Deep Learning for Robotic Vision

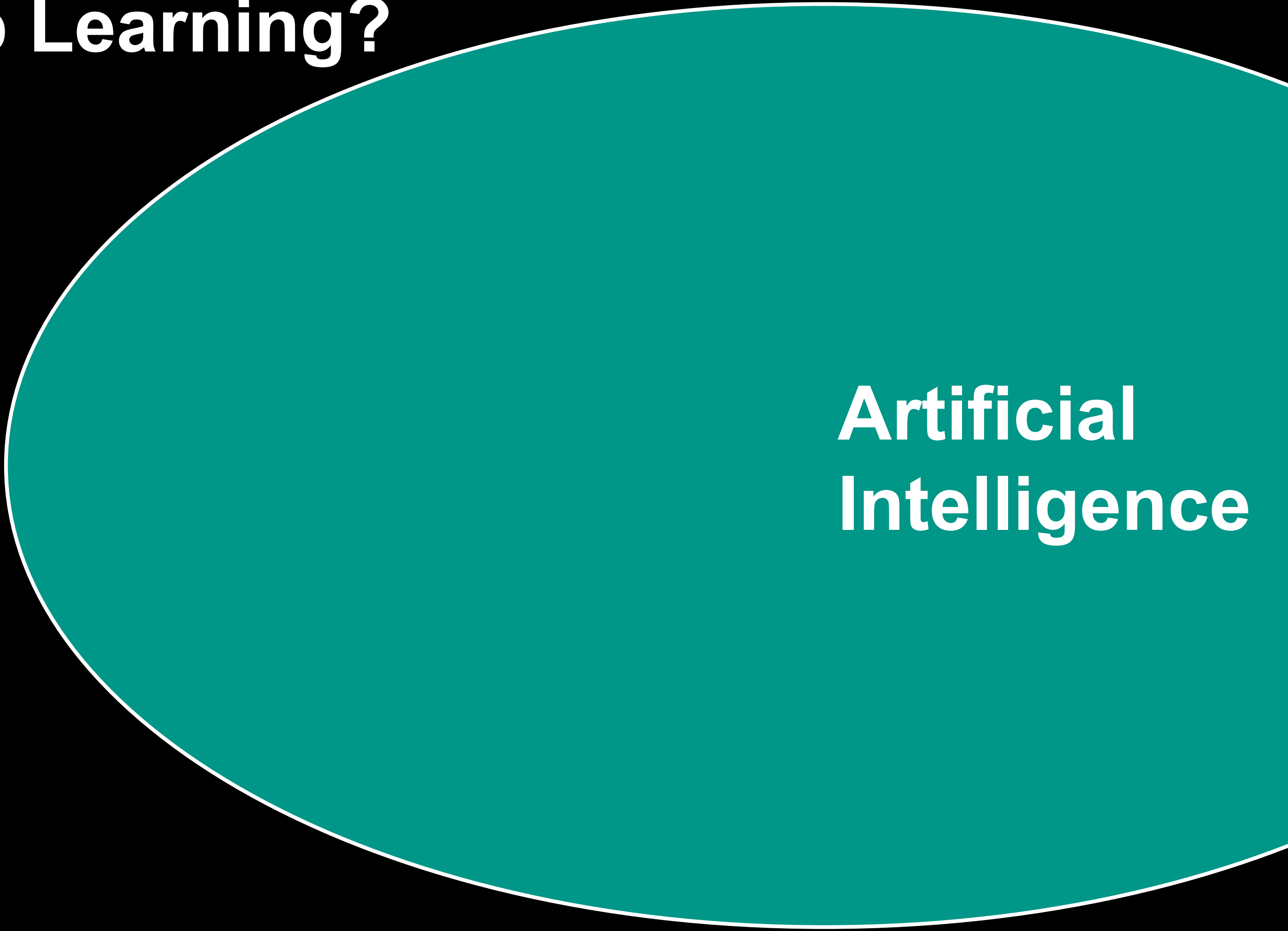
An Introduction

Niko Suenderhauf

Queensland University of Technology
Australian Centre for Robotic Vision

What is Deep Learning?

What is Deep Learning?



What is Deep Learning?

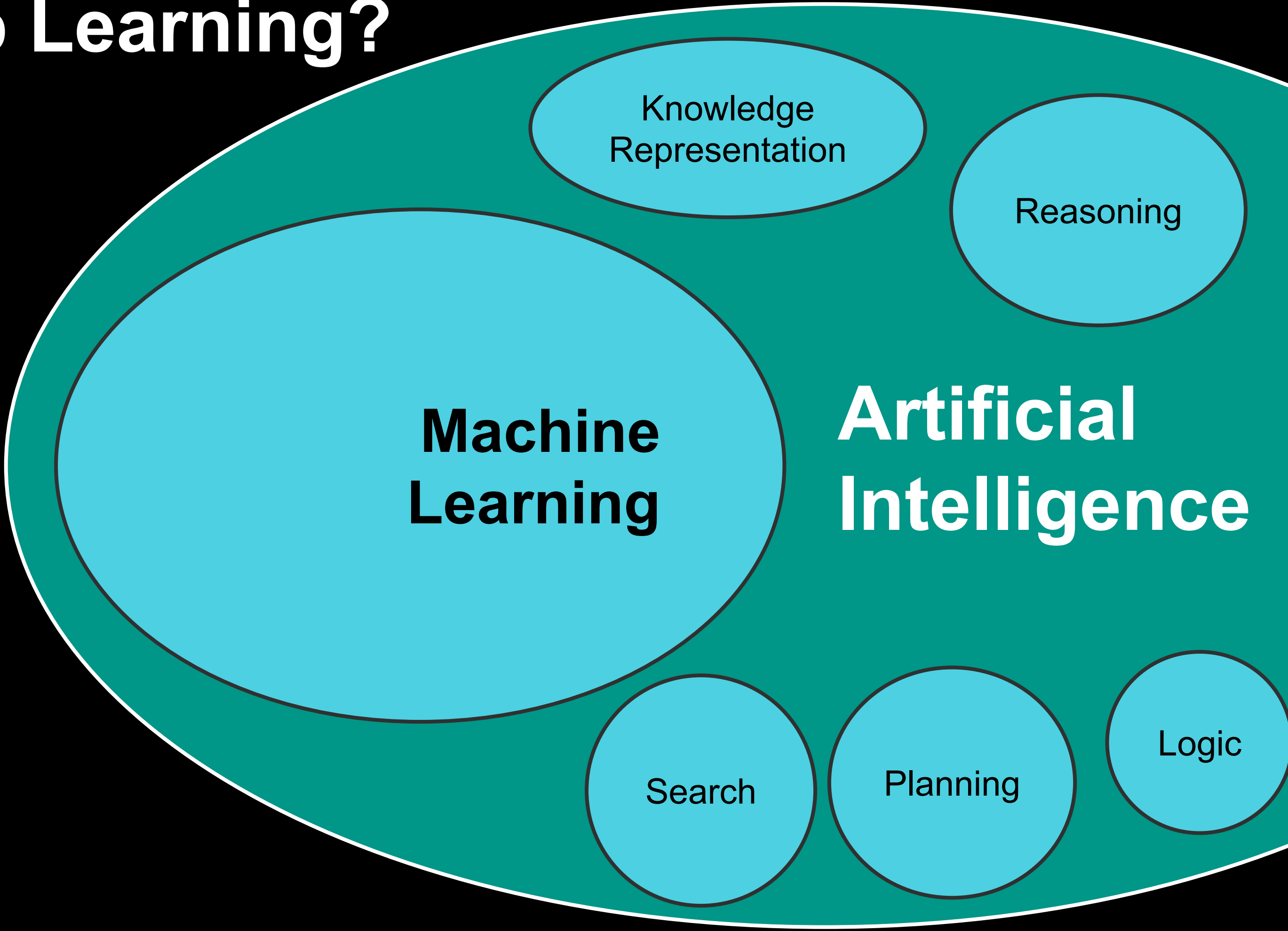
Artificial Intelligence

- Intelligence demonstrated by machines.
  - The study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.
  - Machines that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving".

What is Deep Learning?

Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.

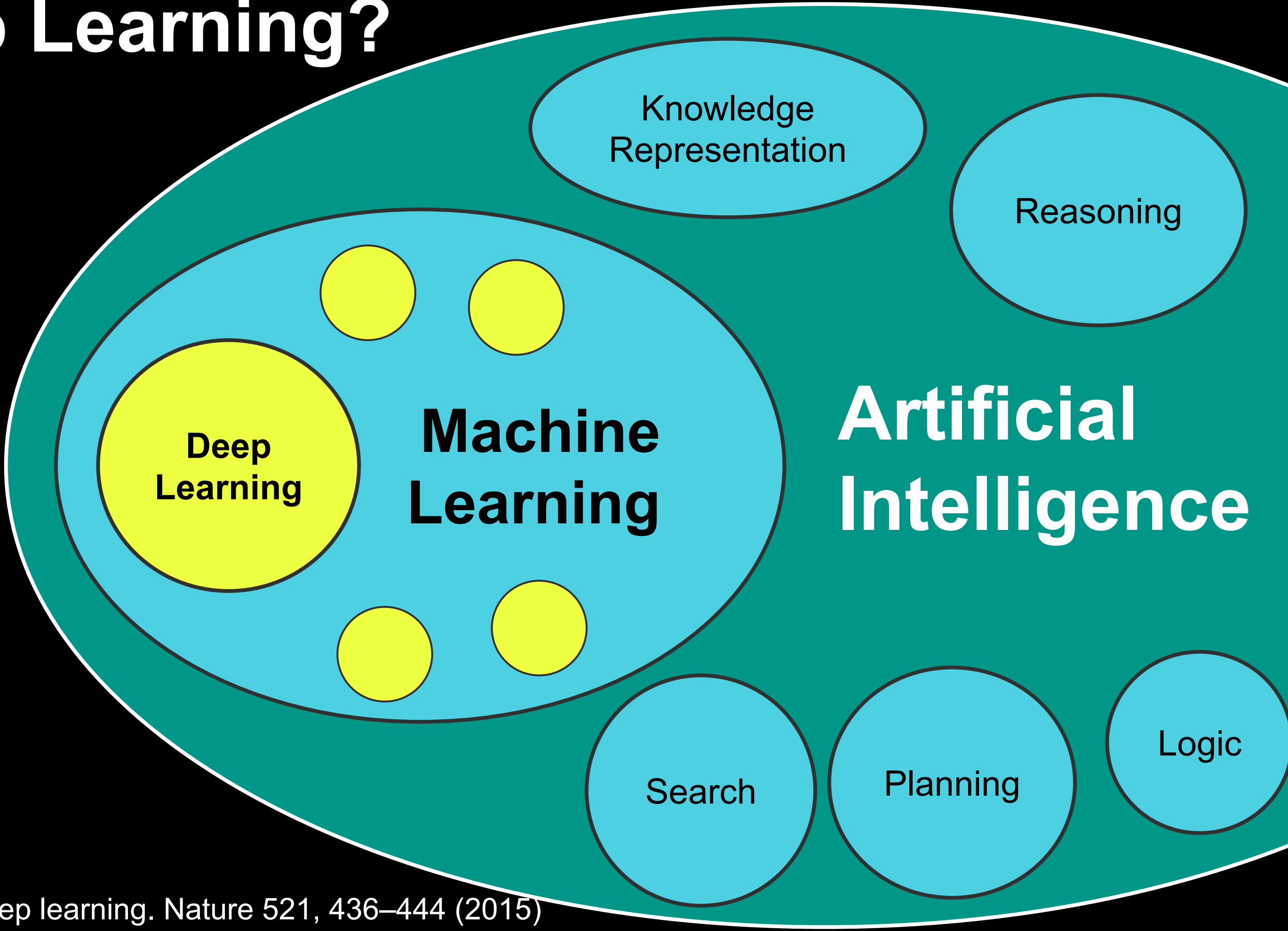
Machine Learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task



What is Deep Learning?

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.



LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015)

What is Robotic Vision?

What is Robotic Vision?

Output

Input

Input \ Output	Images	Data
Images	?	?
Data	?	?

What is Robotic Vision?

Input \ Output	Images	Data
Images	Image Processing	?
Data	?	?

What is Robotic Vision?

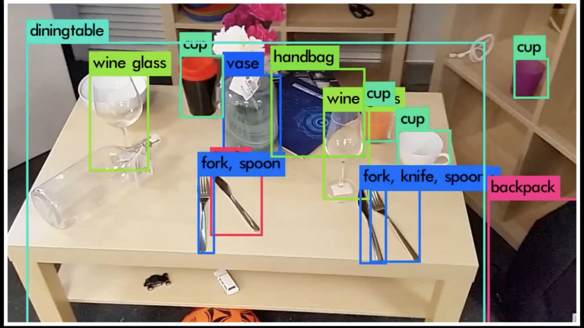
		Output	
		Images	Data
Input	Images	Image Processing	?
	Data	Computer Graphics	?

What is Robotic Vision?

Output

Input

	Images	Data
Images	Image Processing	Computer Vision
Data	Computer Graphics	?



What is Robotic Vision?

Output

Input		Images	Data
	Images	Image Processing	Computer Vision
	Data	Computer Graphics	Data Science

What is Robotic Vision?

Output

Input	Images	Data
Images	Image Processing	Computer Vision
Data	Computer Graphics	Data Science

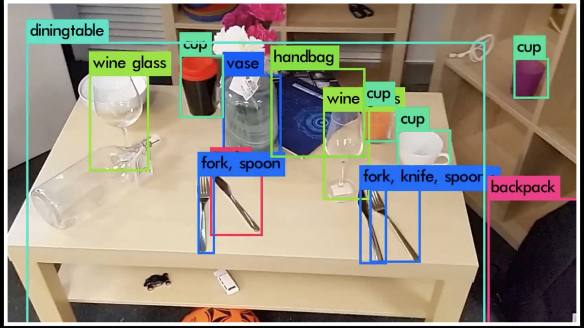
“Computer Vision on a robot?”

What is Robotic Vision?

Output

Input

	Images	Data
Images	Image Processing	Computer Vision
Data	Computer Graphics	Data Science



"Computer Vision on a robot?"

What is Robotic Vision?

Input \ Output	Images	Data	Actions
Images	Image Processing	Computer Vision	Robotic Vision
Data	Computer Graphics	Data Science	

What is Robotic Vision?

This is where robotic vision differs from computer vision. ***For robotic vision, perception is only one part of a more complex, embodied, active, and goal-driven system.***

Robotic vision therefore has to take into account that its immediate outputs (object detection, segmentation, depth estimates, 3D reconstruction, a description of the scene, and so on), will ultimately result in actions in the real world.

In a simplified view, whereas computer vision takes images and translates them into information, robotic vision translates images into actions.

Article

The International Journal of Robotics Research
2018, Vol. 37(4–5) 405–420

DOI: 10.1177/0278364918770733
journals.sagepub.com/home/ijr
SAGE

The limits and potentials of deep learning for robotics

Niko Sünderhauf[1], Oliver Brock[2], Walter Scheirer[3], Raia Hadsell[4], Dieter Fox[5], Jürgen Leitner[1], Ben Upcroft[6], Pieter Abbeel[7], Wolfram Burgard[8], Michael Milford[1] and Peter Corke[1]

Abstract

The application of deep learning in robotics leads to very specific problems and research questions that are typically not addressed by the computer vision and machine learning communities. In this paper we discuss a number of robotics-specific learning, reasoning, and embodiment challenges for deep learning. We explain the need for better evaluation metrics, highlight the importance and unique challenges for deep robotic learning in simulation, and explore the spectrum between purely data-driven and model-driven approaches. We hope this paper provides a motivating overview of important research directions to overcome the current limitations, and helps to fulfill the promising potentials of deep learning in robotics.

Keywords

Robotics, deep learning, machine learning, robotic vision

1. Introduction

A robot is an inherently *active* agent that interacts with the real world, and often operates in uncontrolled or detrimental conditions. Robots have to perceive, decide, plan, and execute actions, all based on incomplete and uncertain knowledge. Mistakes can lead to potentially catastrophic results that will not only endanger the success of the robot's mission, but can even put human lives at risk, e.g. if the robot is a driverless car.

The application of deep learning in robotics therefore motivates research questions that differ from those typically addressed in computer vision: How much trust can we put in the predictions of a deep learning system when misclassifications can have catastrophic consequences? How can we estimate the uncertainty in a deep network's predictions and how can we fuse these predictions with prior knowledge and other sensors in a probabilistic framework? How well does deep learning perform in realistic unconstrained open-set scenarios where objects of unknown class and appearance are regularly encountered?

If we want to use data-driven learning approaches to generate motor commands for robots to move and act in the world, we are faced with additional challenging questions: How can we generate enough high-quality training data? Do we rely on data solely collected on robots in real-world scenarios or do we require data augmentation through simulation? How can we ensure the learned policies transfer well to different situations, from simulation to reality, or between different robots?

This leads to further fundamental questions: How can the structure, the constraints, and the physical laws that govern robotic tasks in the real world be leveraged and exploited by a deep learning system? Is there a fundamental difference between model-driven and data-driven problem solving, or are these rather two ends of a spectrum?

This paper explores some of the challenges, limits, and potentials for deep learning in robotics. The invited speakers and organizers of the workshop on *The Limits and*

[1]Australian Centre for Robotic Vision, Queensland University of Technology (QUT), Brisbane, Australia
[2]Robotics and Biology Laboratory, Technische Universität Berlin, Germany
[3]Department of Computer Science and Engineering, University of Notre Dame, IN, USA
[4]DeepMind, London, UK
[5]Paul G. Allen School of Computer Science & Engineering, University of Washington, WA, USA
[6]Oxbotica Ltd., Oxford, UK
[7]UC Berkeley, Department of Electrical Engineering and Computer Sciences, CA, USA
[8]Department of Computer Science, University of Freiburg, Germany

Corresponding author:
Niko Sünderhauf, Queensland University of Technology (QUT), 2 George Street, Brisbane 4000 QLD, Australia.
Email: niko.suenderhauf@roboticvision.org

The Limits and Potentials of Deep Learning for Robotics. Sünderhauf, Brock, Scheirer, Hadsell, Fox, Leitner, Upcroft, Abbeel, Burgard, Milford, Corke. IJRR 2018.

Supervised (Deep) Learning

Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

It infers a function from labeled training data consisting of a set of training examples.

Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

It infers a function from labeled training data consisting of a set of training examples.

Training examples: (image, label)

X={ ( , ‘dog’),

( , ‘cat’),

( , ‘car’), … }

Supervised Learning

Training examples: (image, label)

X={ ( , ‘dog’),

( , ‘cat’),

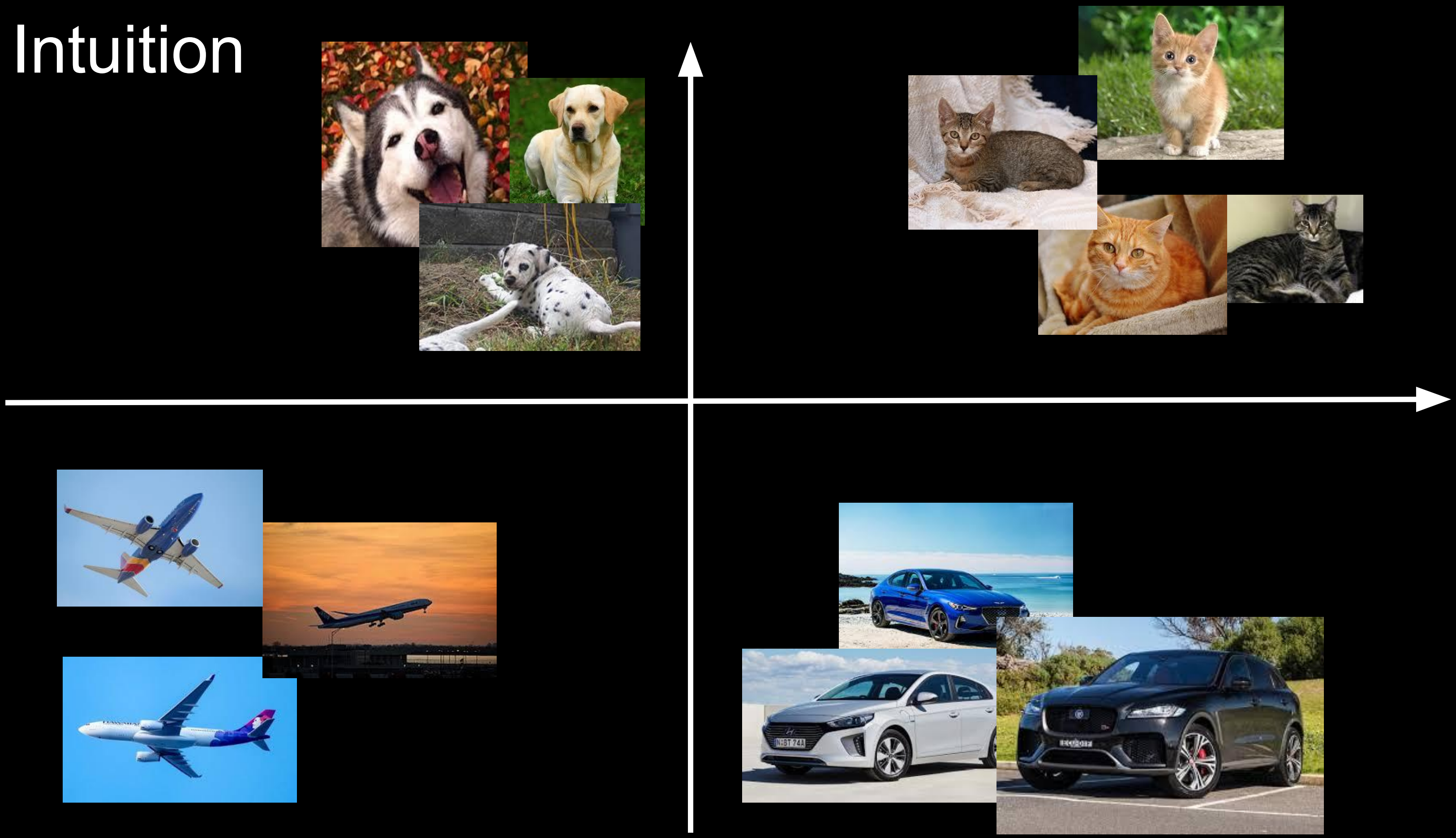
( , ‘car’), … }

Goal: Learn function f: Image → Label

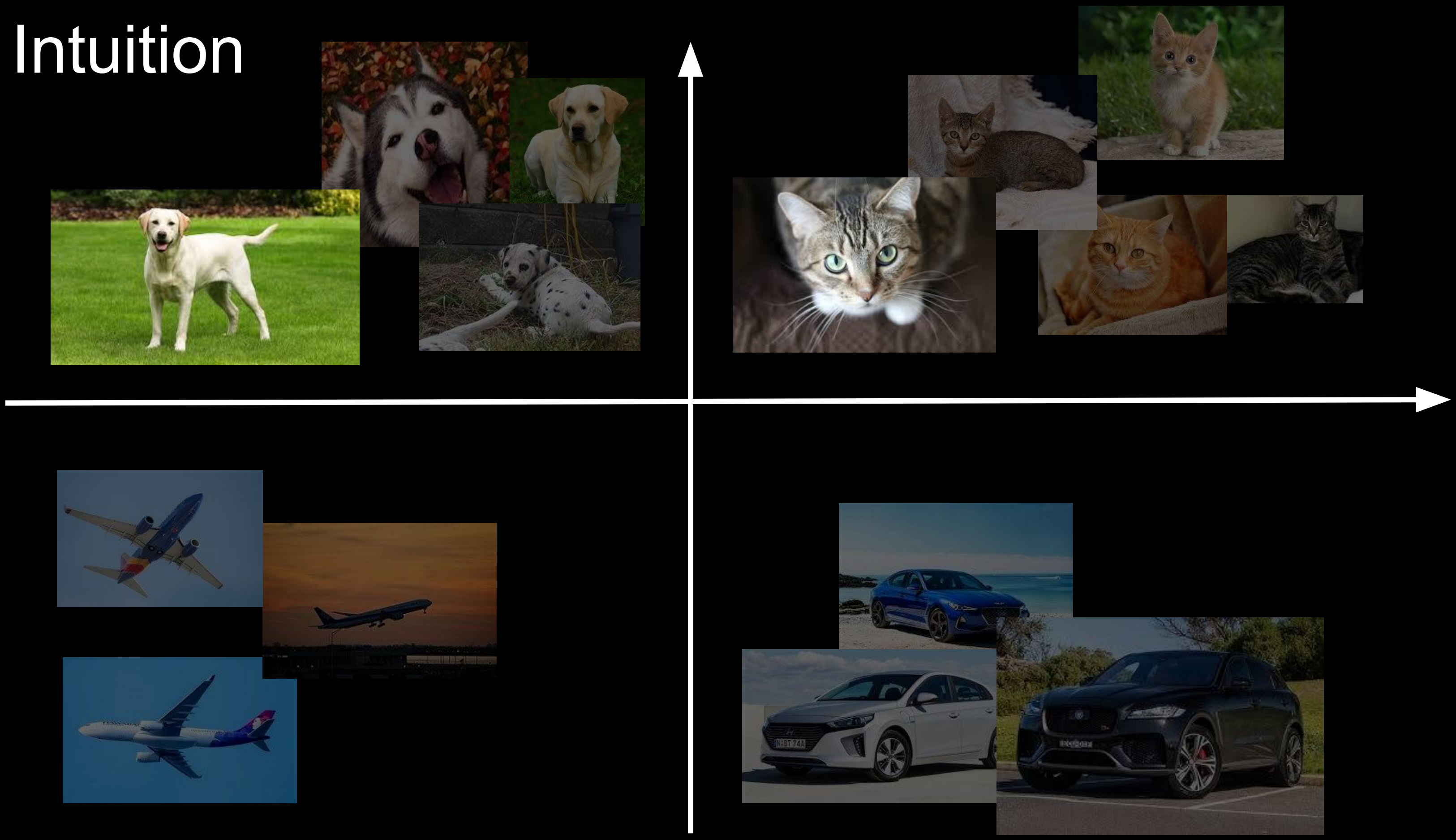
f() = ‘cat’ (if all goes well)

