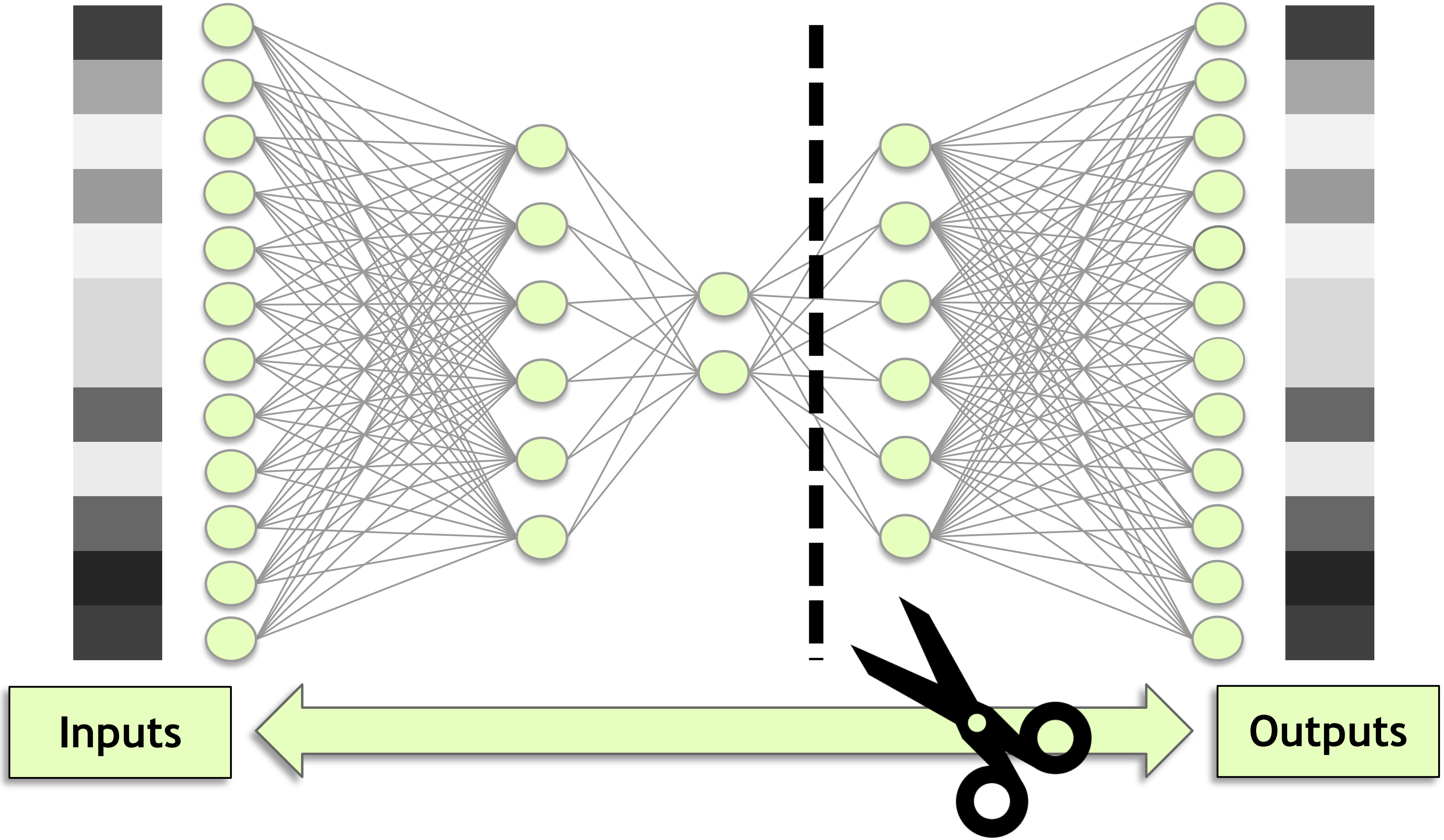
AUTOENCODERS



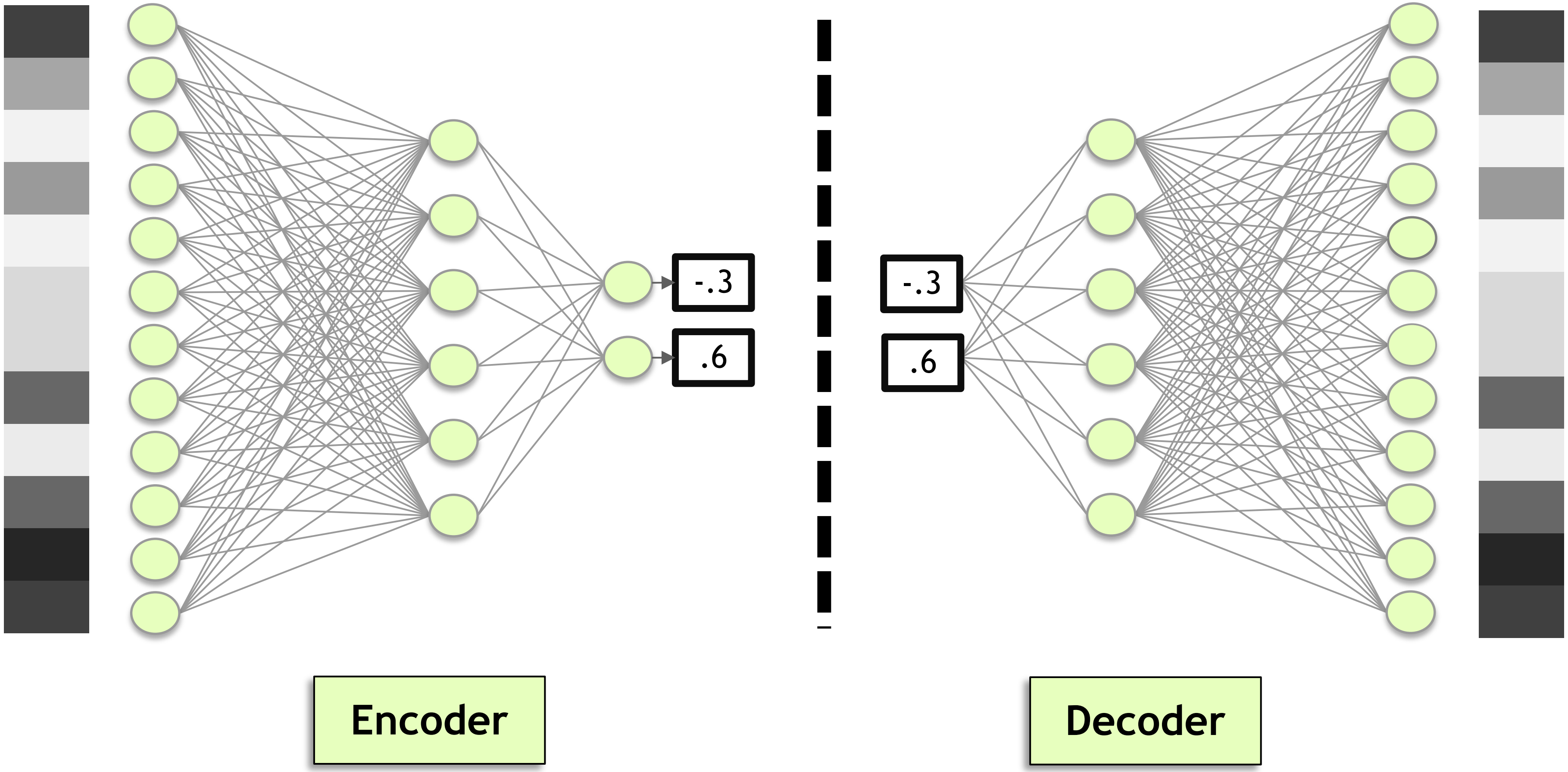
Inputs

Outputs

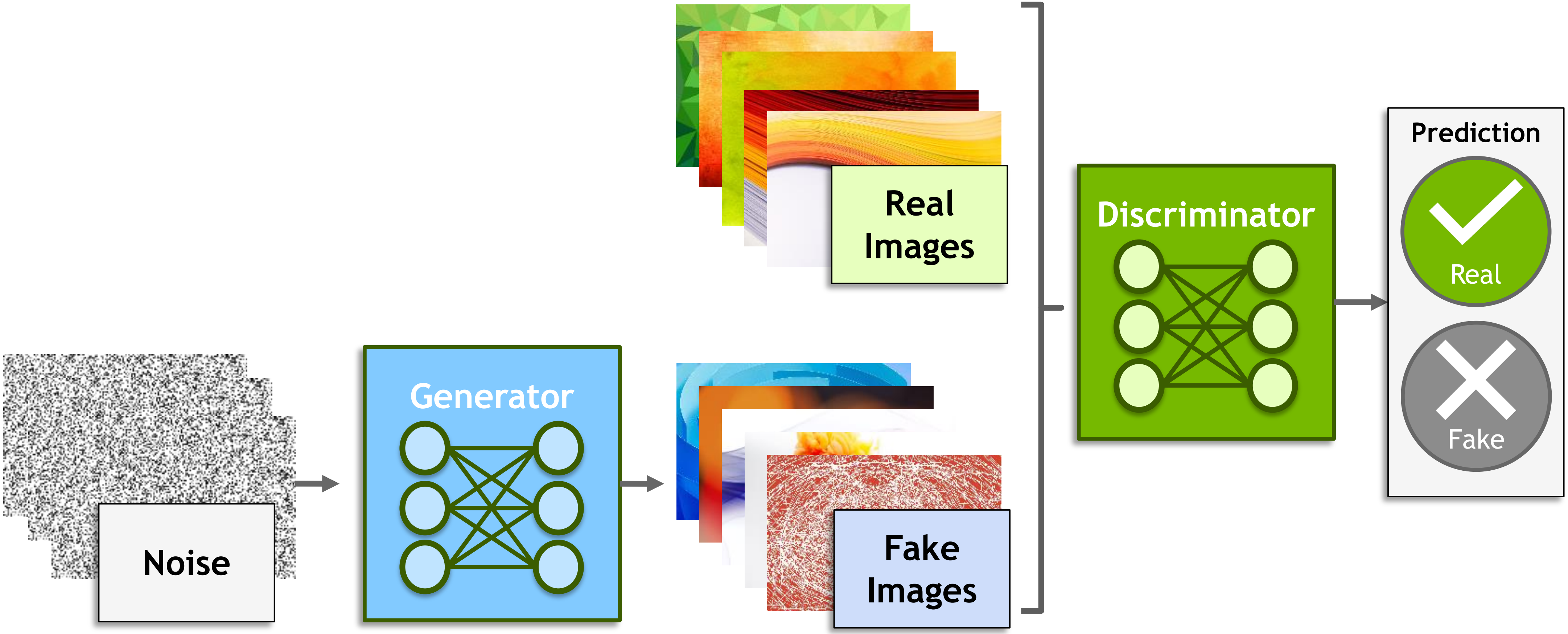
AUTOENCODERS



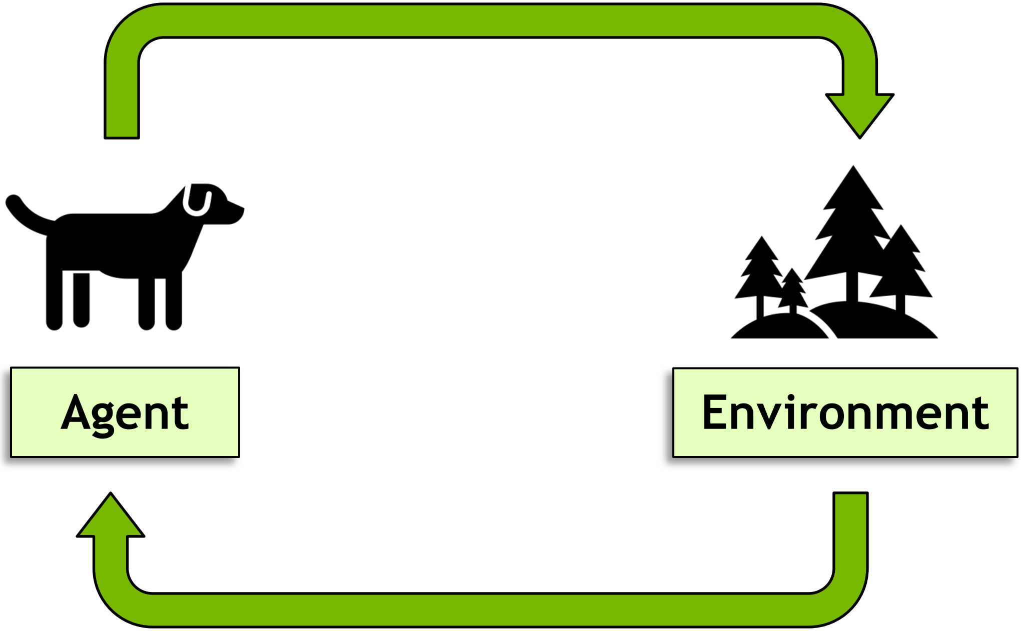
AUTOENCODERS