Nearest Neighbor Classifiers

Intuition



Intuition



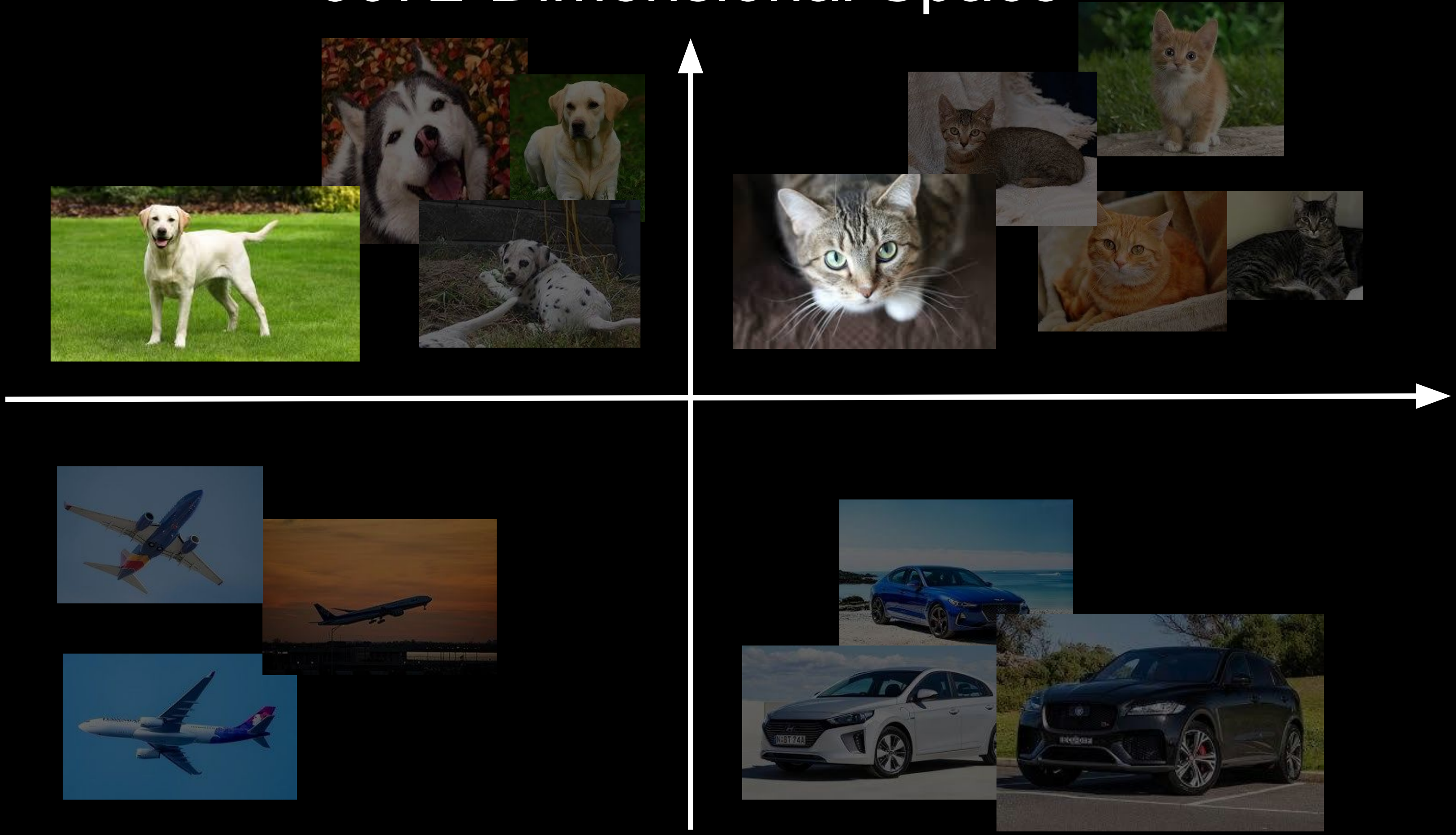
Every Image can be rearranged into a **vector**.

Shape: (32,32,3)

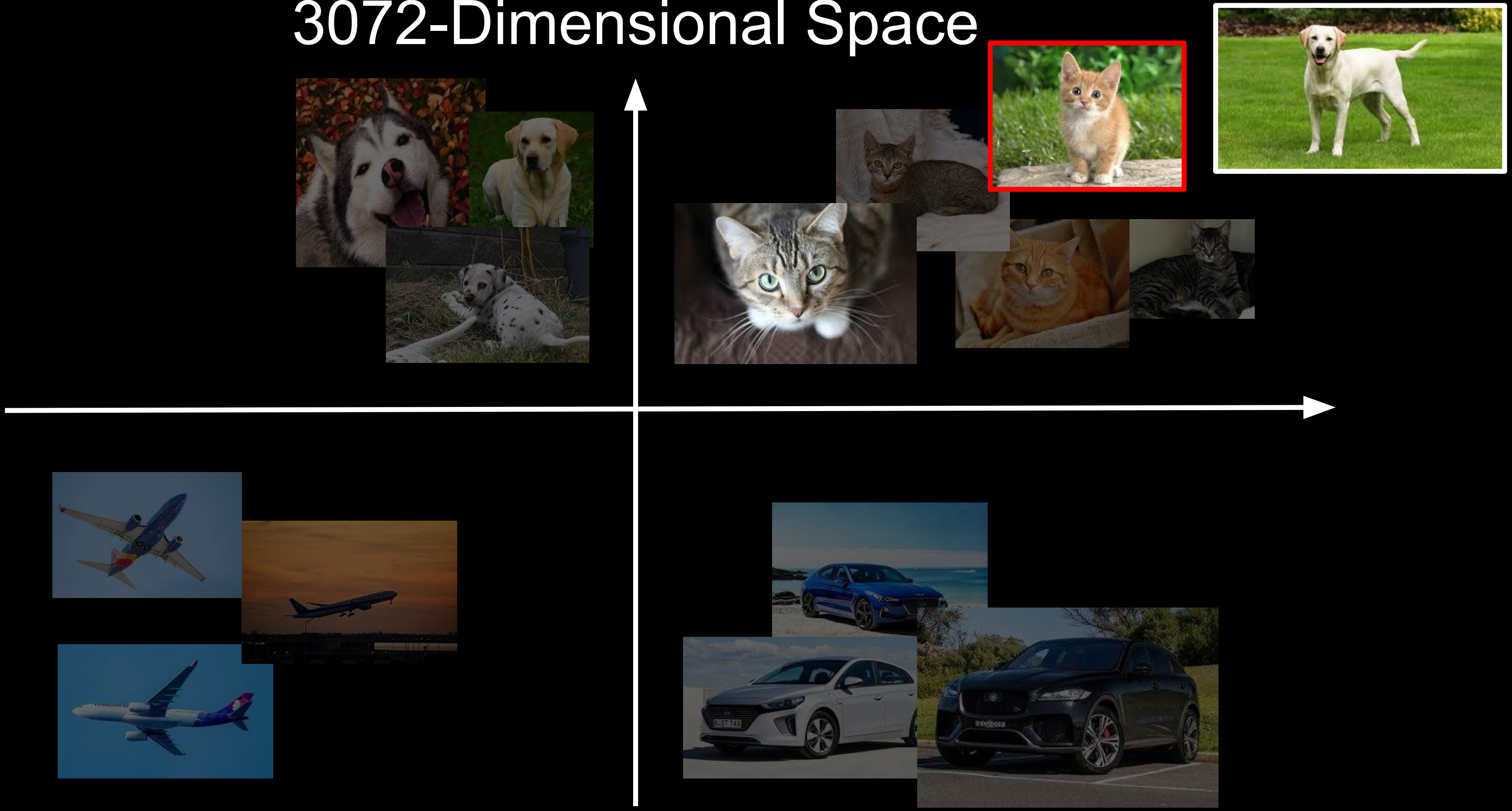
Shape: (1024,1,3)

Shape: (3072,1)

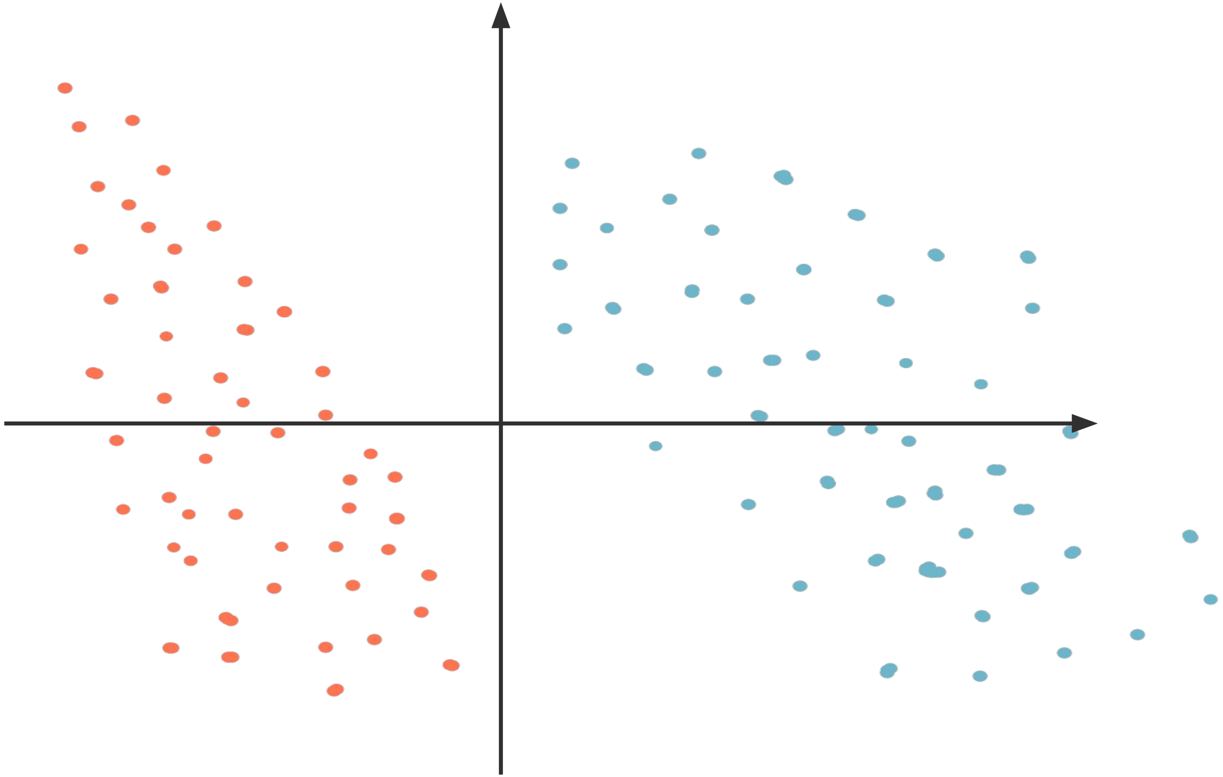
3072-Dimensional Space

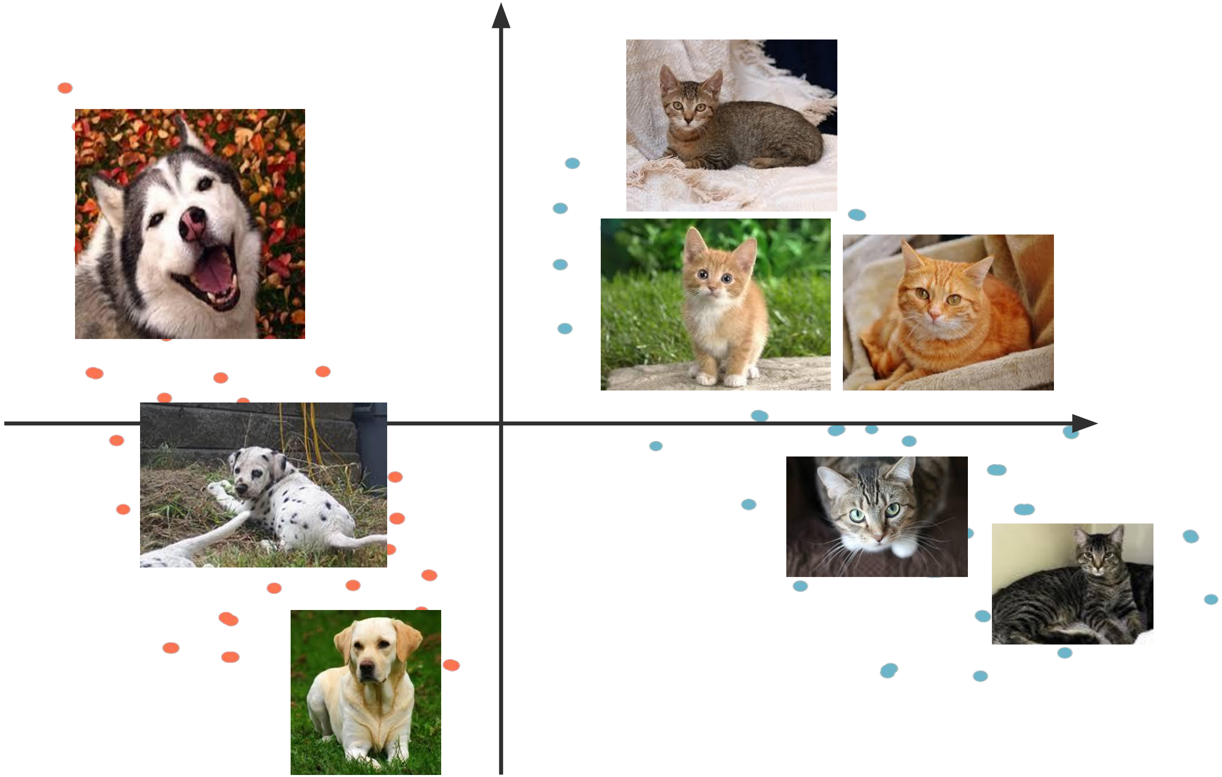


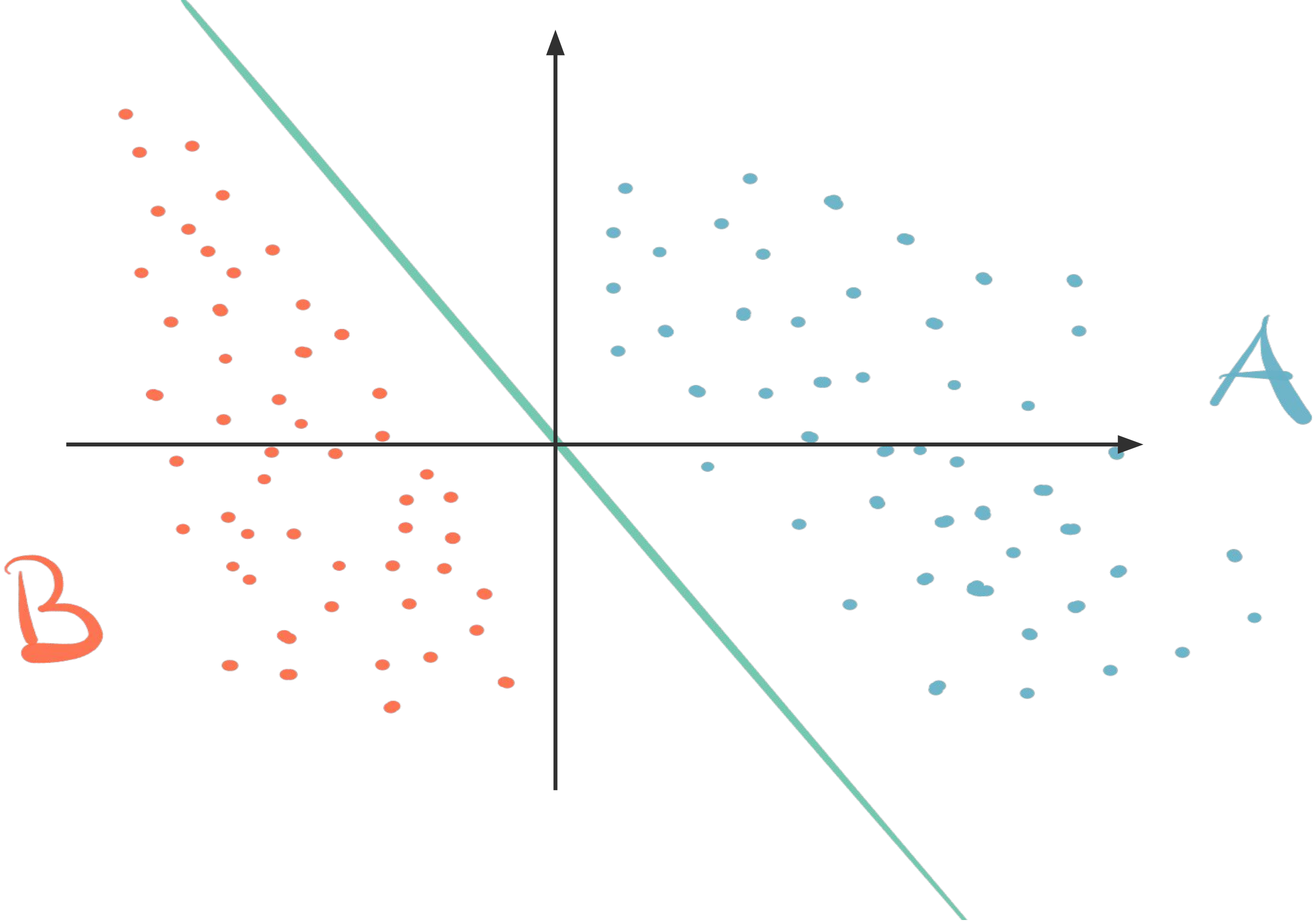
3072-Dimensional Space

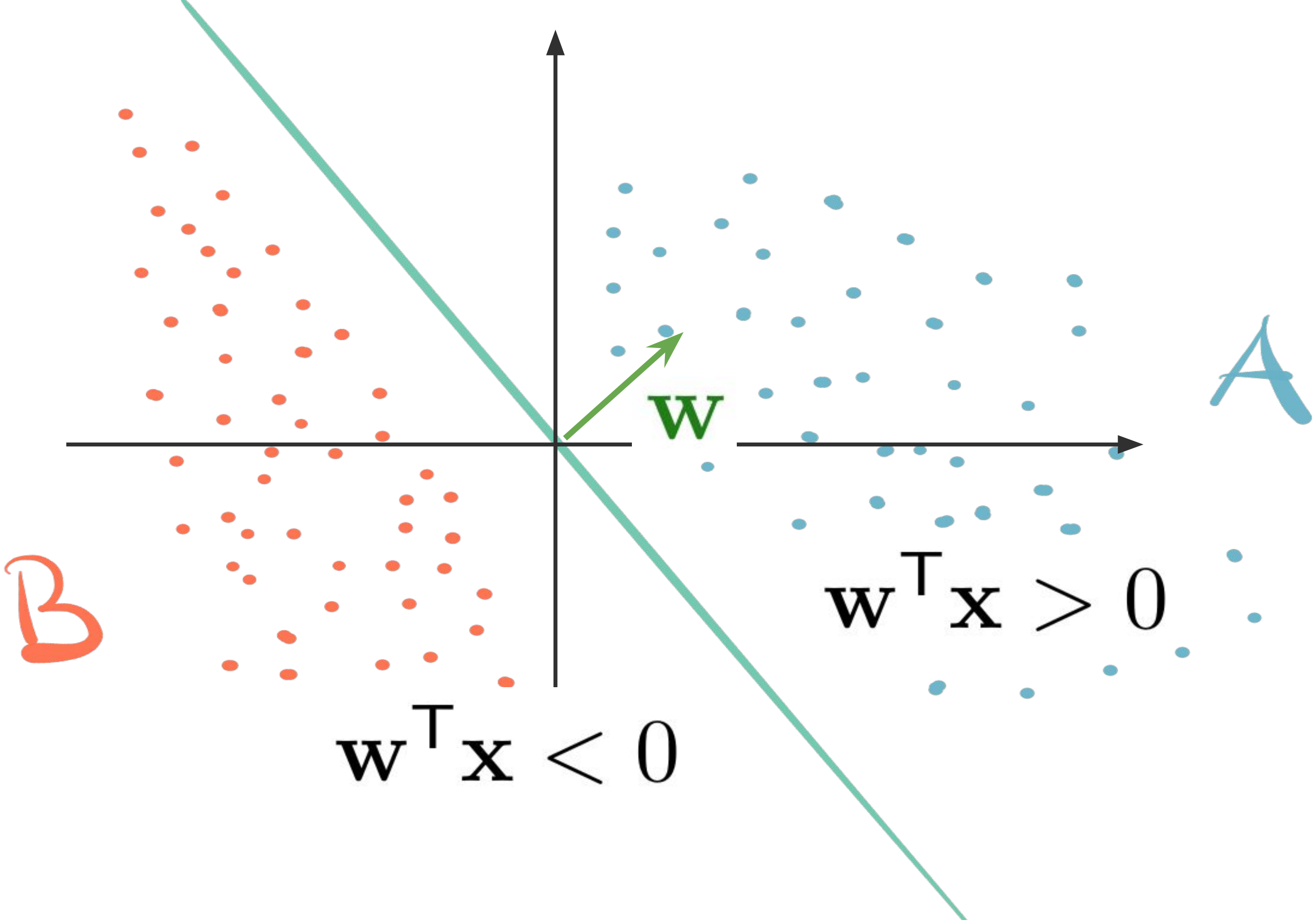


Linear Classifiers

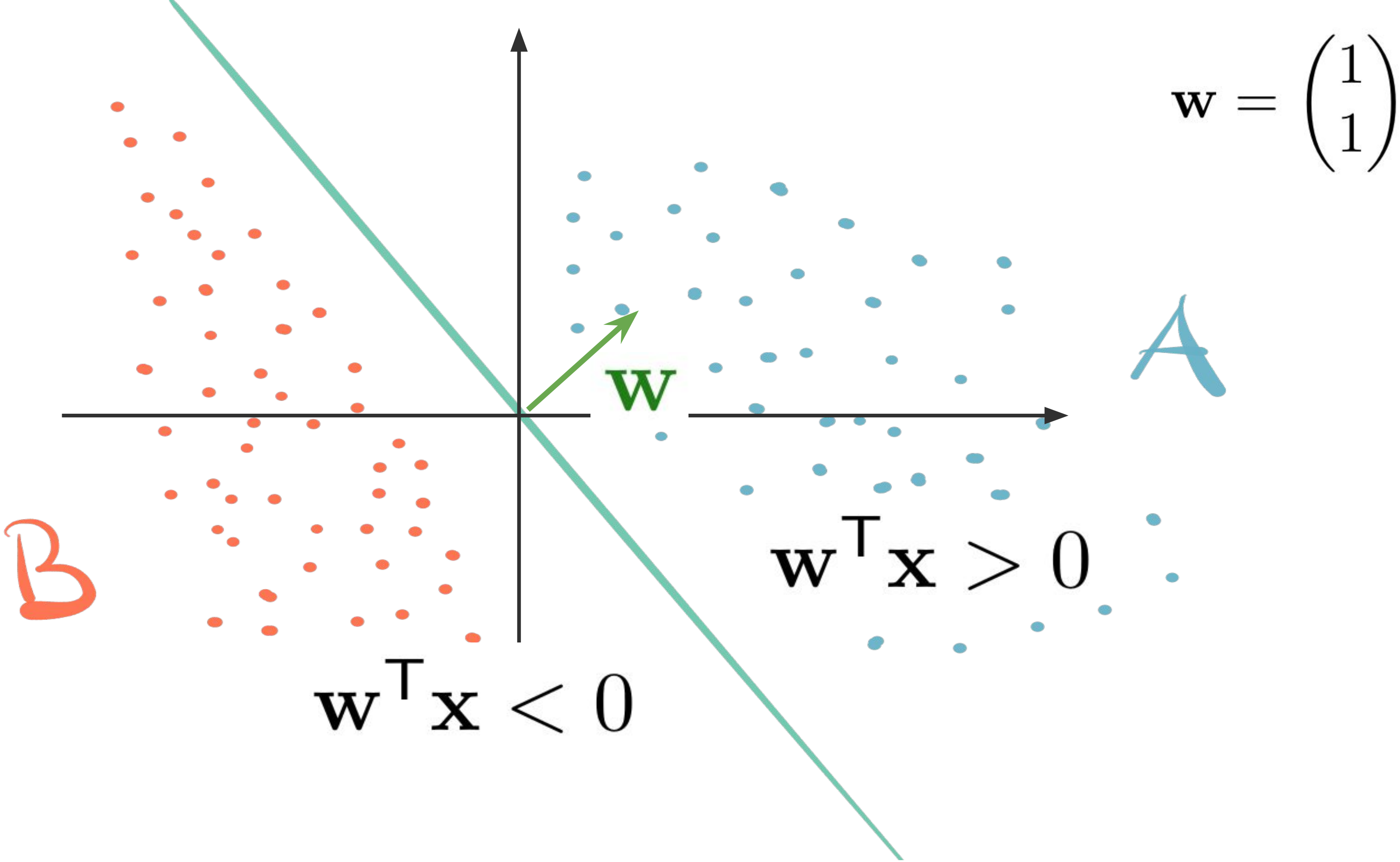





A
B

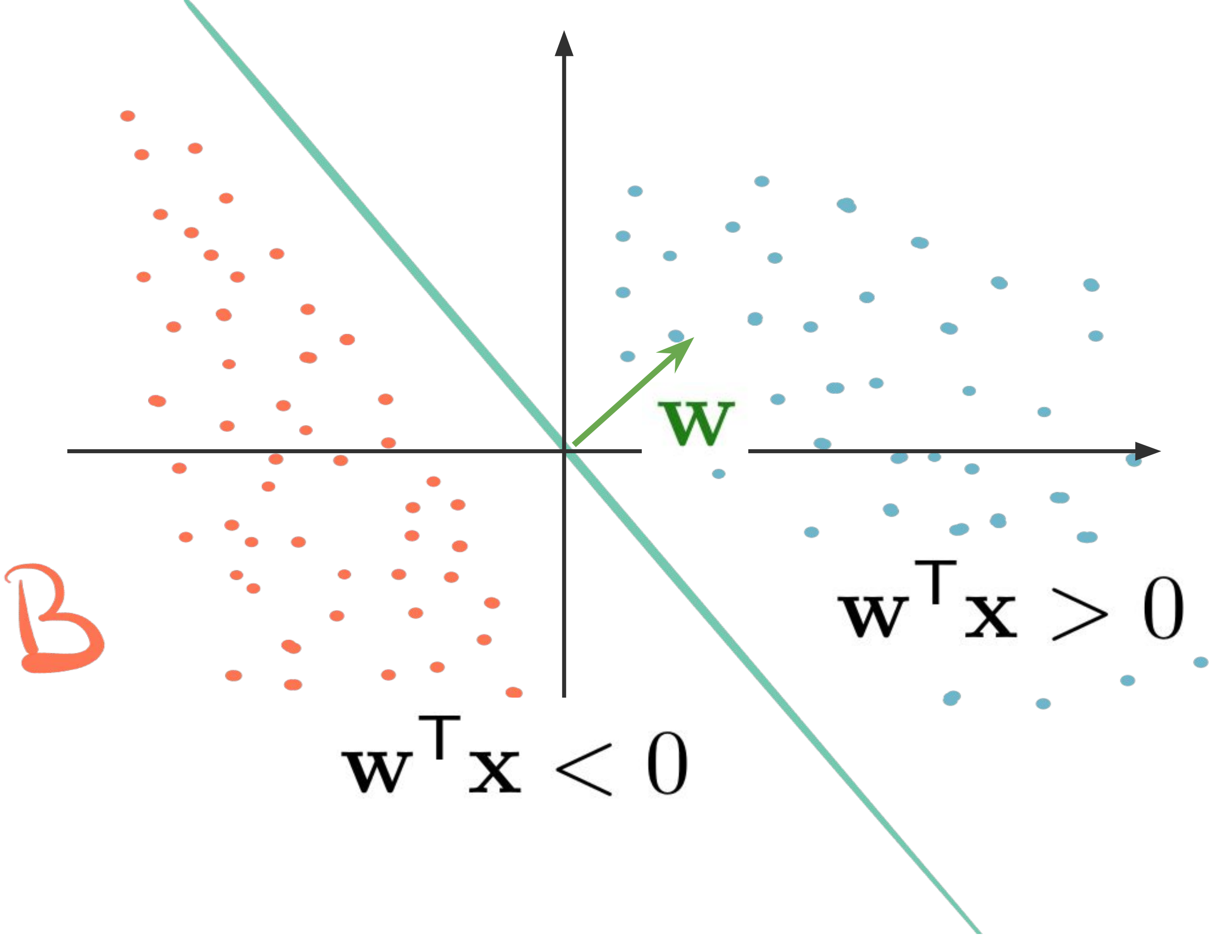


$\mathbf{w}$
$\mathcal{A}$
$\mathcal{B}$
$\mathbf{w}^\top \mathbf{x} > 0$
$\mathbf{w}^\top \mathbf{x} < 0$



$\mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
A
B
w
$\mathbf{w}^\mathsf{T}\mathbf{x} > 0$
$\mathbf{w}^\mathsf{T}\mathbf{x} < 0$

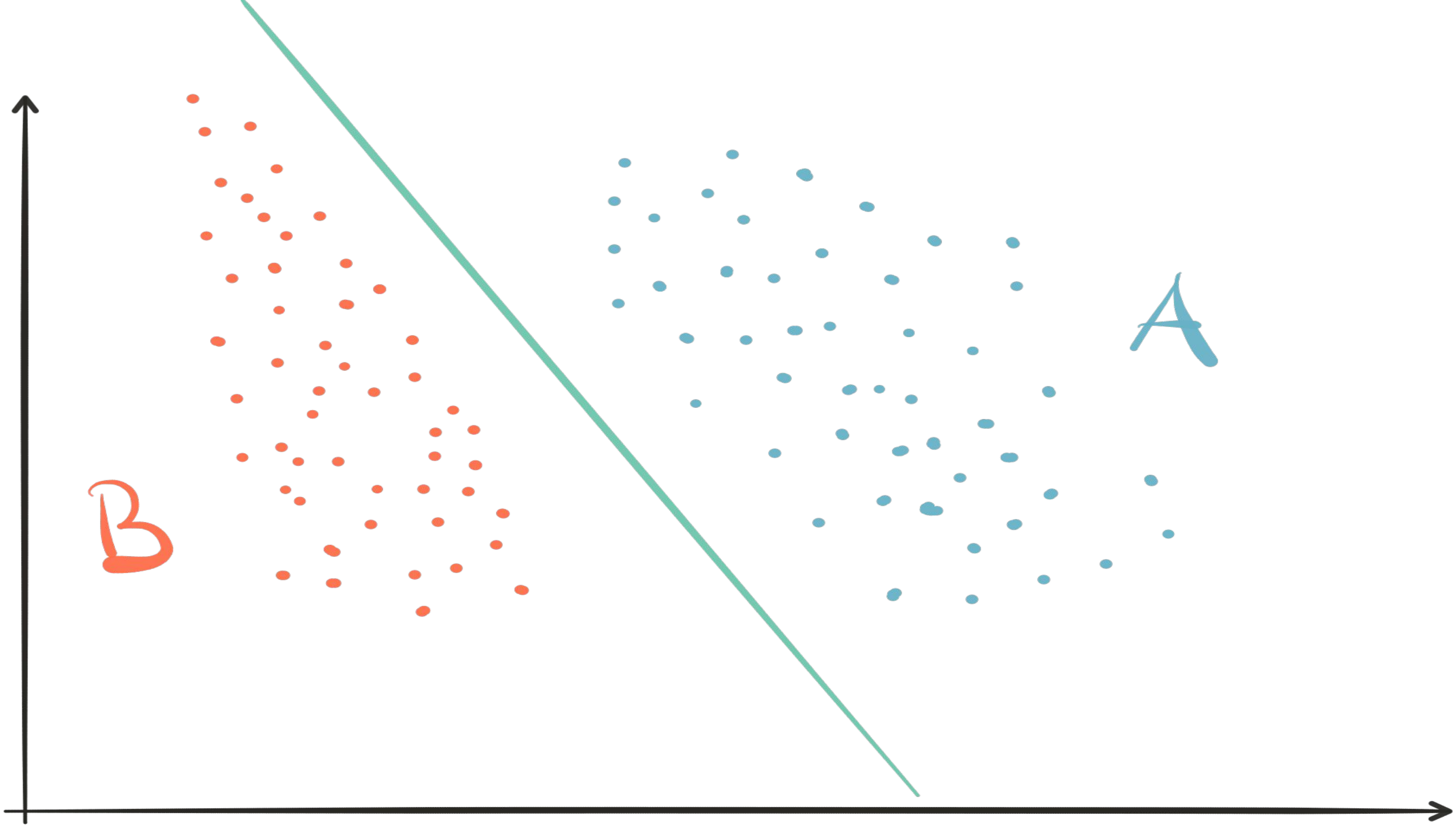
$$\mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

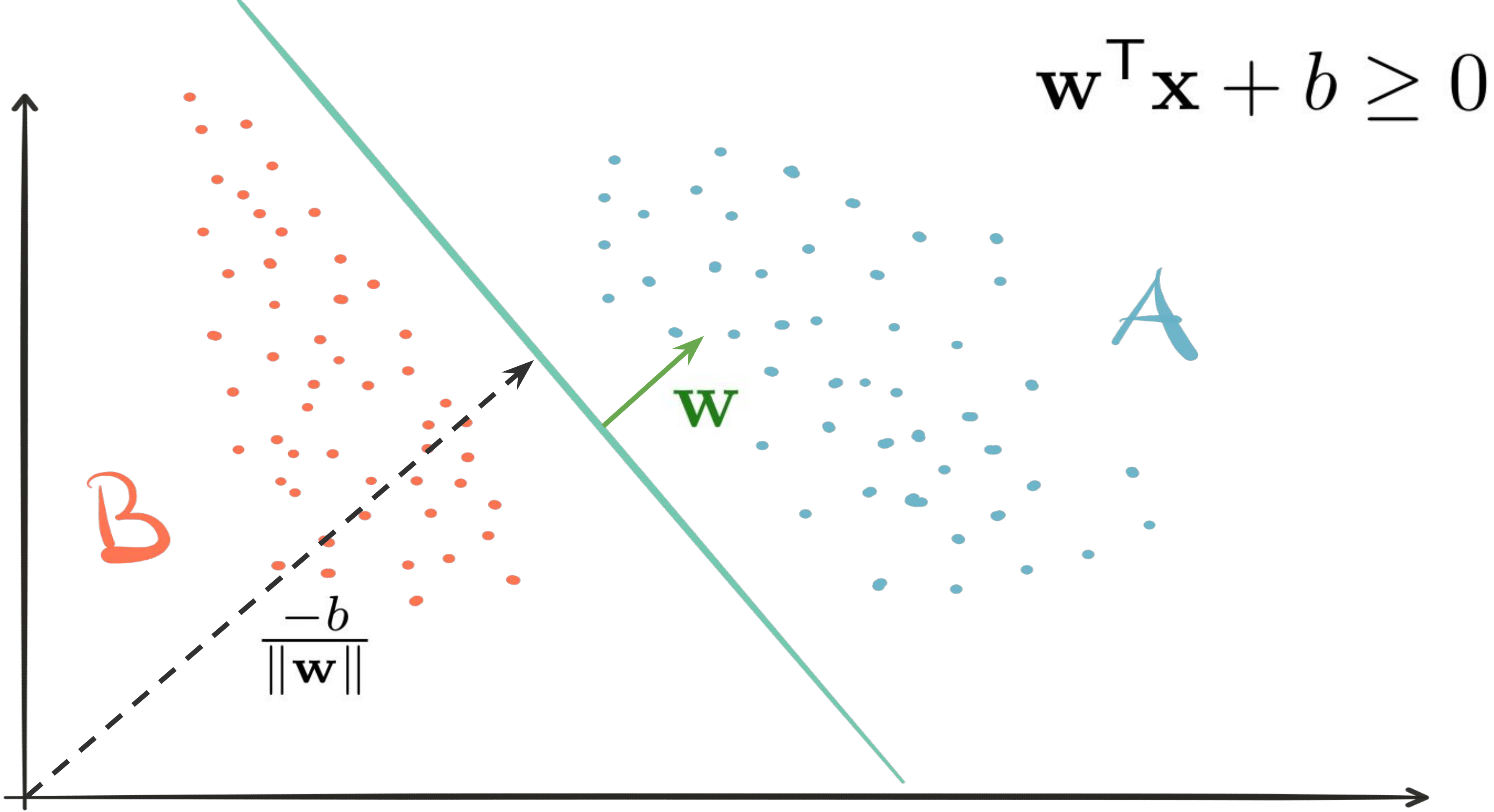


$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^\mathsf{T} \cdot \begin{pmatrix} 5 \\ 5 \end{pmatrix} = 10$$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^\mathsf{T} \cdot \begin{pmatrix} -5 \\ 5 \end{pmatrix} = 0$$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^\mathsf{T} \cdot \begin{pmatrix} -10 \\ 5 \end{pmatrix} = -5$$

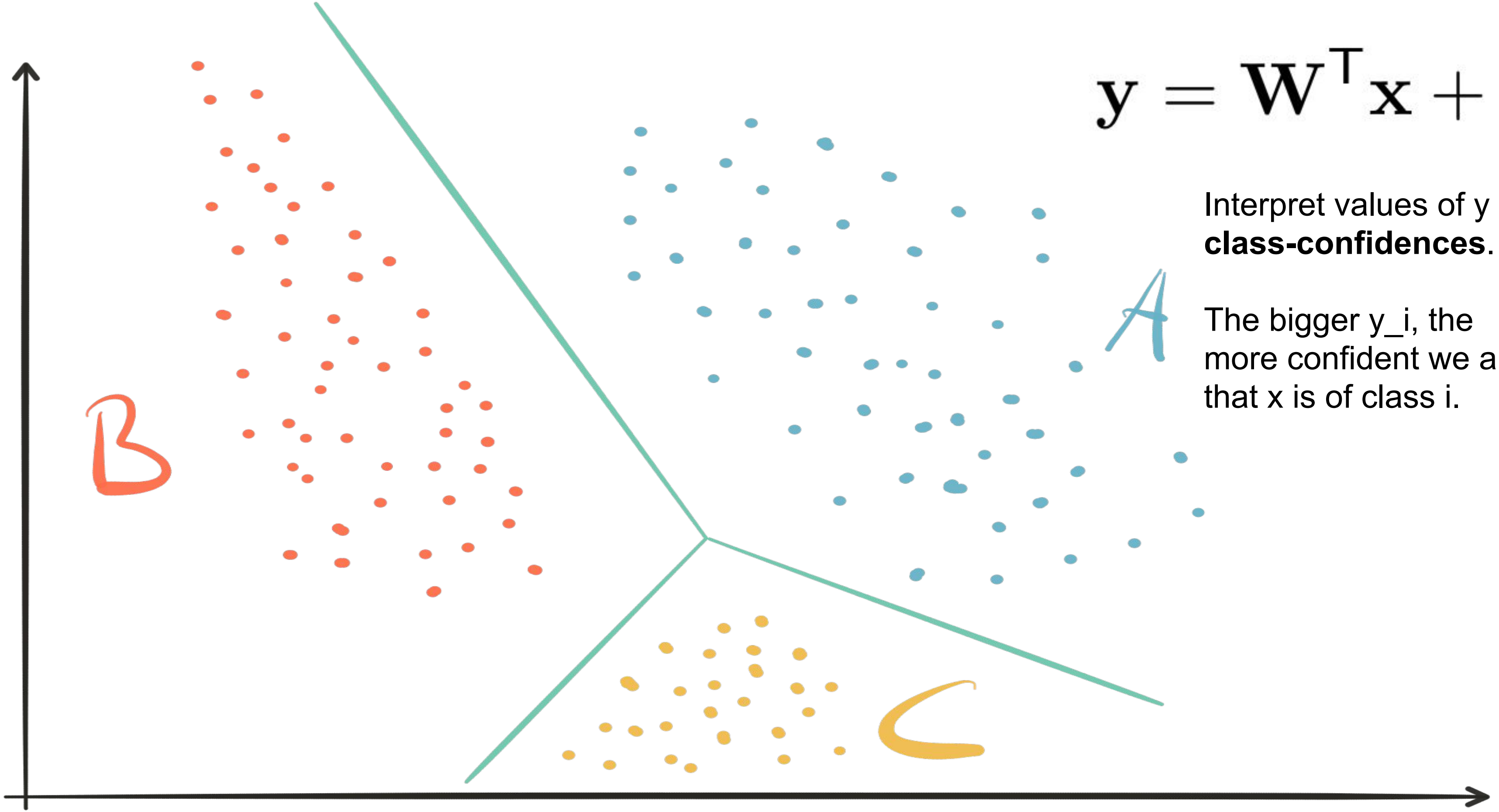

B
A


$\mathbf{w}^\mathsf{T}\mathbf{x} + b \geq 0$
$\mathcal{A}$
$\mathbf{w}$
$\mathcal{B}$
$\frac{-b}{||\mathbf{w}||}$

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

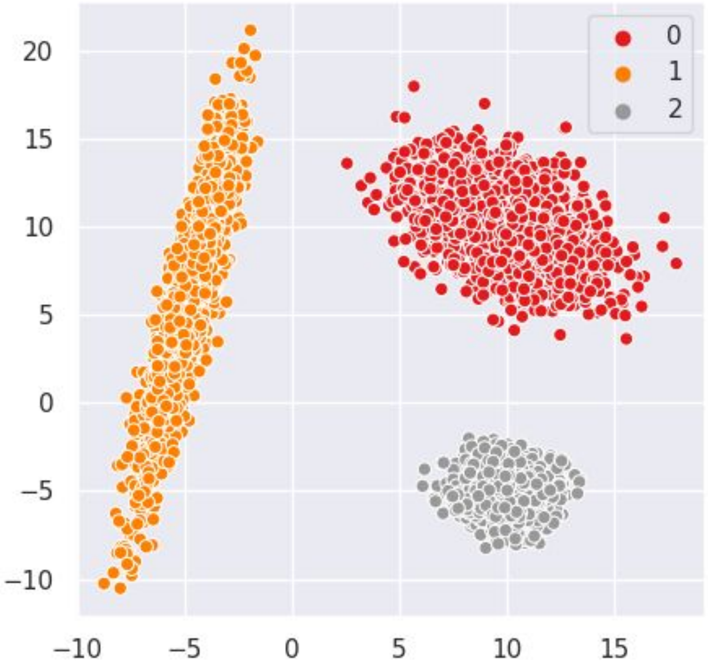
Interpret values of y as **class-confidences**.