GENERATIVE ADVERSARIAL NETWORKS (GANS)



REINFORCEMENT LEARNING



NEXT STEPS

ENABLING PORTABILITY WITH NGC CONTAINERS

Extensive

- Diverse range of workloads and industry specific use cases

Optimized

- DL containers updated monthly
- Packed with latest features and superior performance

Secure & Reliable

- Scanned for vulnerabilities and crypto
- Tested on workstations, servers, & cloud instances

Scalable

- Supports multi-GPU & multi-node systems

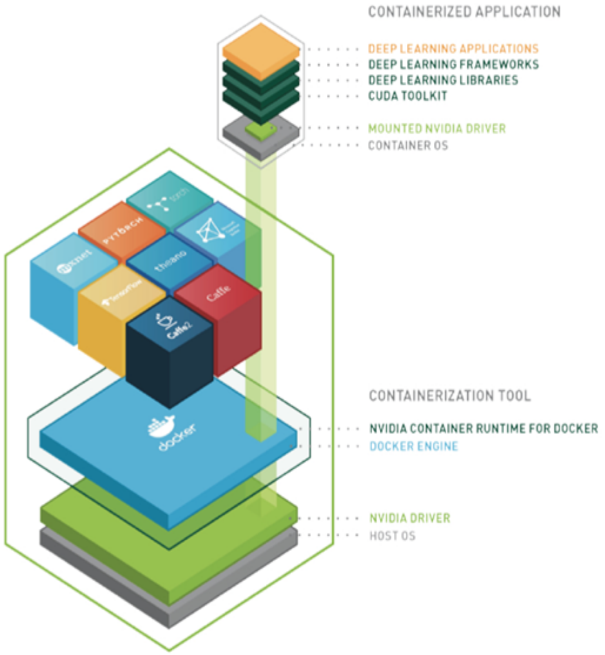
Designed for Enterprise & HPC

- Supports Docker, Singularity & other runtimes

Run Anywhere

- Bare metal, VMs, Kubernetes
- x86, ARM, POWER
- Multi-cloud, on-prem, hybrid, edge