The bigger y_i, the more confident we are that x is of class i.



$$\mathbf{W}^{\mathsf{T}} = \begin{pmatrix} 0.15 & 0.36 \\ -1.63 & 0.28 \\ 0.25 & -1.05 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} -0.84 \\ 1.57 \\ -0.02 \end{pmatrix}$$

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$



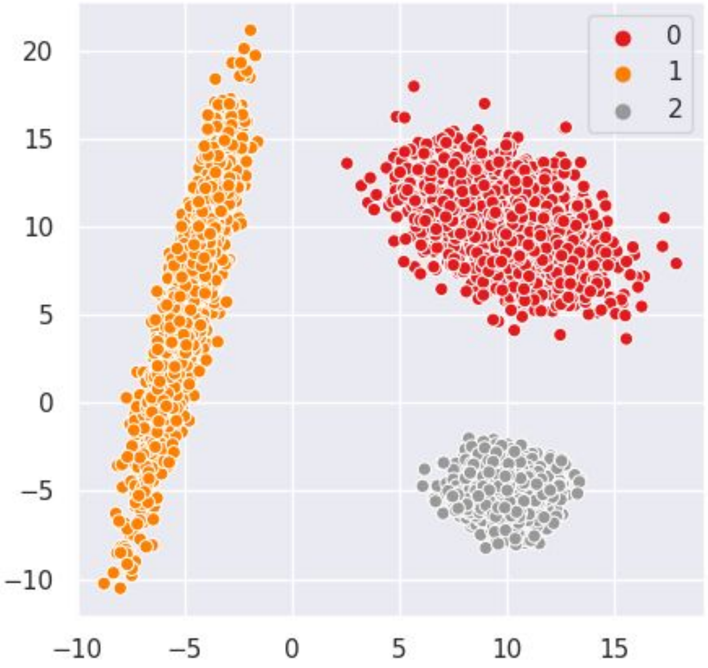
$$\mathbf{W}^\mathsf{T} = \begin{pmatrix} 0.15 & 0.36 \\ -1.63 & 0.28 \\ 0.25 & -1.05 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} -0.84 \\ 1.57 \\ -0.02 \end{pmatrix}$$

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ 10 \end{pmatrix}} + \mathbf{b} =$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ -5 \end{pmatrix}} + \mathbf{b} =$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} -10 \\ 0 \end{pmatrix}} + \mathbf{b} =$$



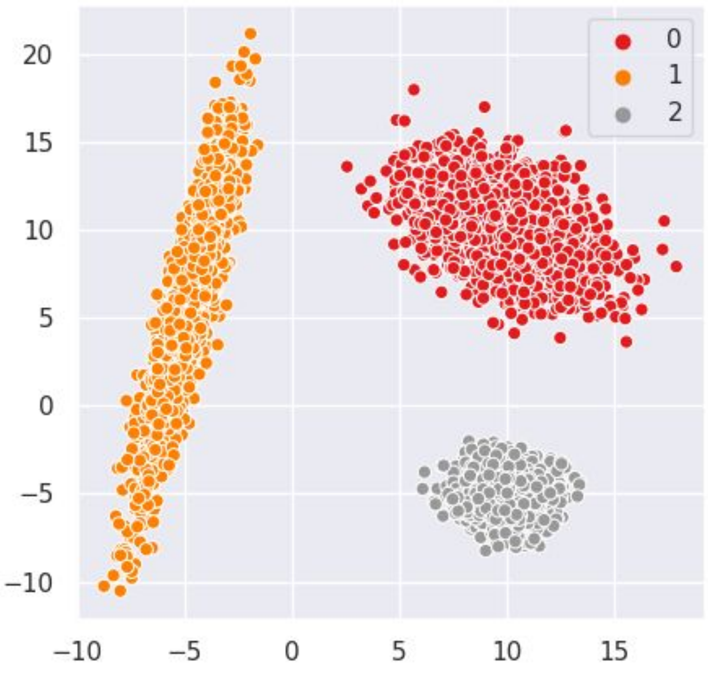
$$\mathbf{W}^\mathsf{T} = \begin{pmatrix} 0.15 & 0.36 \\ -1.63 & 0.28 \\ 0.25 & -1.05 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} -0.84 \\ 1.57 \\ -0.02 \end{pmatrix}$$

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ 10 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} 3.99 \\ -11.94 \\ -8.04 \end{pmatrix}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ -5 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} -1.03 \\ -16.12 \\ 7.75 \end{pmatrix}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} -10 \\ 0 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} -2.32 \\ 17.87 \\ -2.53 \end{pmatrix}$$



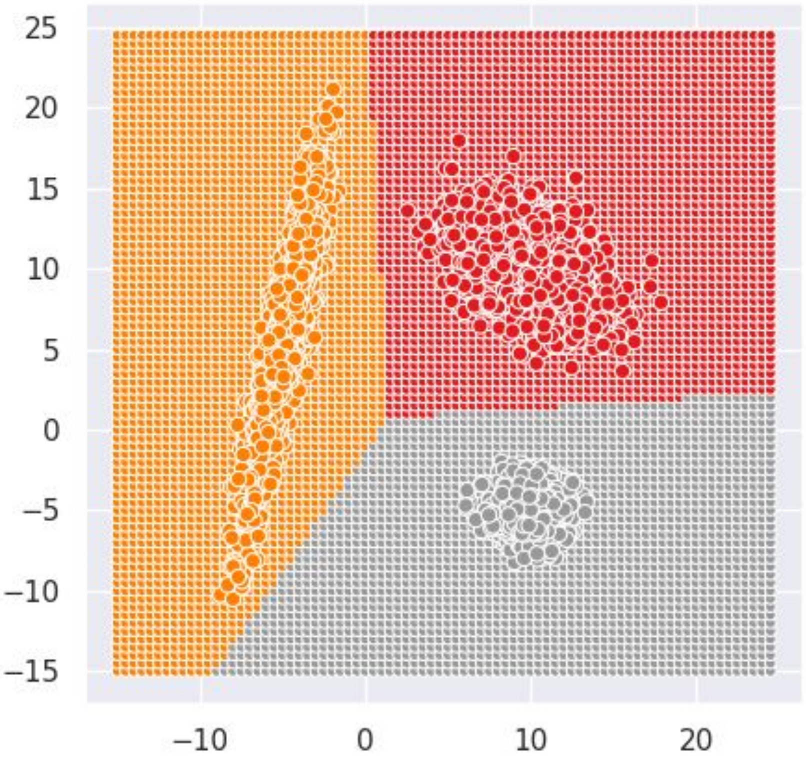
$$\mathbf{W}^\mathsf{T} = \begin{pmatrix} 0.15 & 0.36 \\ -1.63 & 0.28 \\ 0.25 & -1.05 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} -0.84 \\ 1.57 \\ -0.02 \end{pmatrix}$$

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ 10 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} 3.99 \\ -11.94 \\ -8.04 \end{pmatrix}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} 10 \\ -5 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} -1.03 \\ -16.12 \\ 7.75 \end{pmatrix}$$

$$\mathbf{W}^\mathsf{T} \cdot \boxed{\begin{pmatrix} -10 \\ 0 \end{pmatrix}} + \mathbf{b} = \begin{pmatrix} -2.32 \\ 17.87 \\ -2.53 \end{pmatrix}$$

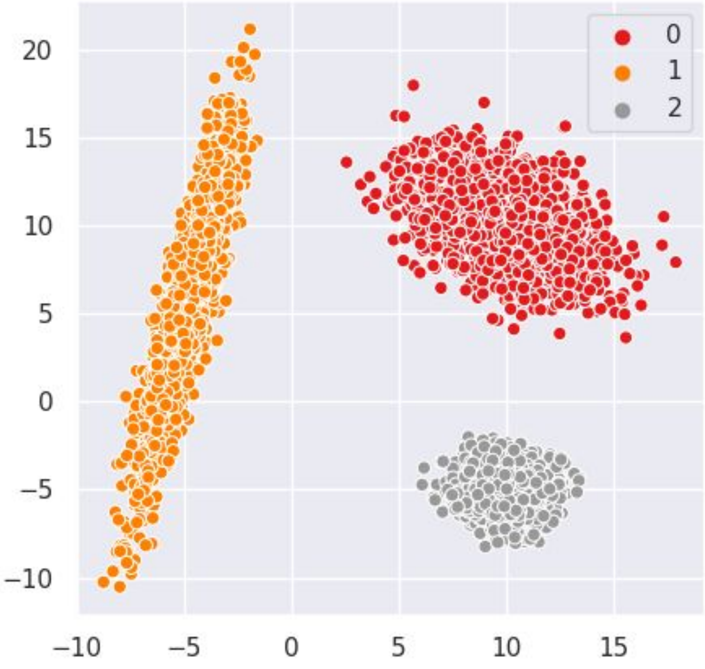


$$\mathbf{W}^\top \cdot \begin{pmatrix}10\\10\end{pmatrix} + \mathbf{b} = \begin{pmatrix}3.99\\-11.94\\-8.04\end{pmatrix}$$

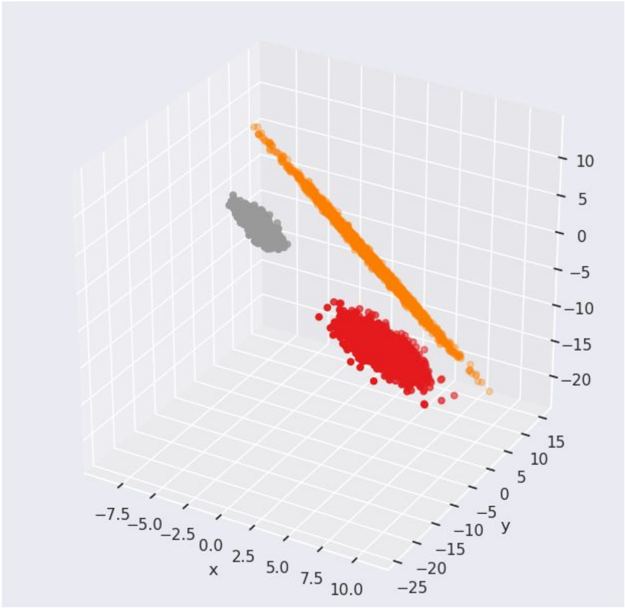
$$\mathbf{W}^\top \cdot \begin{pmatrix}10\\-5\end{pmatrix} + \mathbf{b} = \begin{pmatrix}-1.03\\-16.12\\7.75\end{pmatrix}$$

$$\mathbf{W}^\top \cdot \begin{pmatrix}-10\\0\end{pmatrix} + \mathbf{b} = \begin{pmatrix}-2.32\\17.87\\-2.53\end{pmatrix}$$

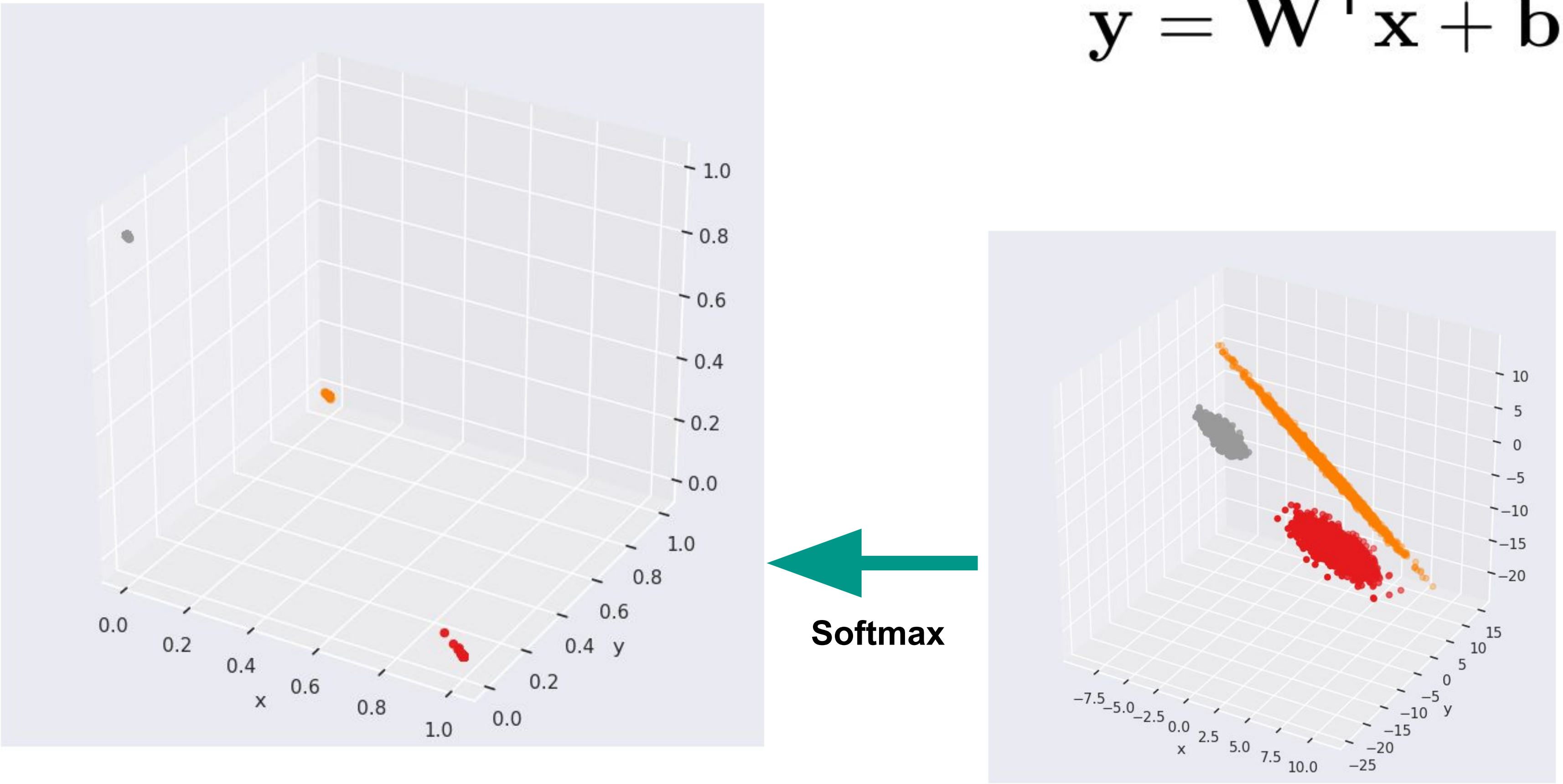
$$\mathbf{y} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$



We are actually projecting from 2D into 3D!



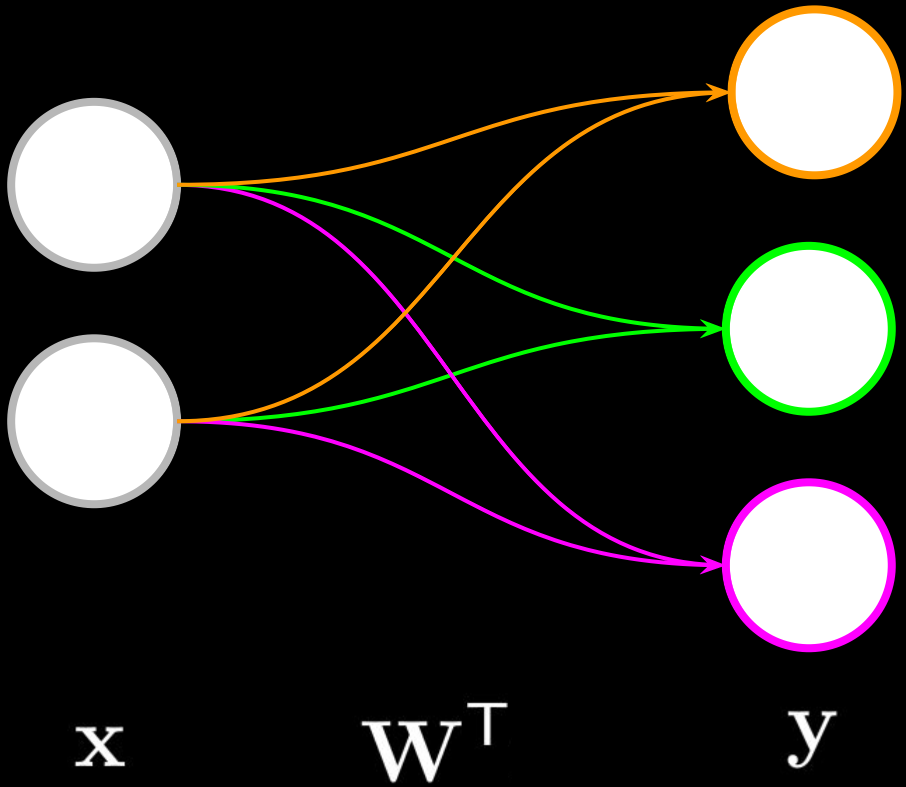
$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$



Towards a Neural Network

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

$$\mathbf{W} = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \end{pmatrix}$$

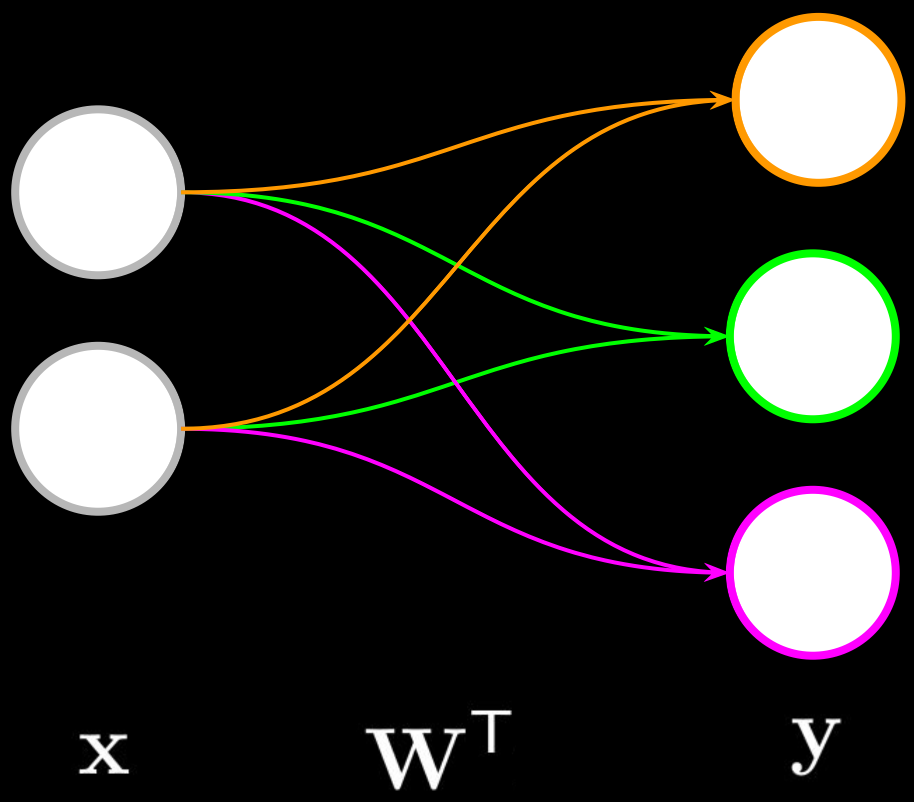


$$\mathbf{b} + \sum_i w_{i,0} \cdot x_i$$

$$\mathbf{b} + \sum_i w_{i,1} \cdot x_i$$

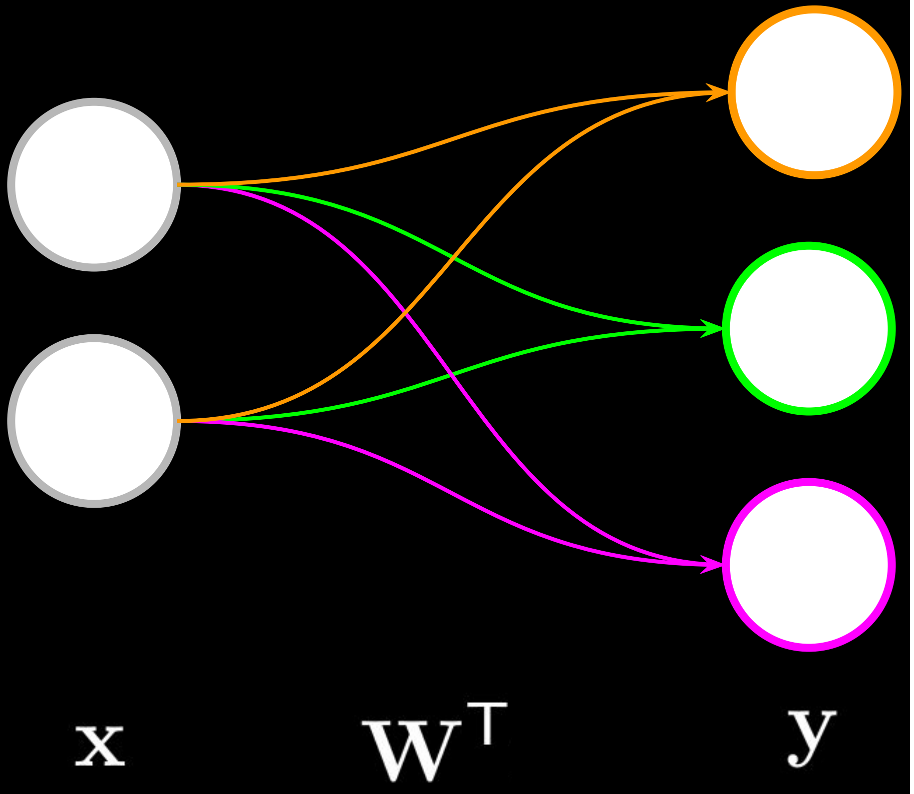
$$\mathbf{b} + \sum_i w_{i,2} \cdot x_i$$

$$y = W^T x + b$$




```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 3)

    def forward(self, x):
        x = self.fc1(x)
        return x
```

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$




```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 3)

    def forward(self, x):
        x = self.fc1(x)
        return x
```

```python
W = net.state_dict()['fc1.weight'].numpy()
b = net.state_dict()['fc1.bias'].numpy()
print('weights W:\n', W)
print('bias b:\n', b)
```

```
weights W:
 [[ 0.32842654  0.49274433]
 [-1.6420735   0.38735208]
 [ 0.42878065 -1.0042973 ]]
bias b:
 [-0.98728037  2.0173173   0.08274412]
```

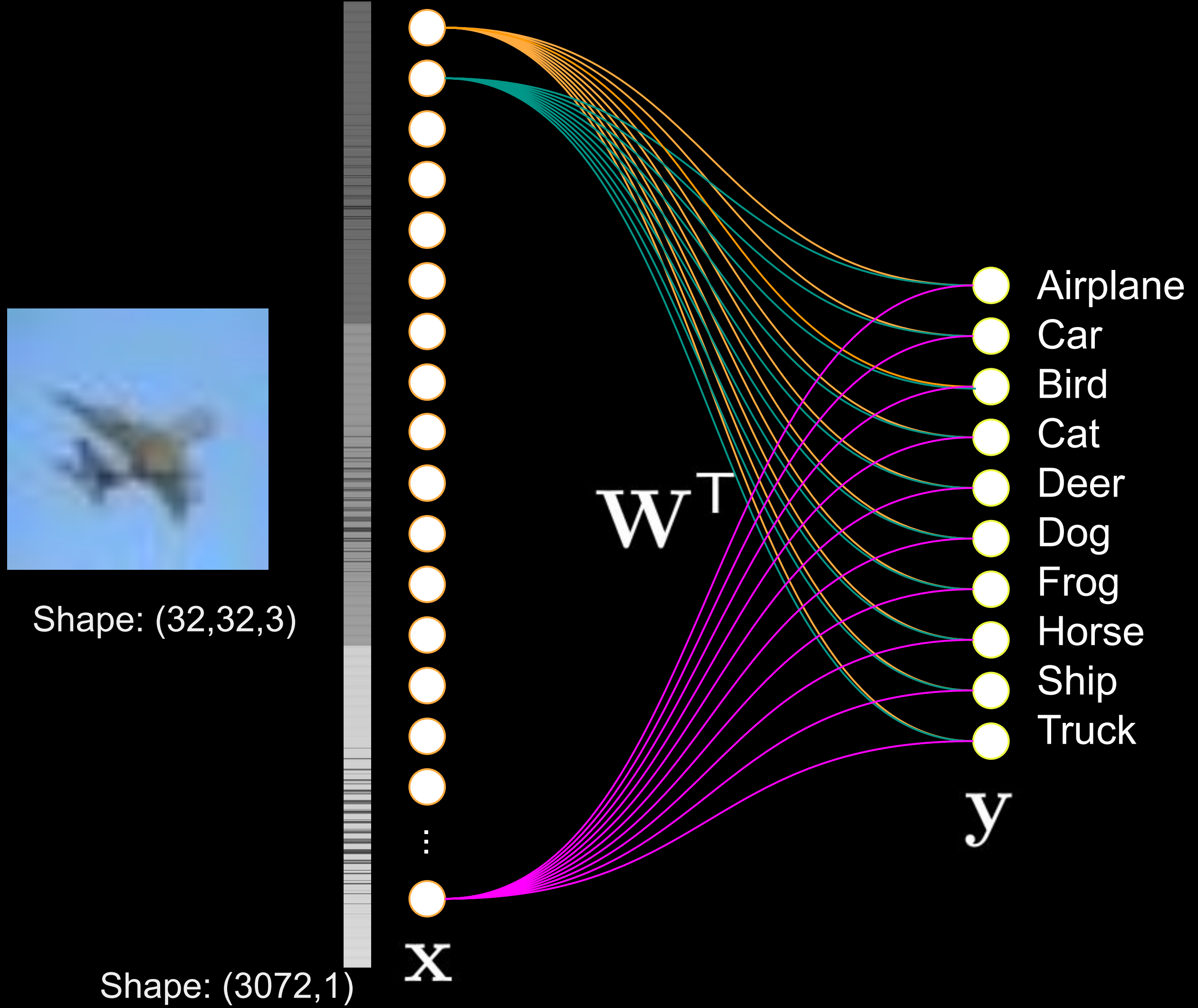
Every Image can be rearranged into a **vector**.

Shape: (32,32,3)

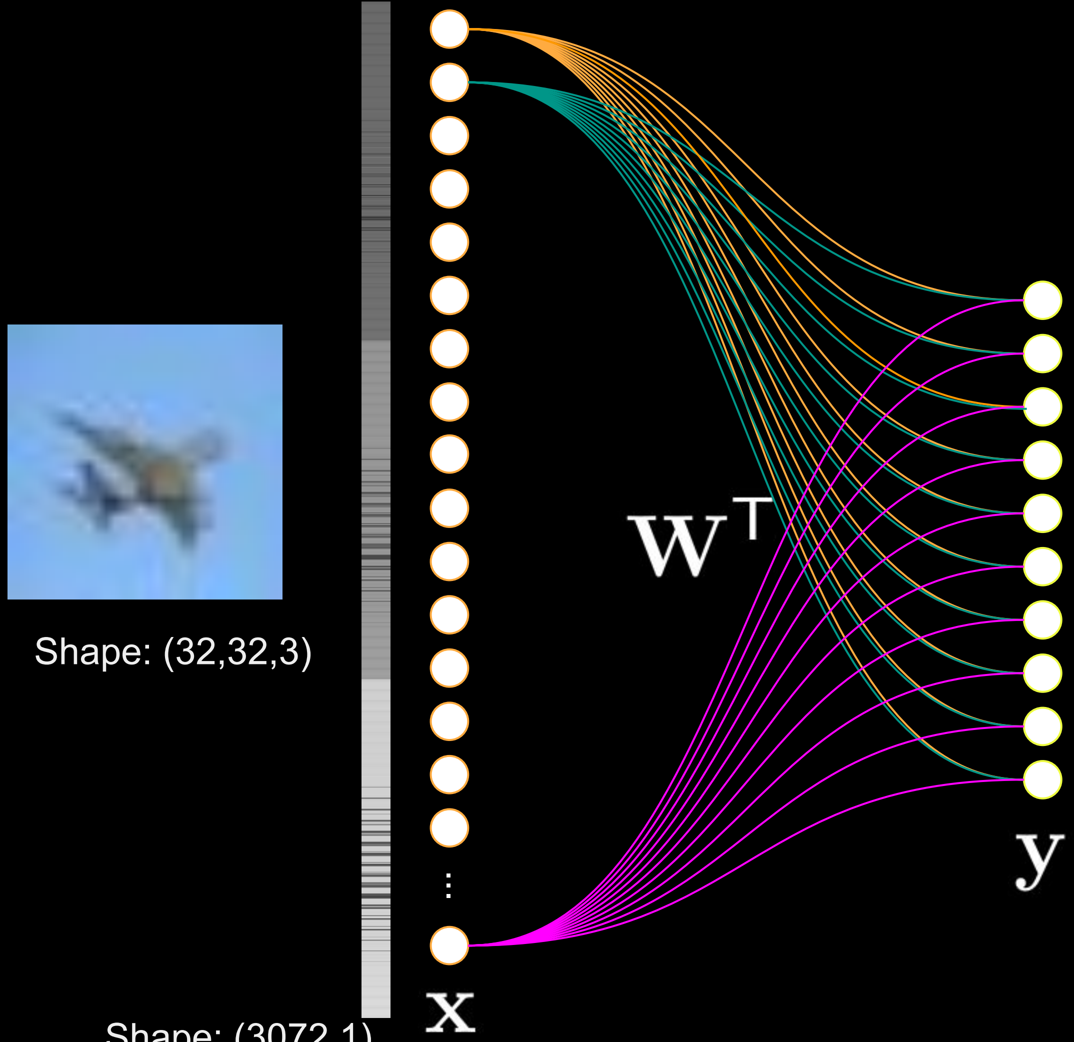
Shape: (1024,1,3)

Shape: (3072,1)

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$



$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$



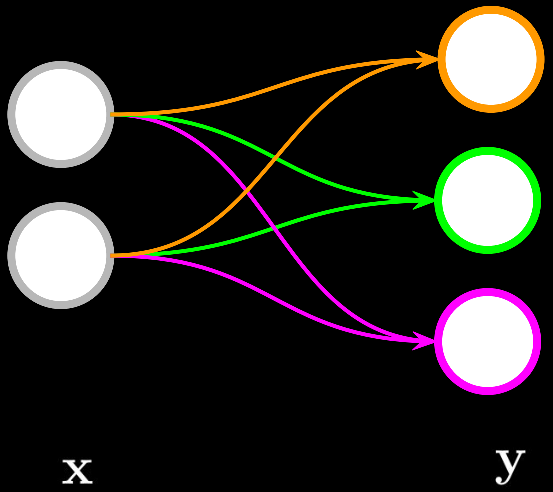

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3072, 10)

    def forward(self, x):
        x = self.fc1(x)
        return x
```

Loss Functions
(How Good is the Model?)

Loss Function

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

How good or bad are the current parameters?



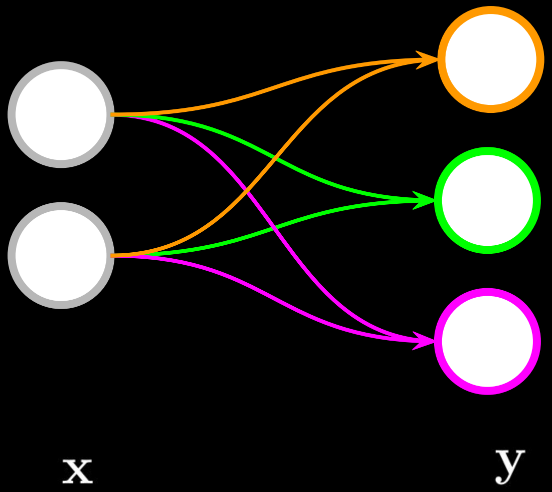
Loss Function

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

How good or bad are the current parameters?

Cross-Entropy Loss (Softmax Classifier)

- Interpret outputs **y** as probabilities for each class.
  - (unnormalised log-probabilities)
  - e.g. apply Softmax function to get probabilities



$$L = -\boxed{y_{\text{true}}} + \log \sum_j e^{y_j}$$

score assigned to true class

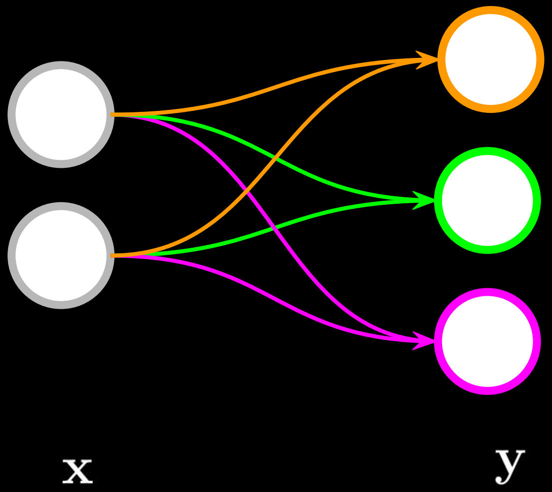
$$L = -\log\left(\frac{e^{\boxed{y_{\text{true}}}}}{\sum_j e^{y_j}}\right)$$

Loss Function Example 1

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

True class: “0”

$$\begin{aligned}
\mathbf{y} &= (5, -10, -10)^{\mathsf{T}} \\
L &= -5 + \log\left(e^5 + e^{-10} + e^{-10}\right) \\
&= -5 + \log 148.4132 \\
&= -5 + 5.000000276 \\
&\approx 0
\end{aligned}$$

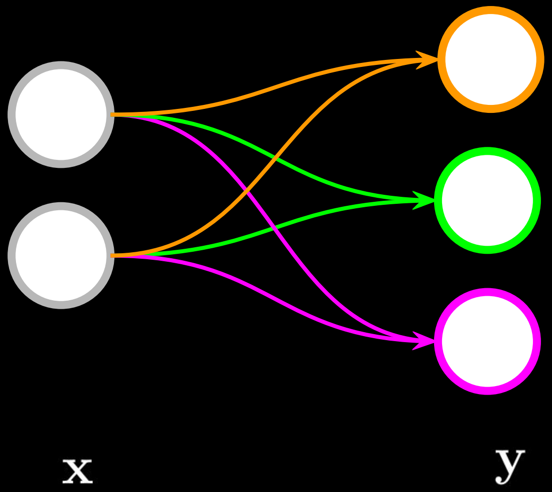


Loss Function Example 2

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

True class: “1”

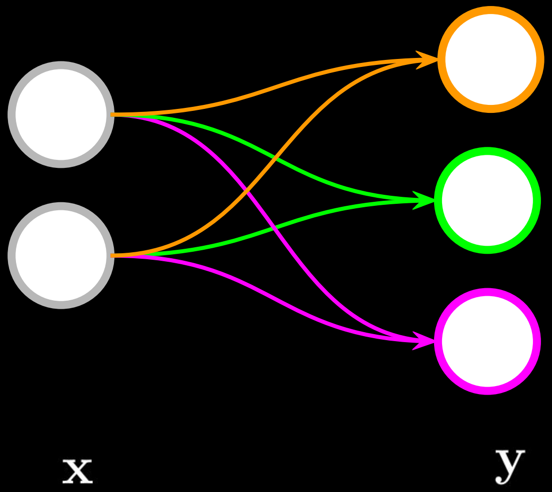
$$\begin{aligned}
\mathbf{y} &= (5, -10, -10)^{\mathsf{T}} \\
L &= 10 + \log\left(e^{5} + e^{-10} + e^{-10}\right) \\
&= 10 + \log 148.4132 \\
&= 10 + 5.000000276 \\
&\approx 15
\end{aligned}$$



Cross Entropy Loss Intuition

$$L = \boxed{-y_{\text{true}}} + \boxed{\log \sum_j e^{y_j}}$$

approximates a max function!



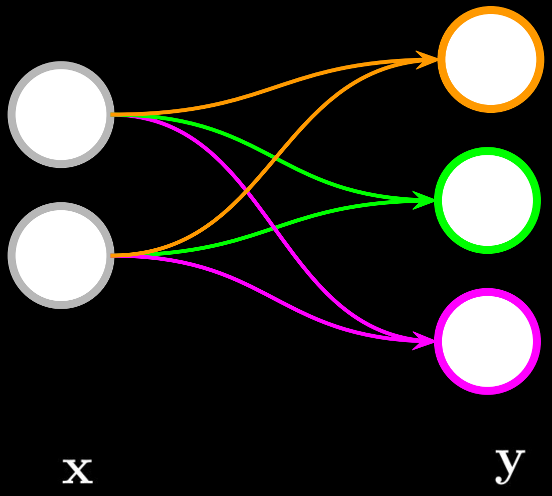
Cross Entropy Loss Intuition

$$L = \boxed{-y_{\text{true}}} + \boxed{\log \sum_j e^{y_j}}$$

approximates a max function!