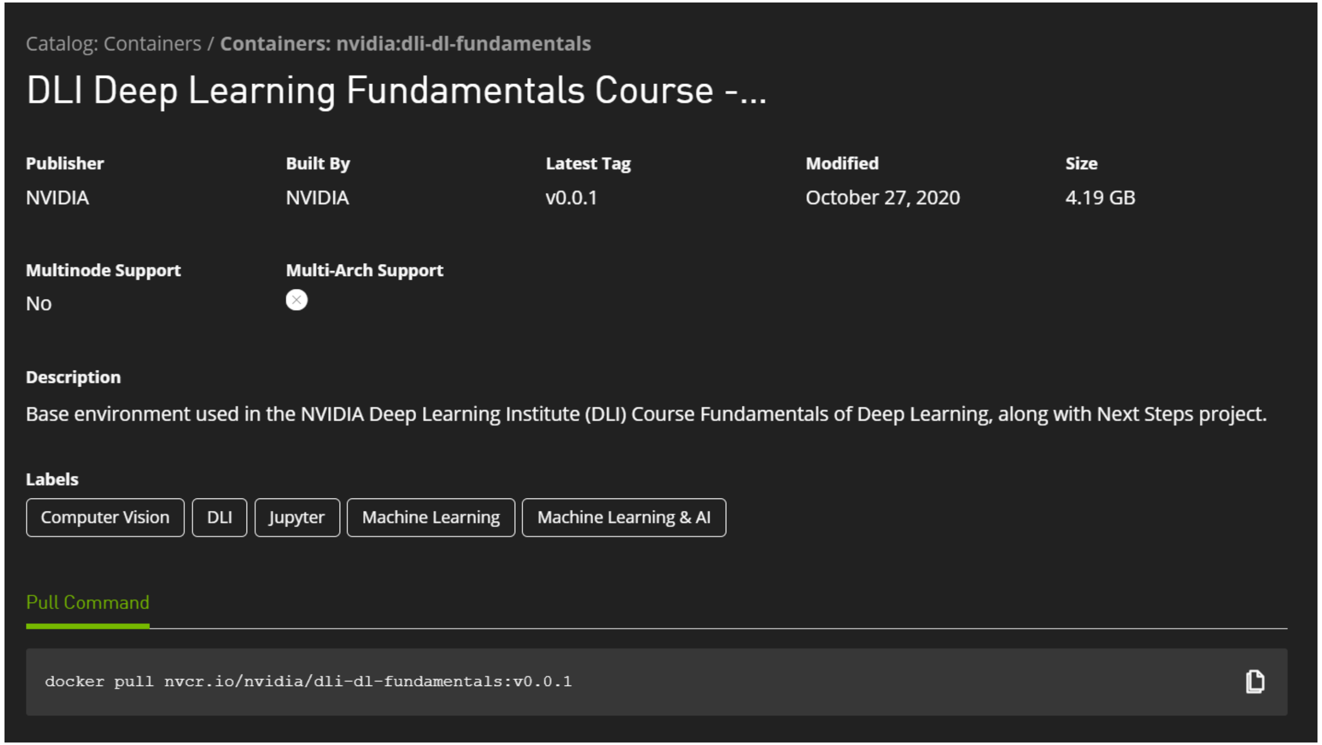
NGC Deep Learning Containers



CONVERSATIONAL AI	HEALTHCARE	SMART CITIES	TELECOM	AUTONOMOUS DRIVING	ROBOTICS	HPC
JARVIS	CLARA	DEEPSTREAM & SMART PARKING	AERIAL	DRIVE	ISAAC	HPC SDK

Learn more about NGC Containers

NEXT STEPS FOR THIS CLASS



Catalog: Containers / **Containers: nvidia:dli-dl-fundamentals**

DLI Deep Learning Fundamentals Course -...

Publisher	Built By	Latest Tag	Modified	Size
NVIDIA	NVIDIA	v0.0.1	October 27, 2020	4.19 GB

Multinode Support	Multi-Arch Support
No	☒

Description

Base environment used in the NVIDIA Deep Learning Institute (DLI) Course Fundamentals of Deep Learning, along with Next Steps project.