```
p = np.random.randn(5)*10
print(p)
np.log(np.sum(np.exp(p)))
```

```
[ -5.7700444  -13.26877559  -6.04029885   2.19204188   6.73428813]
6.744887755211229
```

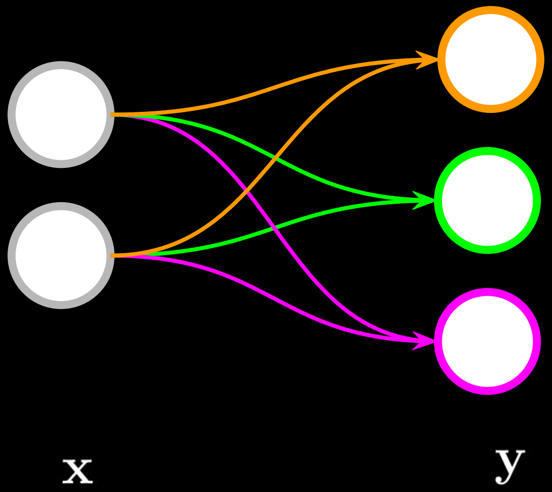


Cross Entropy Loss Intuition

$$L = \boxed{-y_{\text{true}}} + \boxed{\log \sum_j e^{y_j}}$$

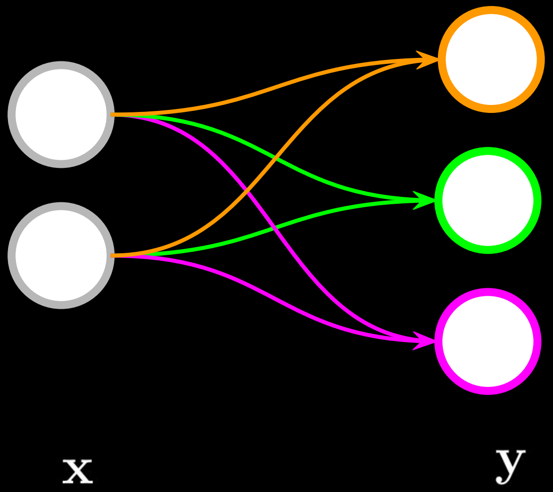
approximates a max function!

→ Minimum Loss when: **highest score for correct class!**



Cross Entropy Loss Intuition

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$



→ Minimum Loss when: **highest score for correct class!**

→ minimize average loss for all training samples

Training
Finding Good Weights (and Biases)

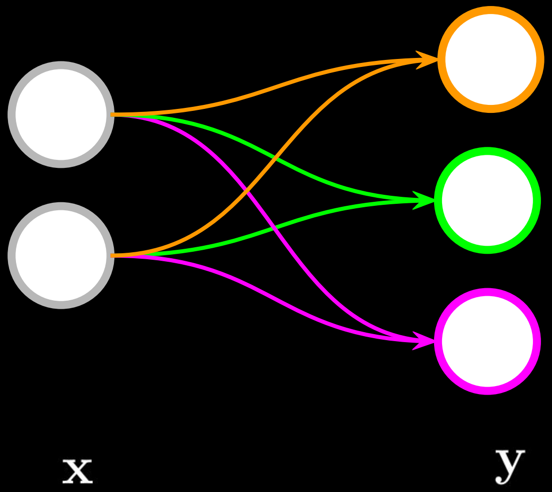
How do we find the best (W,b)?

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

Objective: minimize average loss for all training samples.
But how? Some ideas:

- Random search
  - randomly choose (W,b), and remember the best



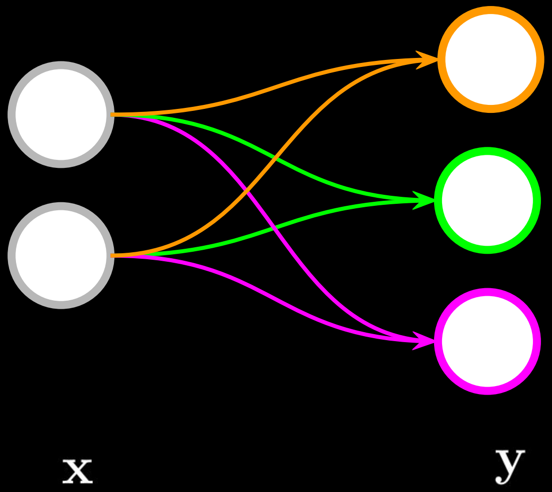
How do we find the best (W,b)?

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}$$

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

Objective: minimize average loss for all training samples.
But how? Some ideas:

- Random search
  - randomly choose (W,b), and remember the best
- Random local search
  - randomly change (W,b) slowly by adding a small increment, check if that made it better



How do we find the best (W,b)?

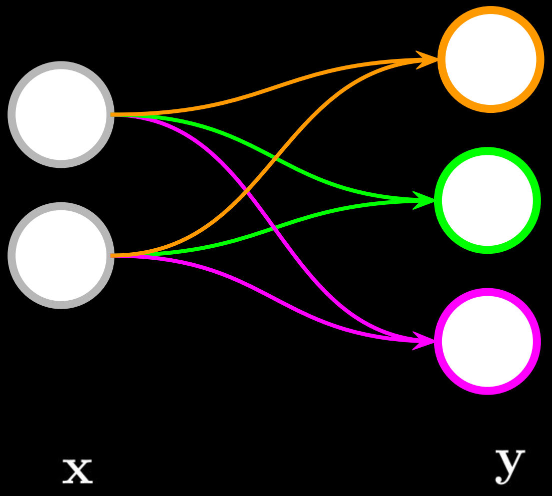
$$\mathbf{y} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$

$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

Objective: minimize average loss for all training samples.
But how? Some ideas:

- Random search
  - randomly choose (W,b), and remember the best
- Random local search
  - randomly change (W,b) slowly by adding a small increment, check if that made it better
- Follow the gradient
  - systematically change (W,b) by computing derivatives

"Gradient Descent"



Gradient Descent

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

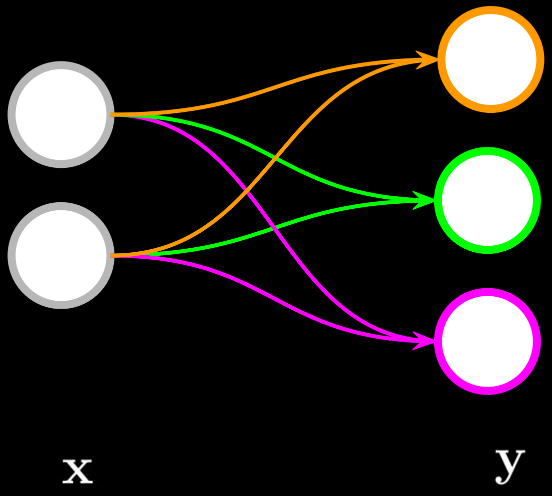
$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

learning rate
step size

$$\mathbf{W}^{+} = \mathbf{W} - \boxed{s} \cdot \boxed{\frac{\partial \mathbf{L}}{\partial \mathbf{W}}}$$

derivative of loss
with respect to
the weights

$$\mathbf{b}^{+} = \mathbf{b} - s \cdot \frac{\partial \mathbf{L}}{\partial \mathbf{b}}$$



Gradient Descent

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

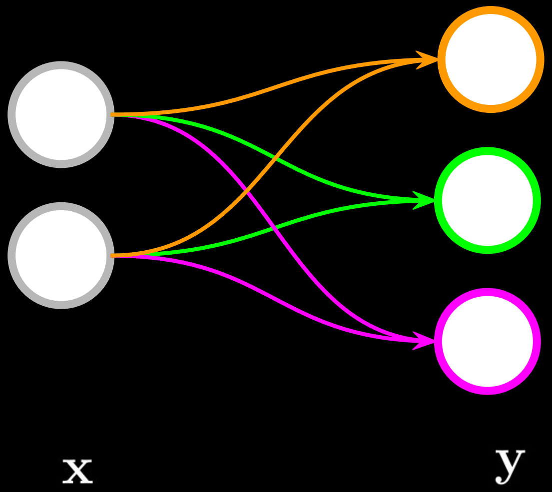
$$L = -y_{\text{true}} + \log \sum_j e^{y_j}$$

learning rate
step size

$$\mathbf{W}^{+} = \mathbf{W} - \boxed{s} \cdot \boxed{\frac{\partial \mathbf{L}}{\partial \mathbf{W}}}$$

derivative of loss
with respect to
the weights

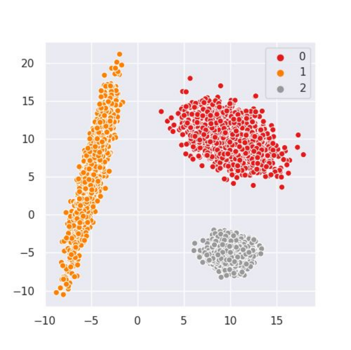
$$\mathbf{b}^{+} = \mathbf{b} - s \cdot \frac{\partial \mathbf{L}}{\partial \mathbf{b}}$$

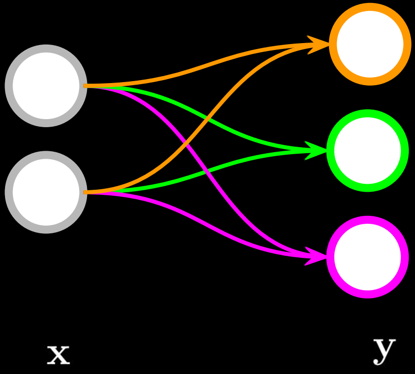


Fortunately, automatic differentiation is part of most DL libraries!
Same for various optimization methods!

Training a simple linear classifier

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$$

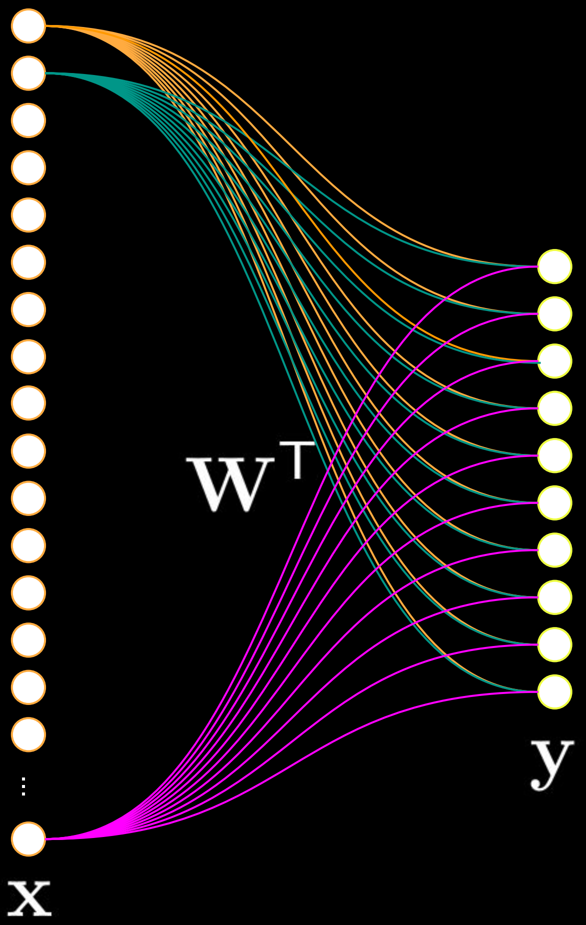




And Now: Actual Neural Networks

Missing Ingredient

Nonlinear activation function



$$\mathrm{b} + \sum_i w_{i,0} \cdot x_i$$

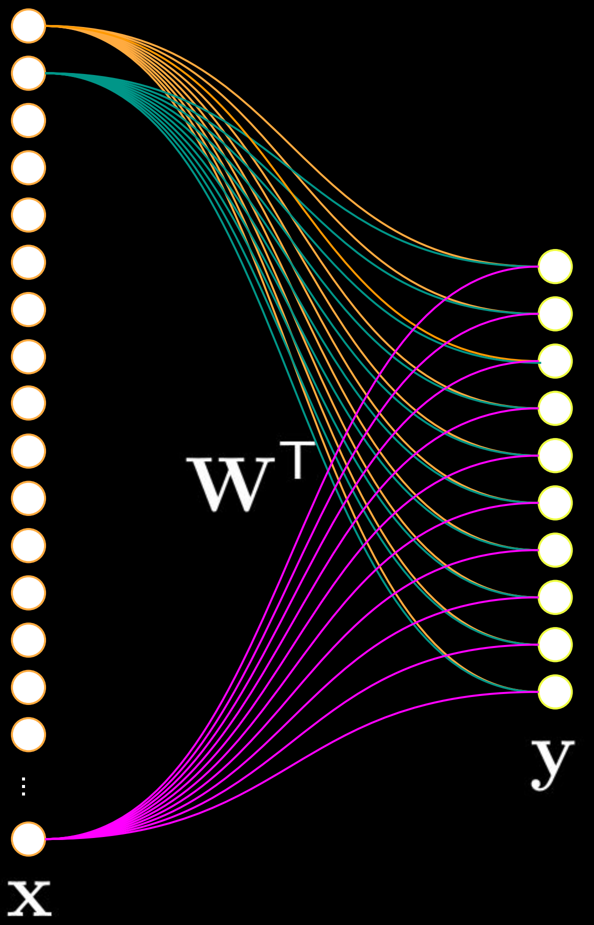
$$\mathrm{b} + \sum_i w_{i,1} \cdot x_i$$

$$\mathrm{b} + \sum_i w_{i,2} \cdot x_i$$

Missing Ingredient

(nonlinear) activation function

- Linear models are often overly simple
- Enables meaningful “stacking” of layers
  → **deep** networks



$$a\Big(\mathbf{b} + \sum_i w_{i,0} \cdot x_i\Big)$$

$$a\Big(\mathbf{b} + \sum_i w_{i,1} \cdot x_i\Big)$$

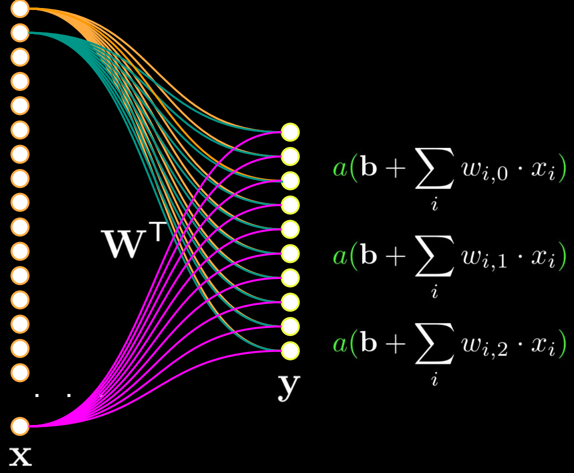
$$a\Big(\mathbf{b} + \sum_i w_{i,2} \cdot x_i\Big)$$

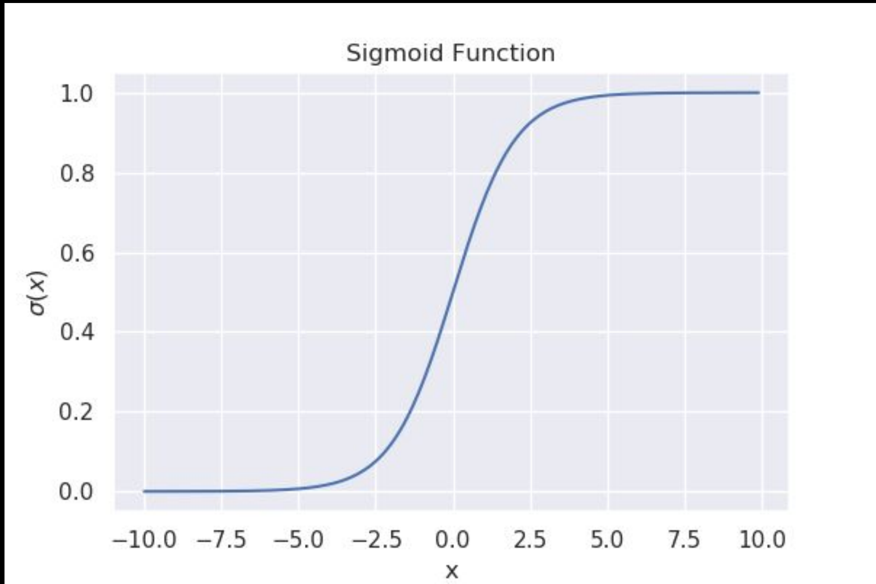
Missing Ingredient

(nonlinear) activation function

- Linear models are often overly simple
- Enables meaningful “stacking” of layers
  → **deep** networks
- Historically: sigmoid function

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





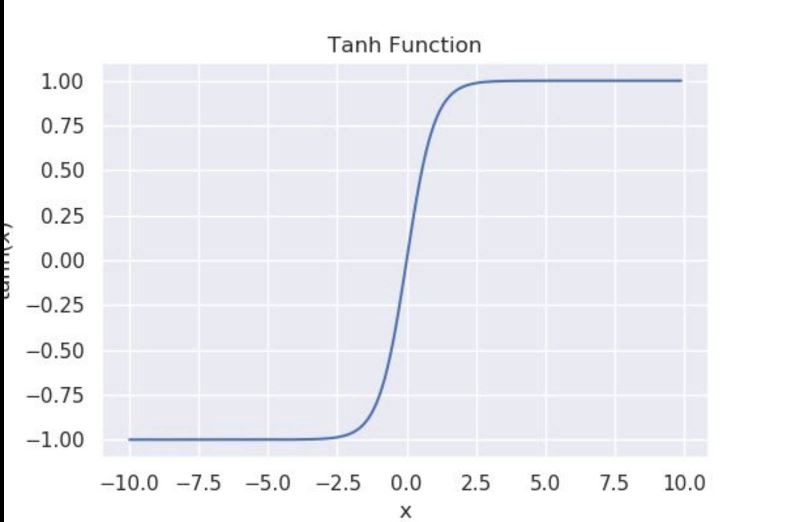
Missing Ingredient

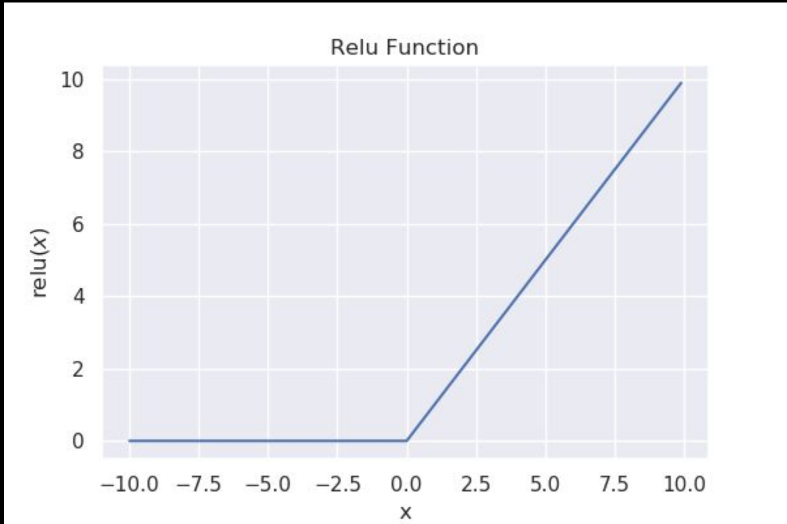
(nonlinear) activation function

- Linear models are often overly simple
- Enables meaningful “stacking” of layers → **deep** networks
- Historically: sigmoid function

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

- Many other choices:
  - tanh(x)
  - Rectified Linear Unit ReLU = max(0,x)
  - ...





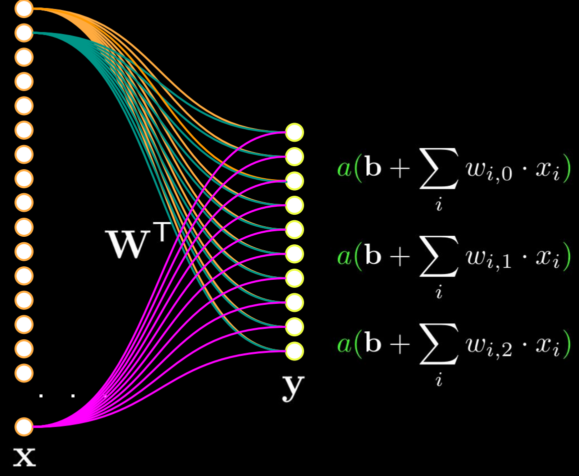
Missing Ingredient

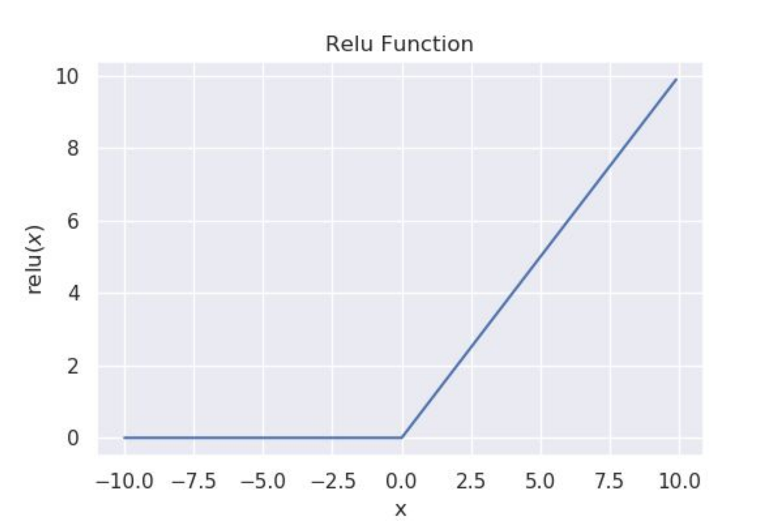
(nonlinear) activation function

- Linear models are often overly simple
- Enables meaningful “stacking” of layers → **deep** networks
- Historically: sigmoid function

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

- Many other choices:
    - tanh(x)
    - **Rectified Linear Unit ReLU = max(0,x)**
    - ReLU is most commonly used



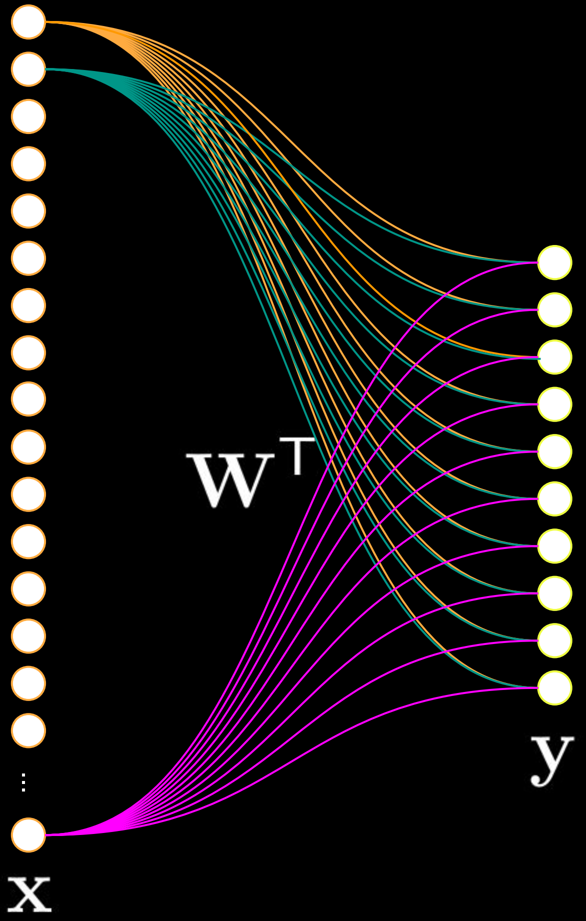


Missing Ingredient

(nonlinear) activation function

- Linear models are often overly simple
- Enables meaningful “stacking” of layers
  → **deep** networks
- Historically: sigmoid function