Labels

```
docker pull nvcr.io/nvidia/dli-dl-fundamentals:v0.0.1
```

Step 1 Setup Docker

https://www.docker.com/

Step 2 Visit NGC Catalog

https://ngc.nvidia.com/catalog/containers/nvidia:dli-dl-fundamentals

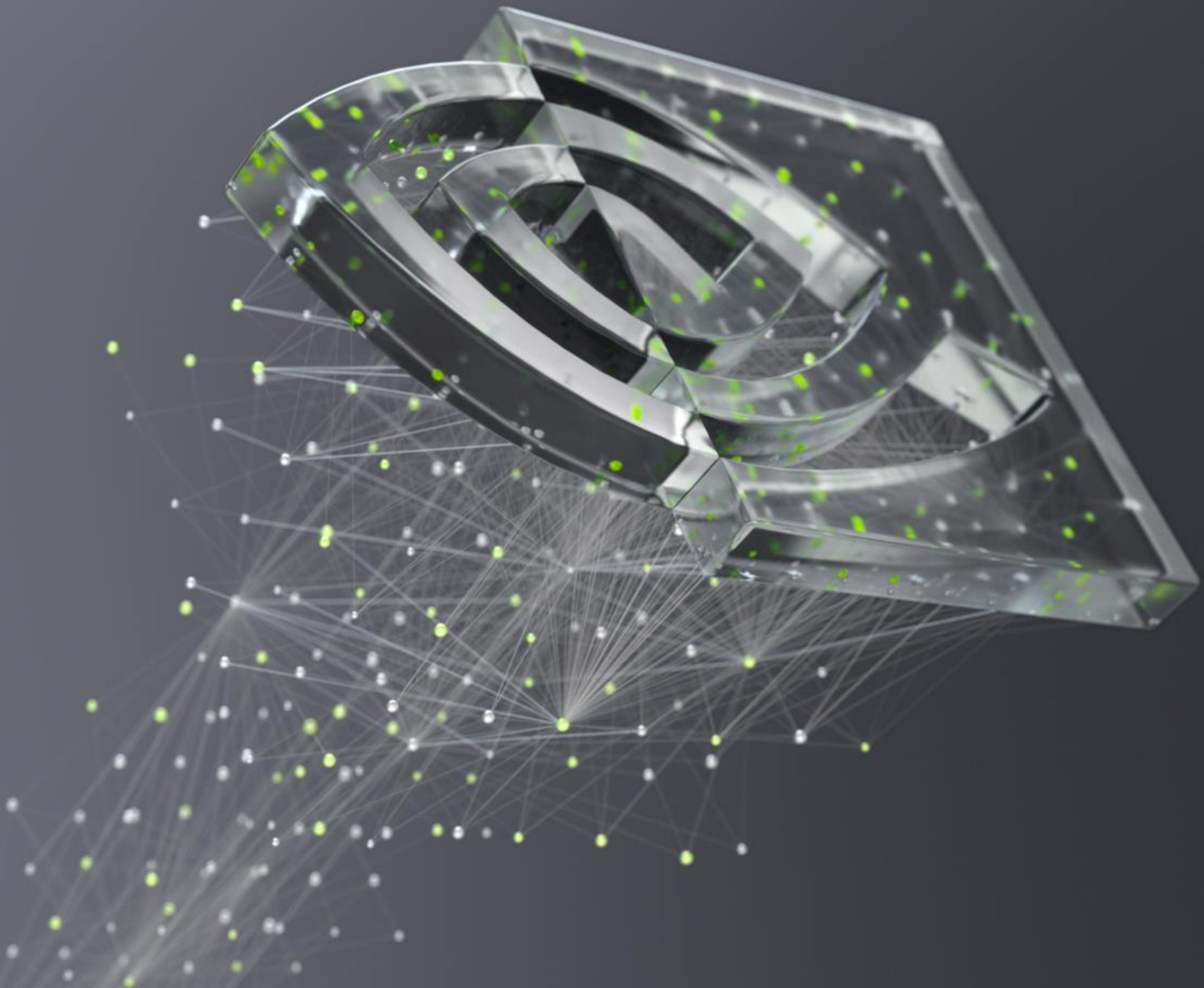
Step 3 Pull and Run Container

Visit localhost:8888 to check out a JupyterLab environment with a Next Steps Project

CLOSING THOUGHTS

COPYING ROCKET SCIENCE

LET’S GET STARTED!



NVIDIA

DEEP
LEARNING
INSTITUTE