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



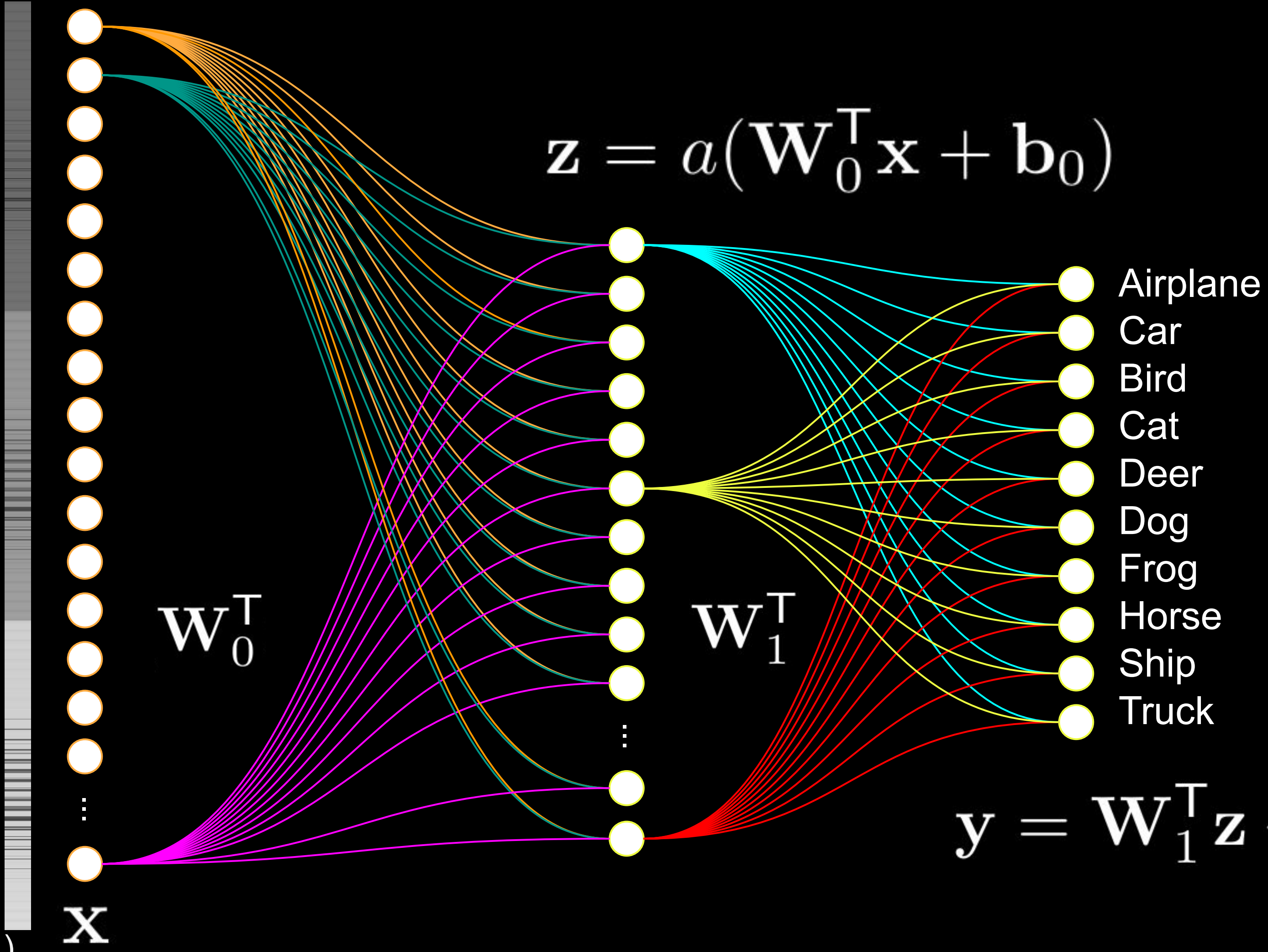
$$a\left(\mathbf{b} + \sum_i w_{i,0} \cdot x_i\right)$$

$$a\left(\mathbf{b} + \sum_i w_{i,1} \cdot x_i\right)$$

$$a\left(\mathbf{b} + \sum_i w_{i,2} \cdot x_i\right)$$

Deep Networks

Shape: (32,32,3)



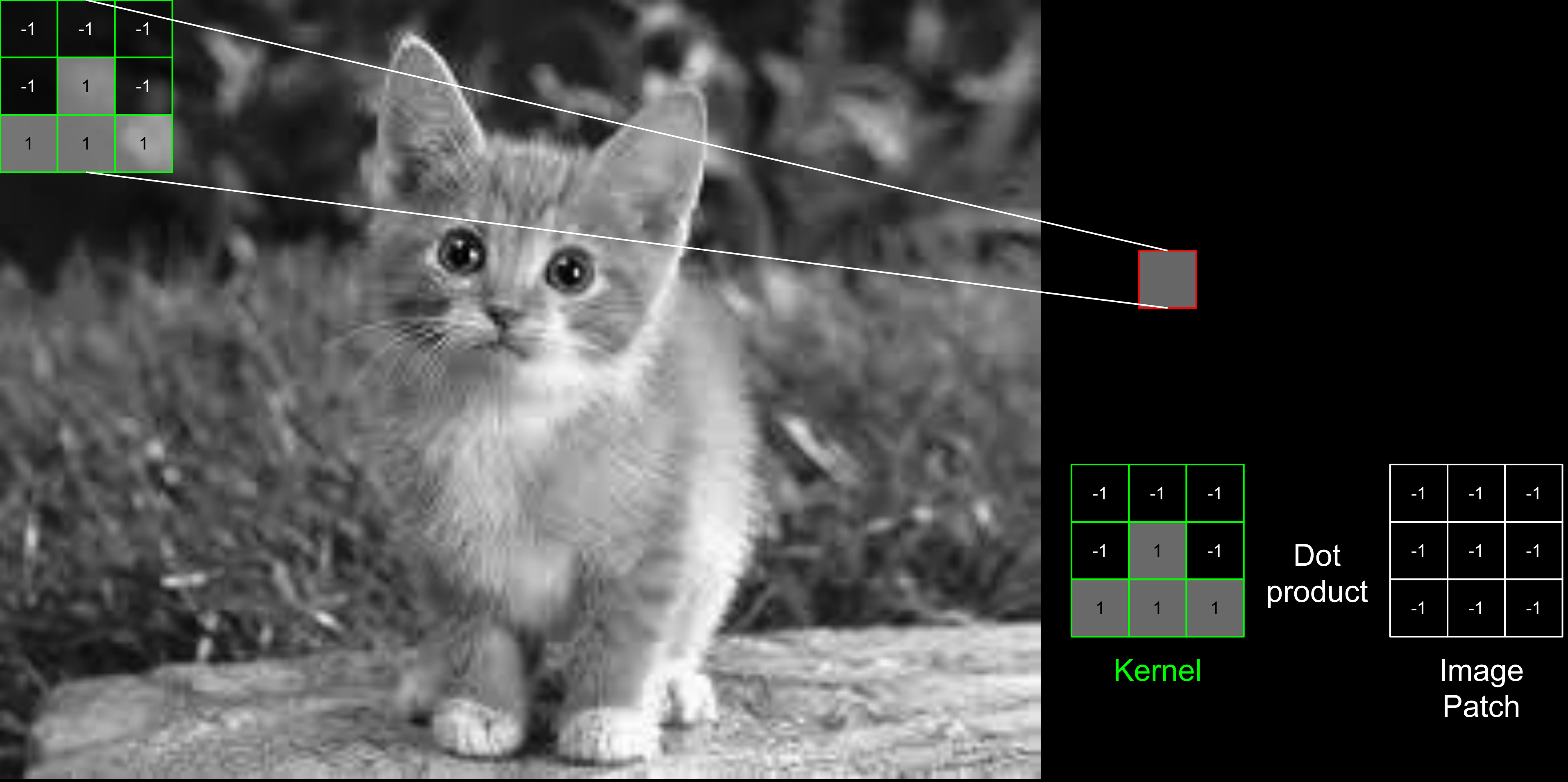
$$\mathbf{z} = a(\mathbf{W}_0^T \mathbf{x} + \mathbf{b}_0)$$

$$\mathbf{y} = \mathbf{W}_1^T \mathbf{z} + \mathbf{b}_1$$

Shape: (3072,1)

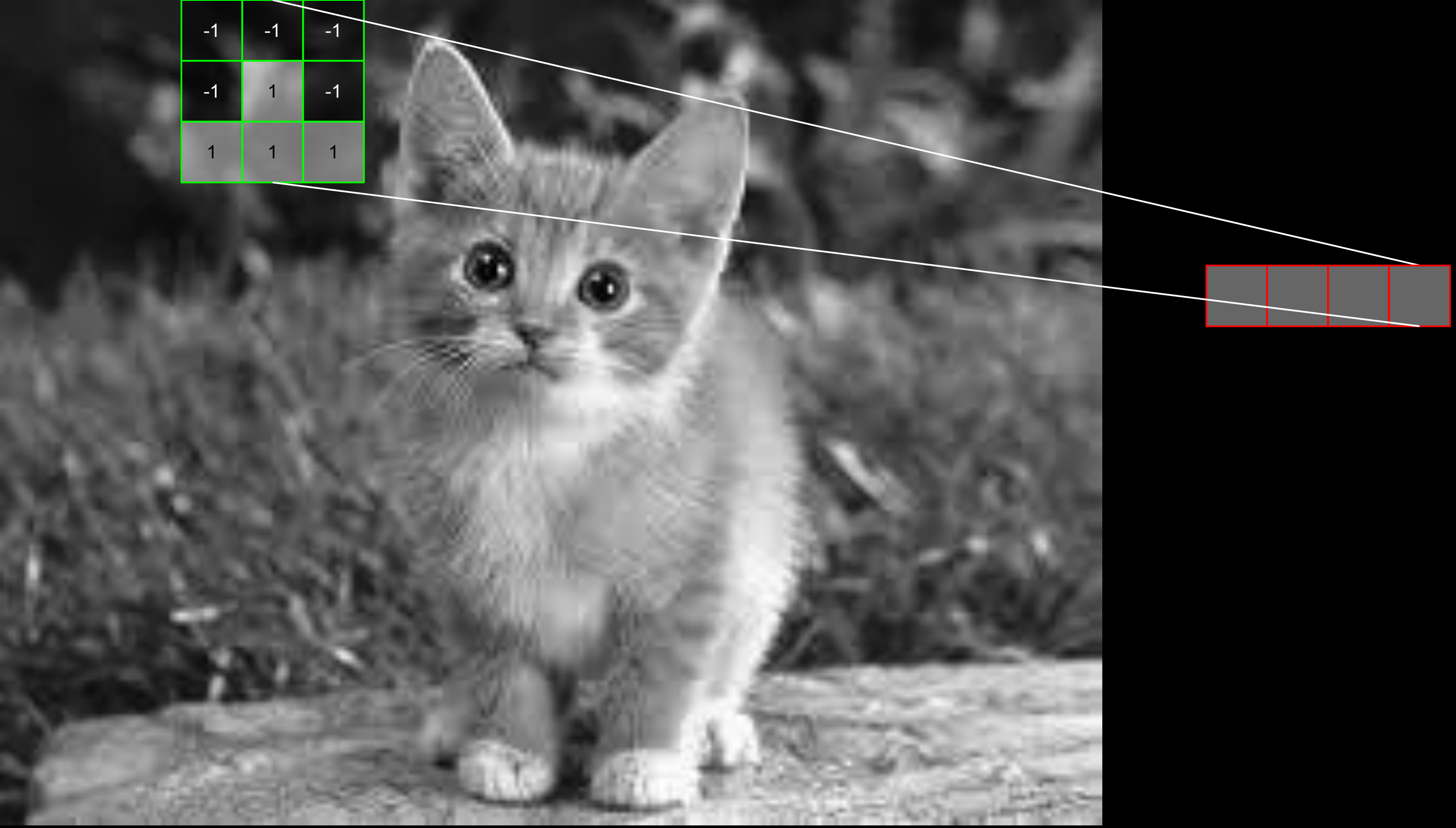
Convolutional Networks

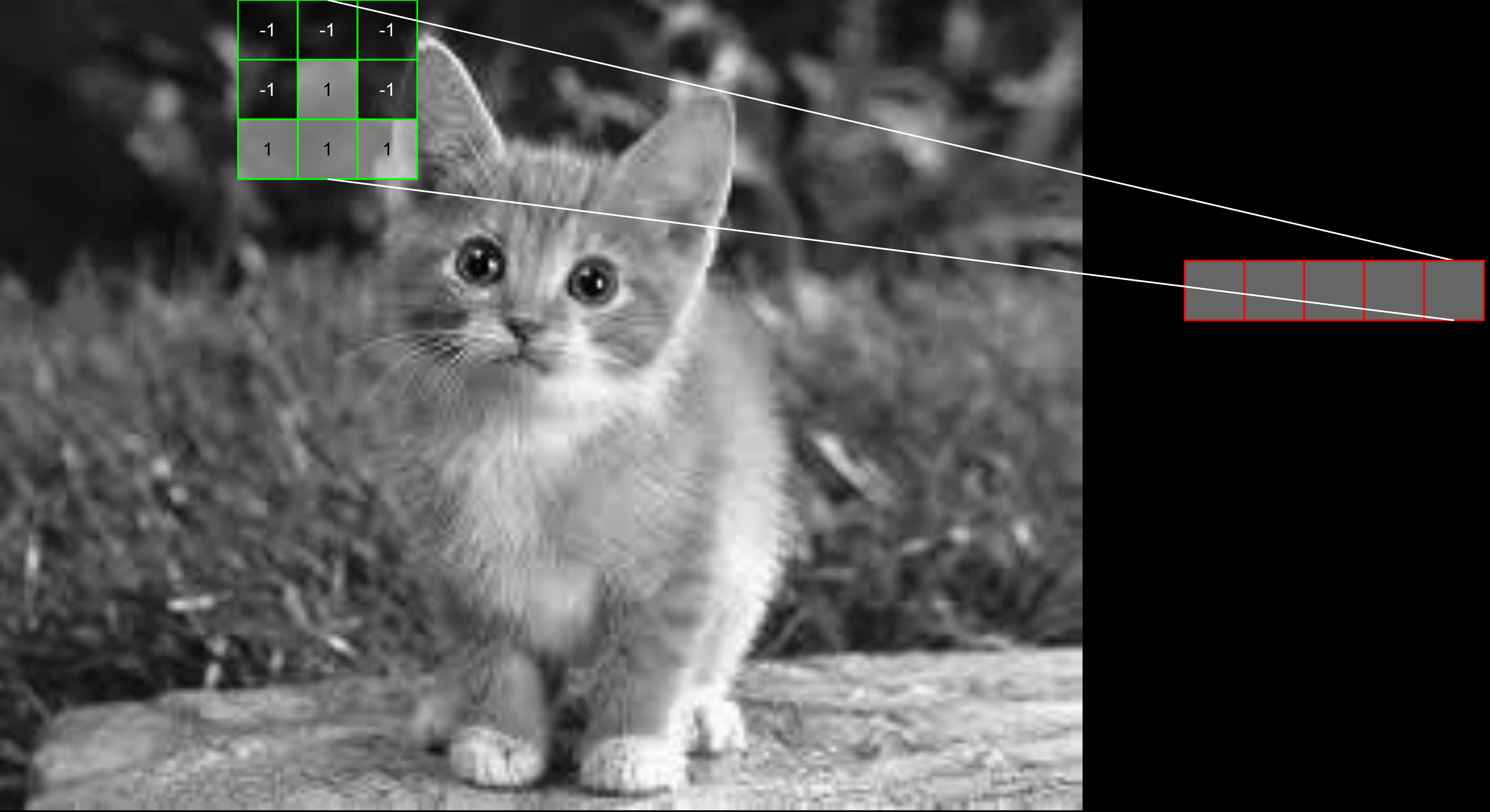
-1 -1 -1
-1 1 -1
1 1 1

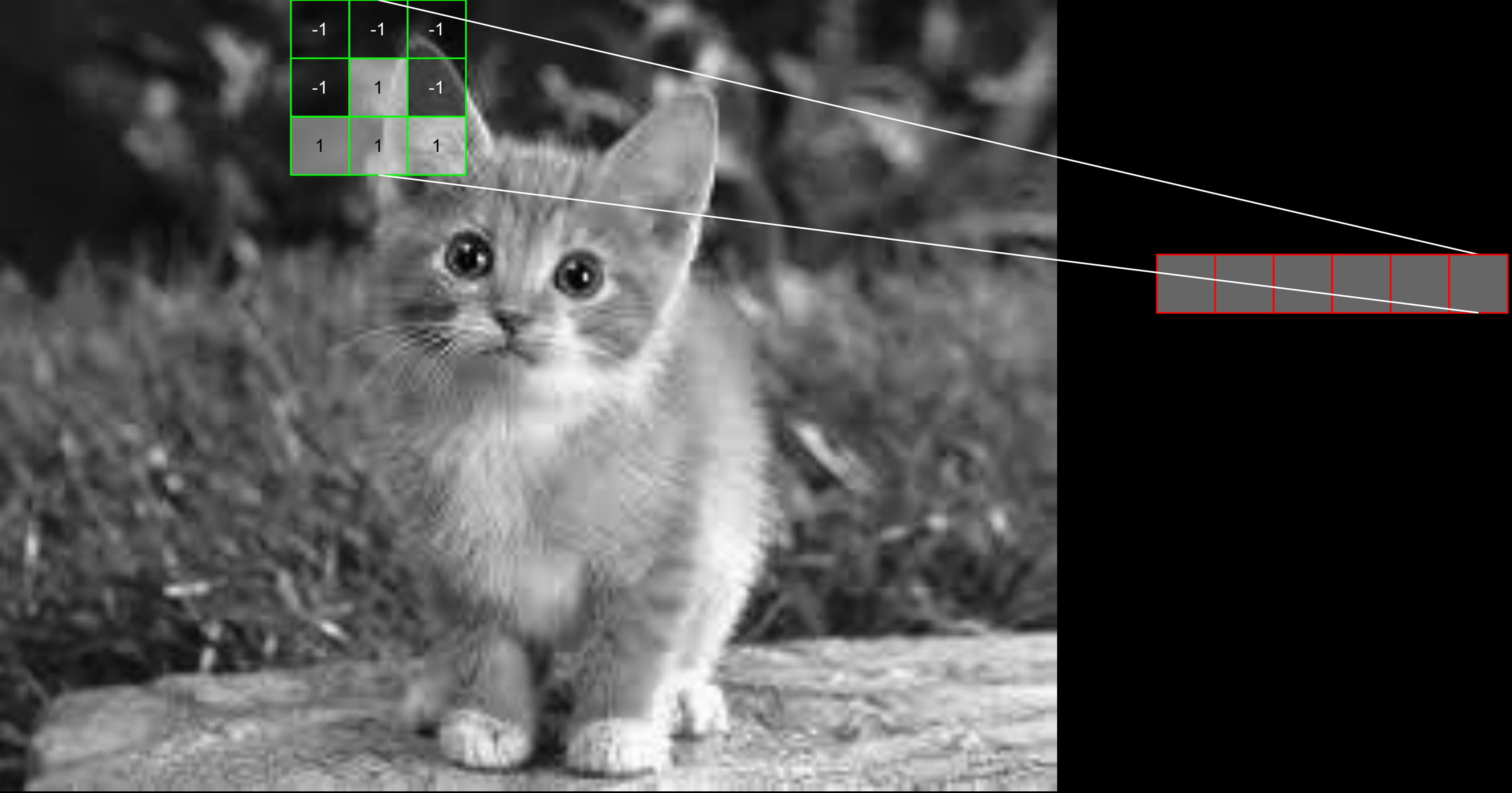

-1 -1 -1
-1 1 -1
1 1 1
Dot product
-1 -1 -1
-1 -1 -1
-1 -1 -1
Kernel
Image Patch

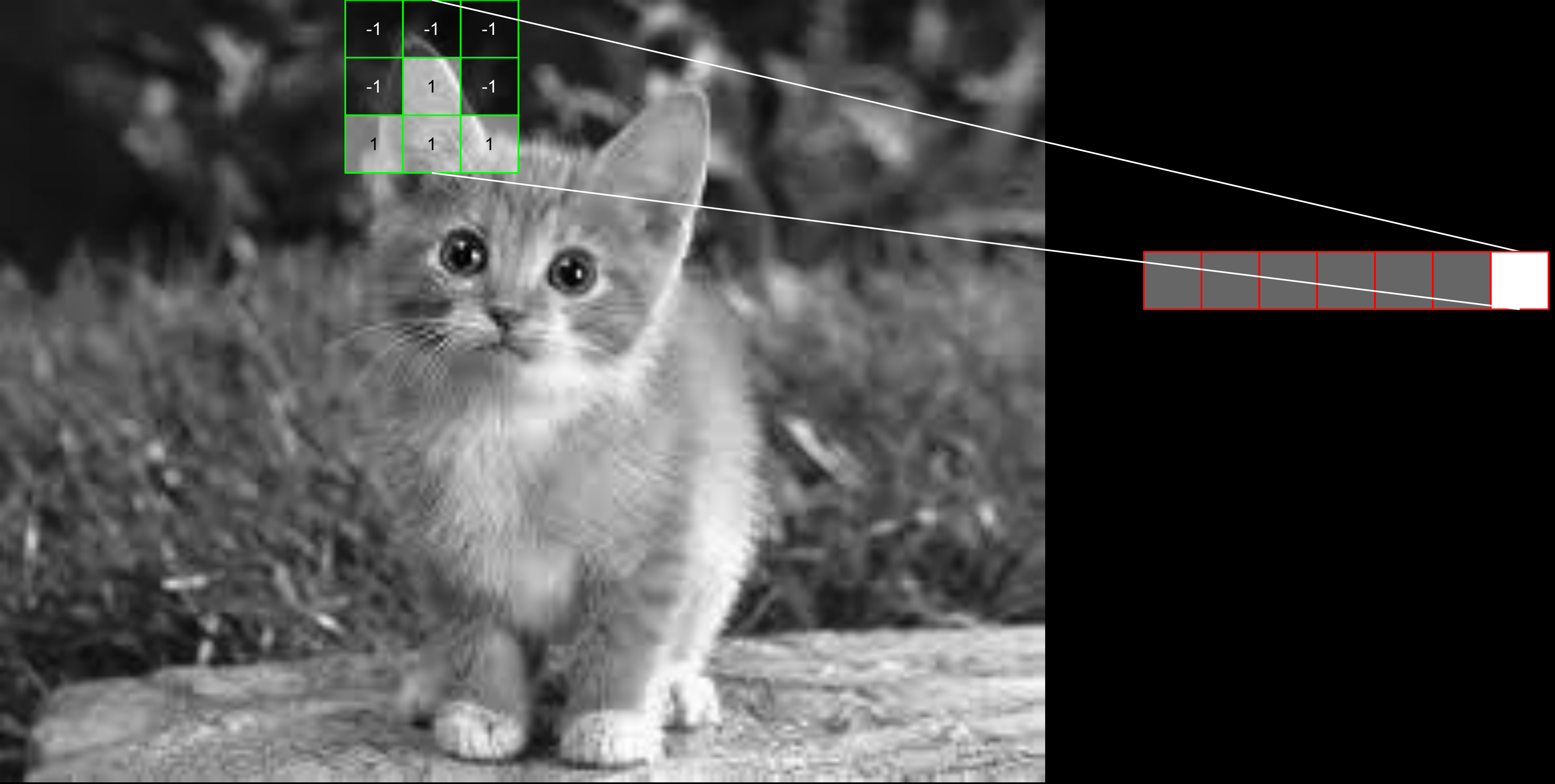

-1 -1 -1
-1 1 -1
1 1 1

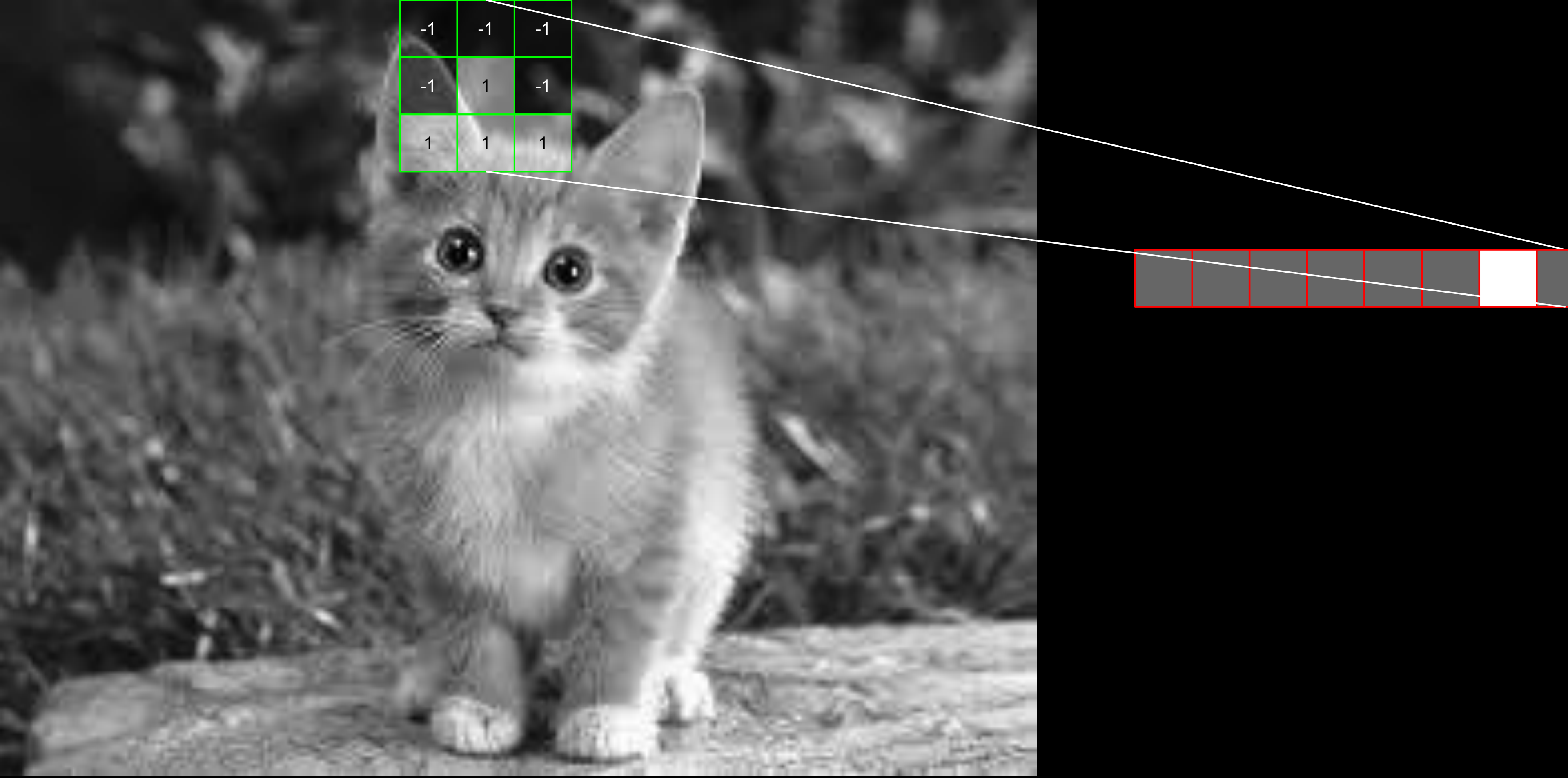

-1 -1 -1
-1 1 -1
1 1 1

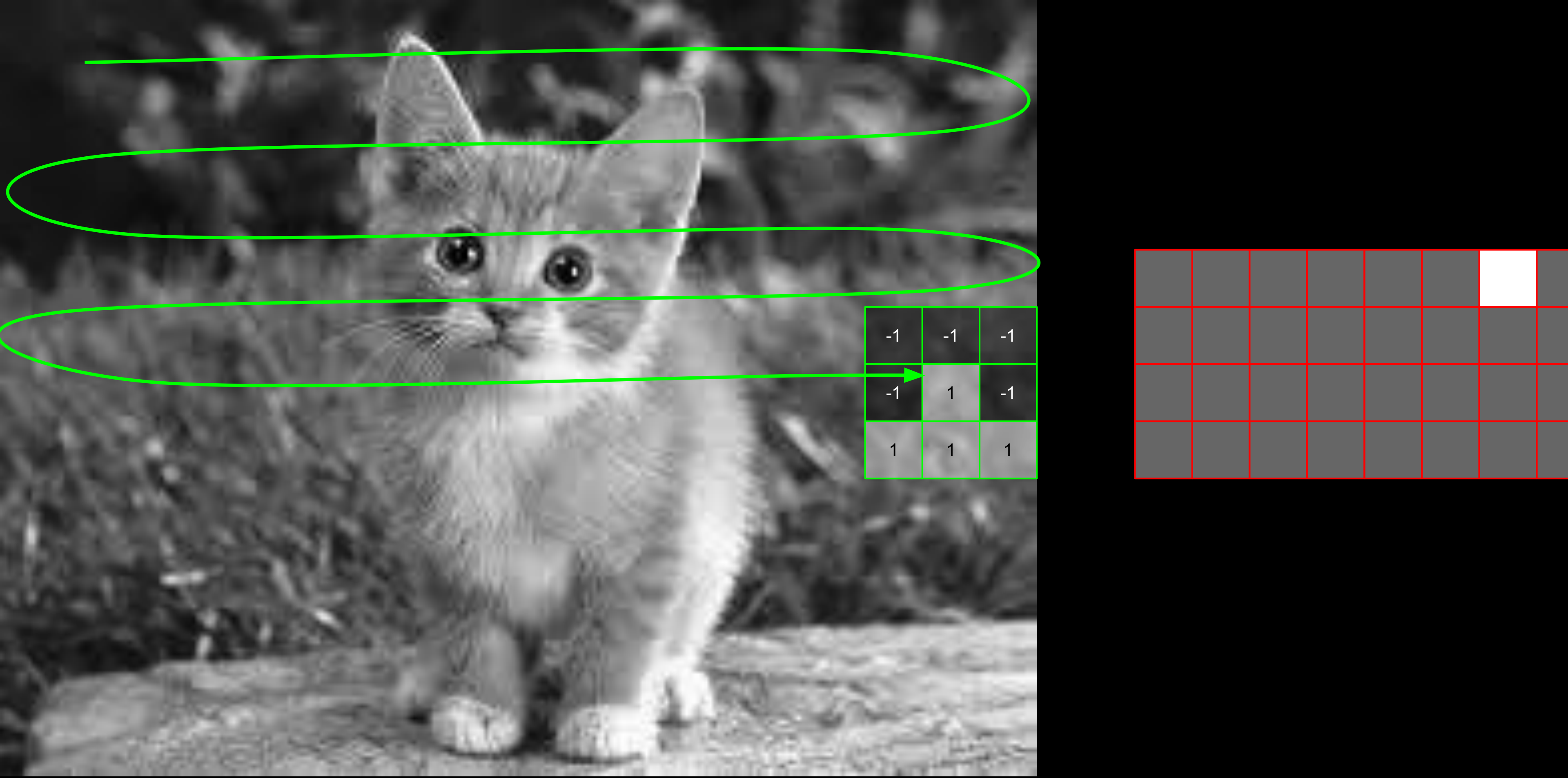

-1
-1
-1
-1
1
-1
1
1
1


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


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

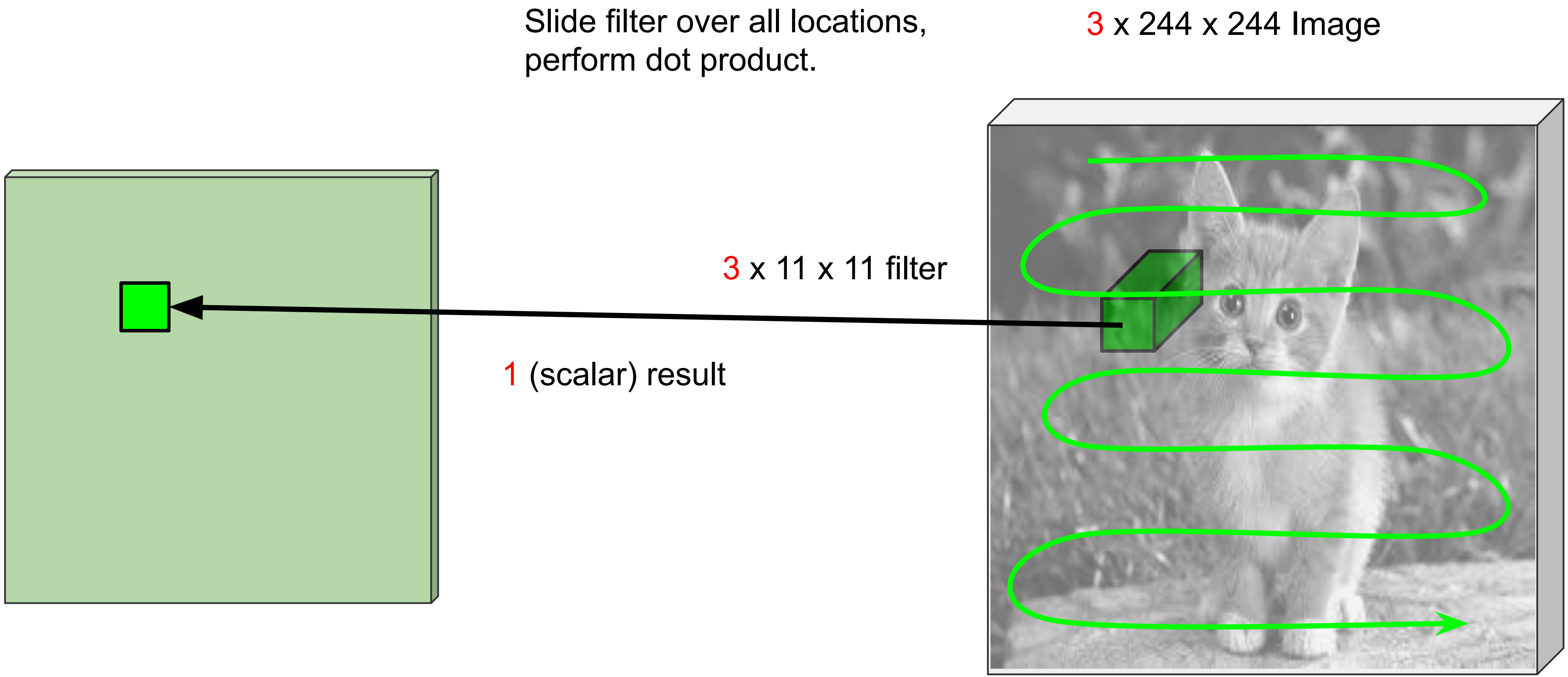

-1 -1 -1
-1 1 -1
1 1 1


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


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

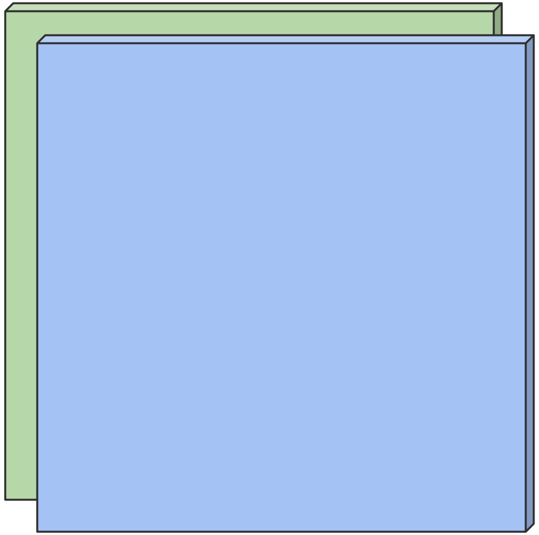
3 channels (RGB)
shape (3, 244, 244)

Convolution:
Slide filter over all locations,
perform dot product.

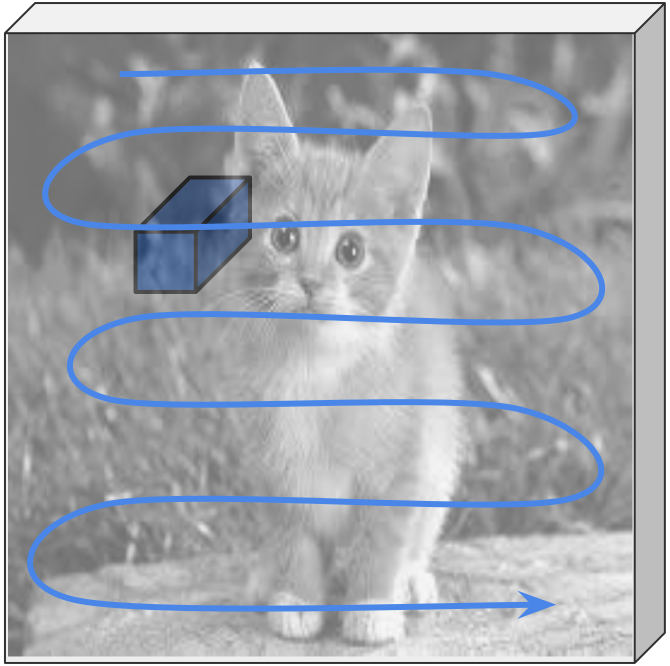


Convolution:
Slide filter over all locations, perform dot product.

3 x 244 x 244 Image

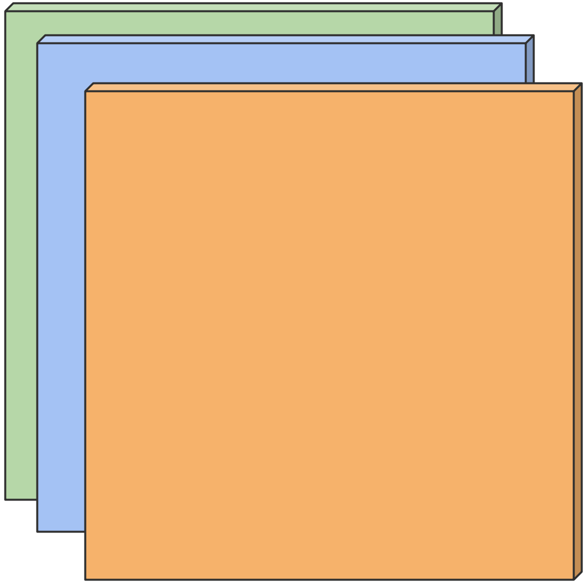


3 x 11 x 11 filter

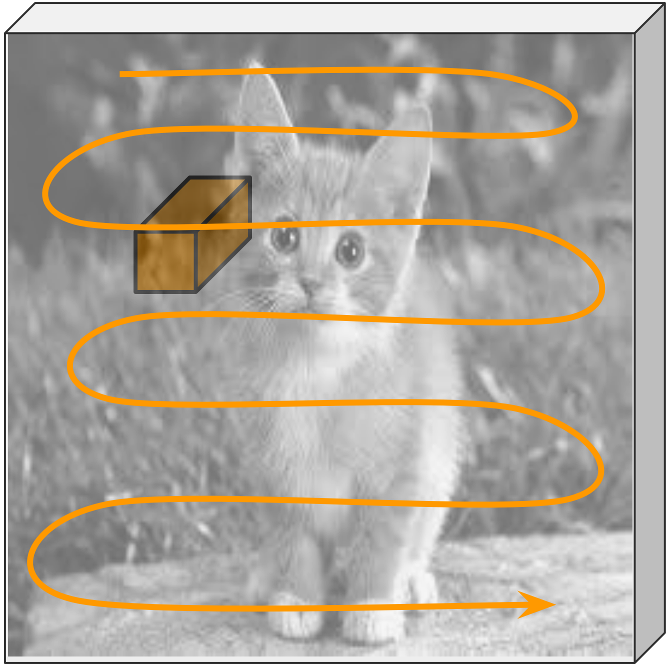


Convolution:
Slide filter over all locations,
perform dot product.

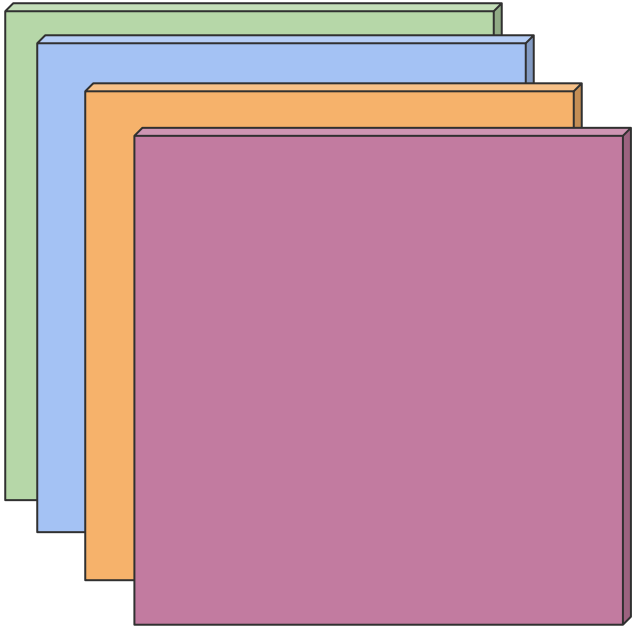
3 x 244 x 244 Image

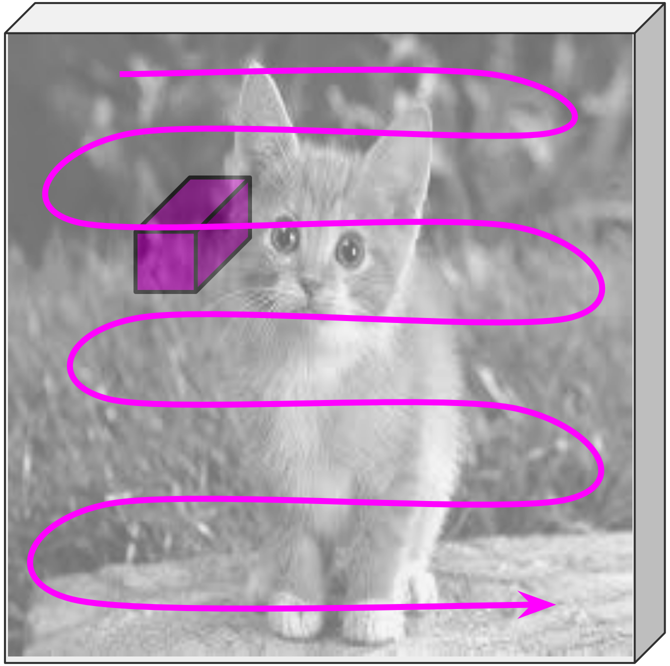


3 x 11 x 11 filter



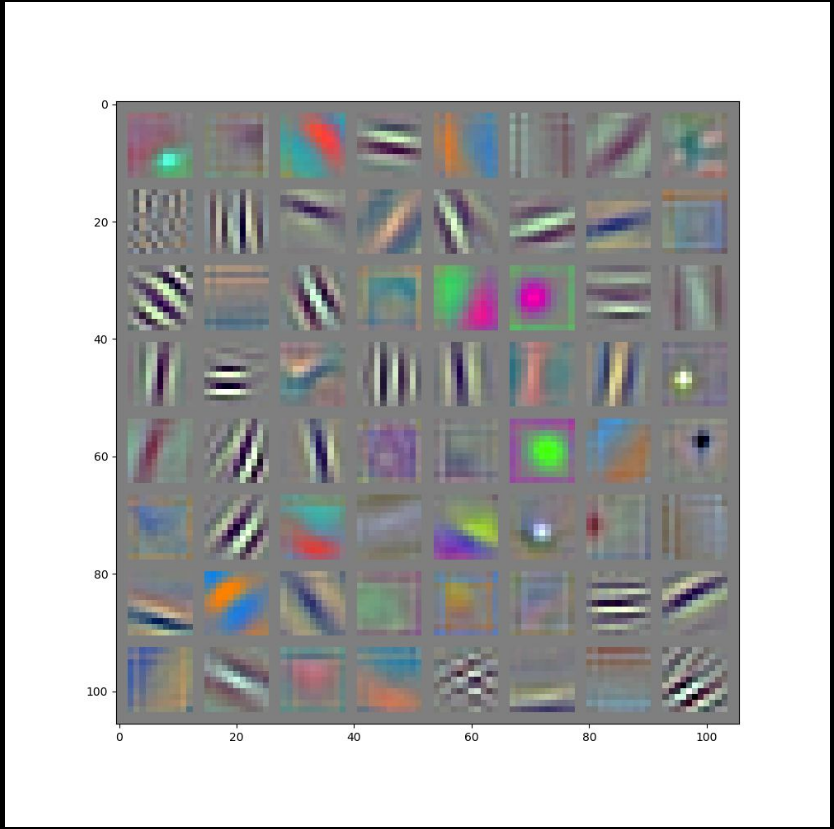
Convolution:
Slide filter over all locations, perform dot product.



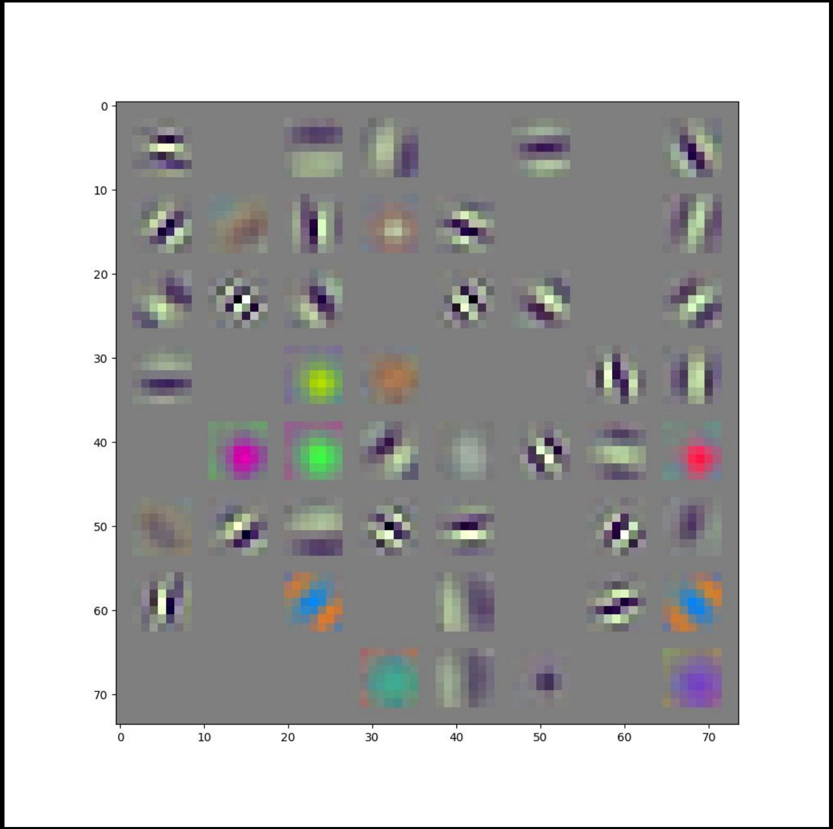


1st Convolutional Layer

Alexnet

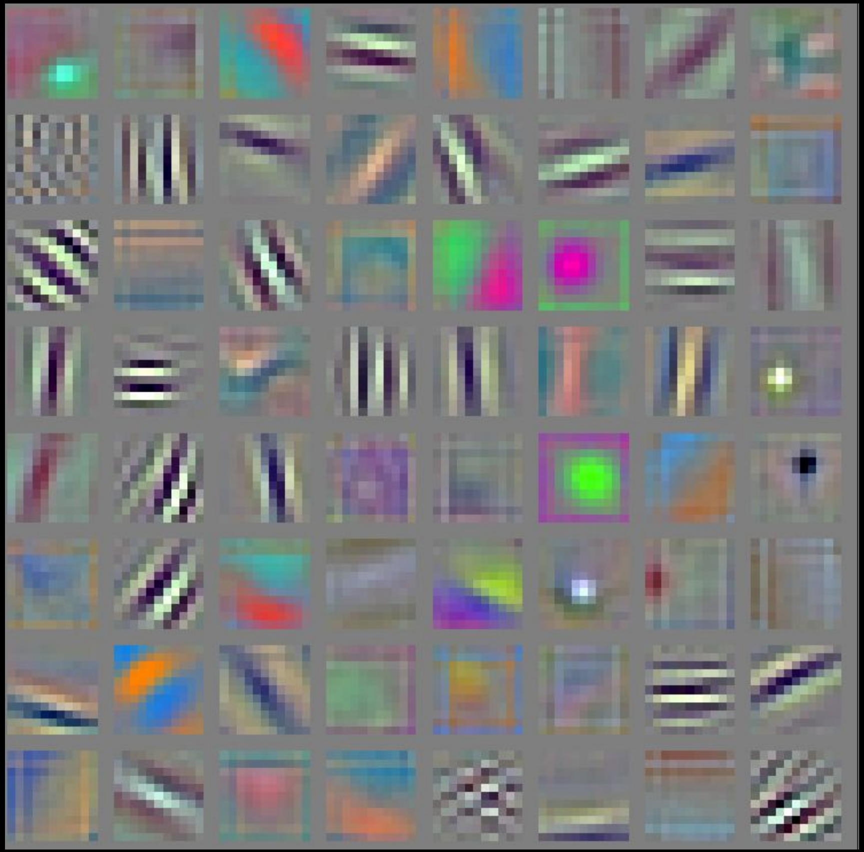


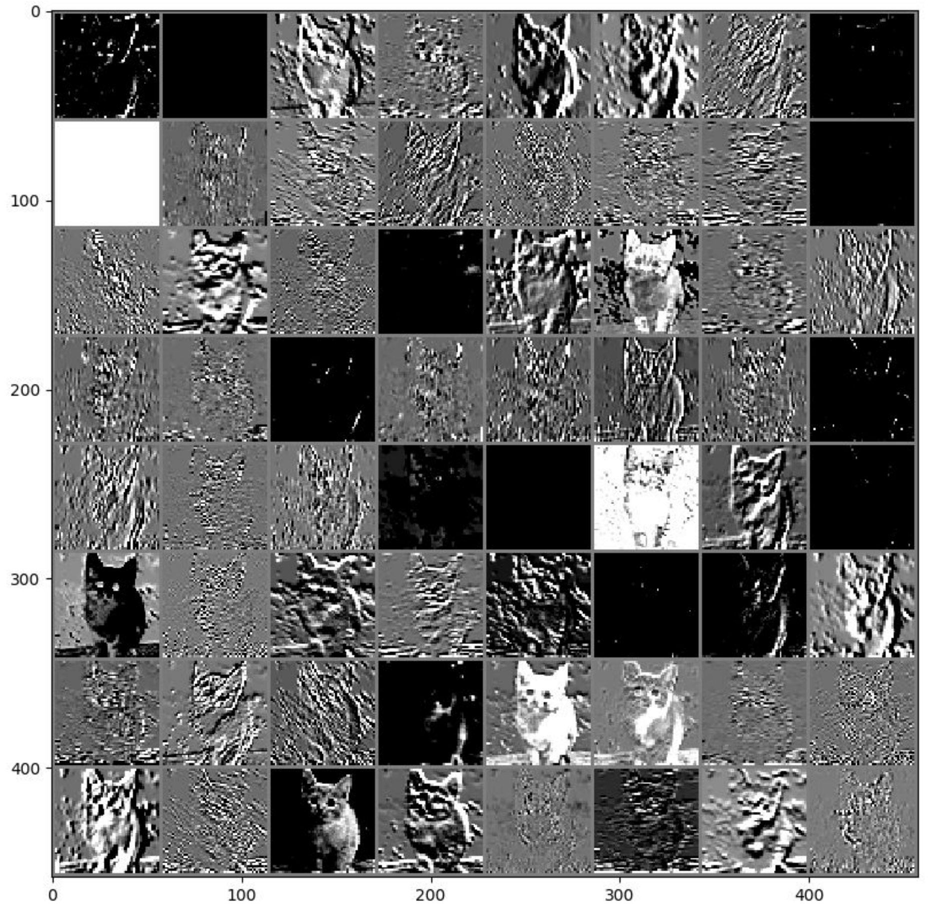
ResNeXt



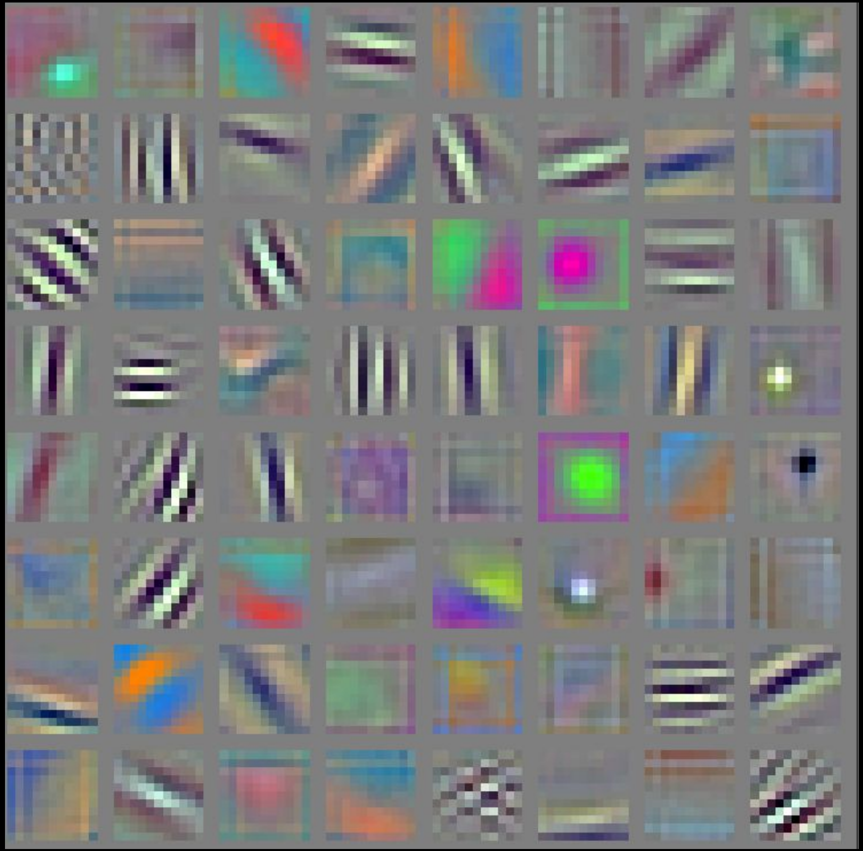
3 channels (RGB)
shape (3, 244, 244)

Alexnet Conv1: 64 filters, size (3, 11, 11)



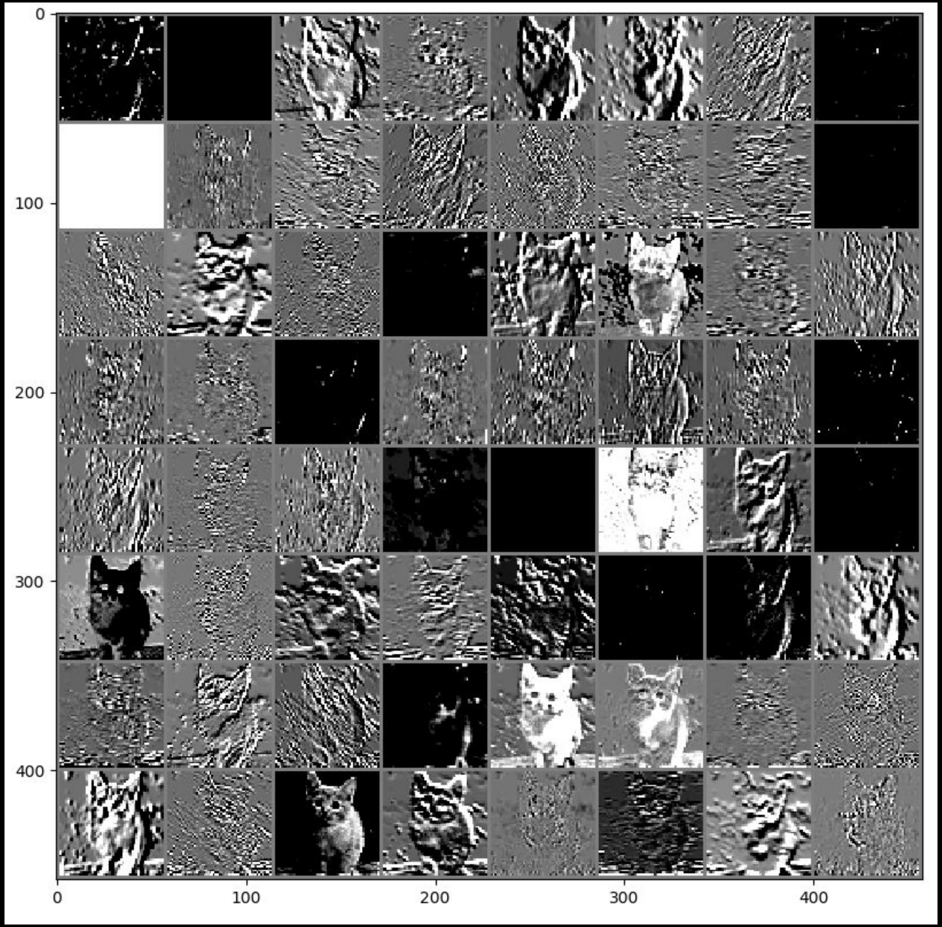


Alexnet Conv1: 64 filters, size (3, 11, 11)

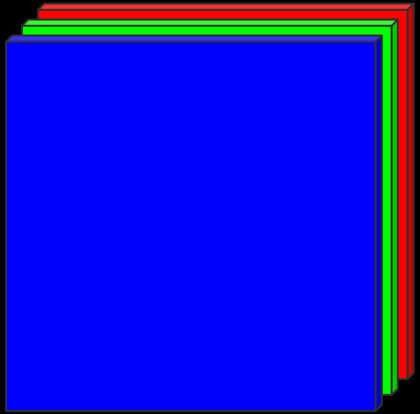


3 channels (RGB)
shape (3, 244, 244)

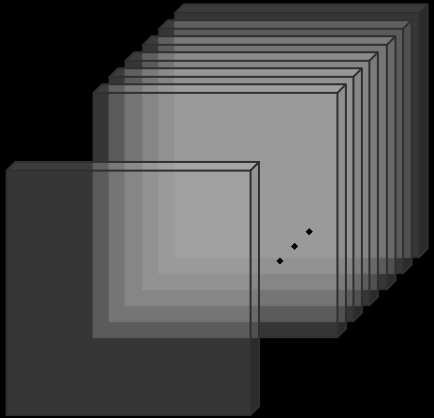
Result: (64, 55, 55)

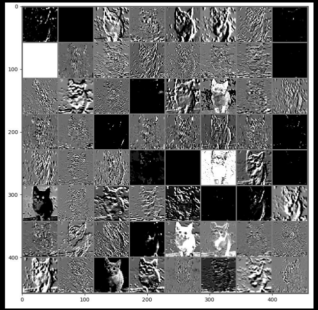


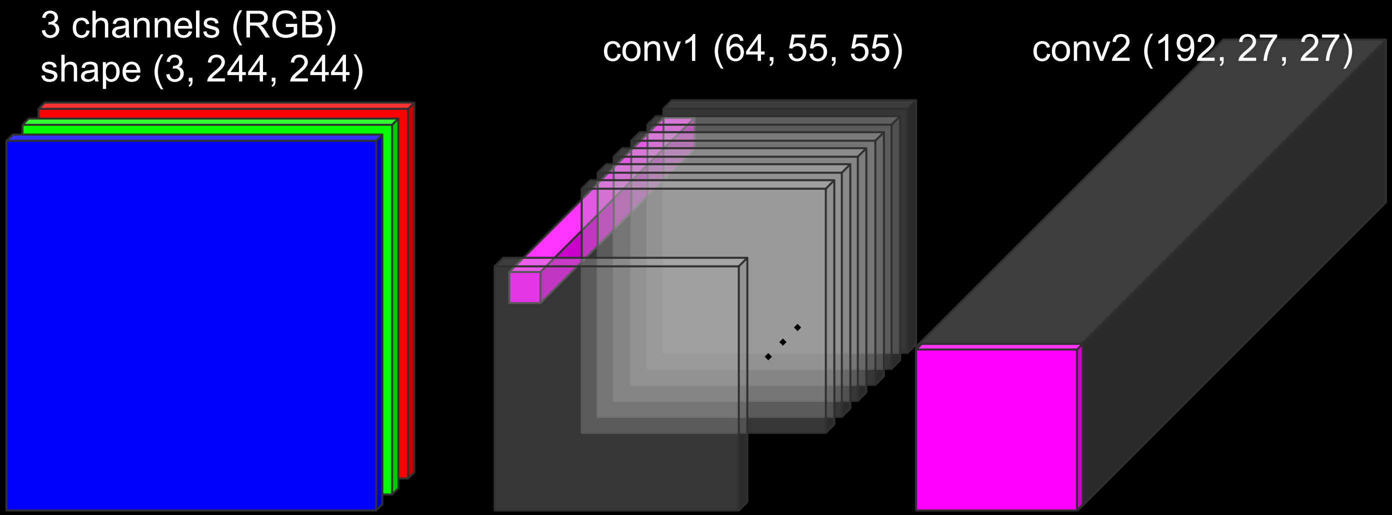
3 channels (RGB)
shape (3, 244, 244)

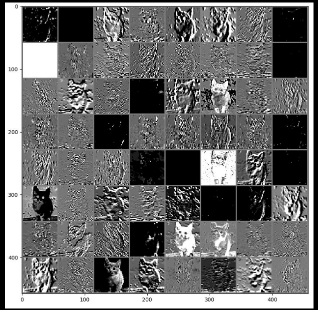


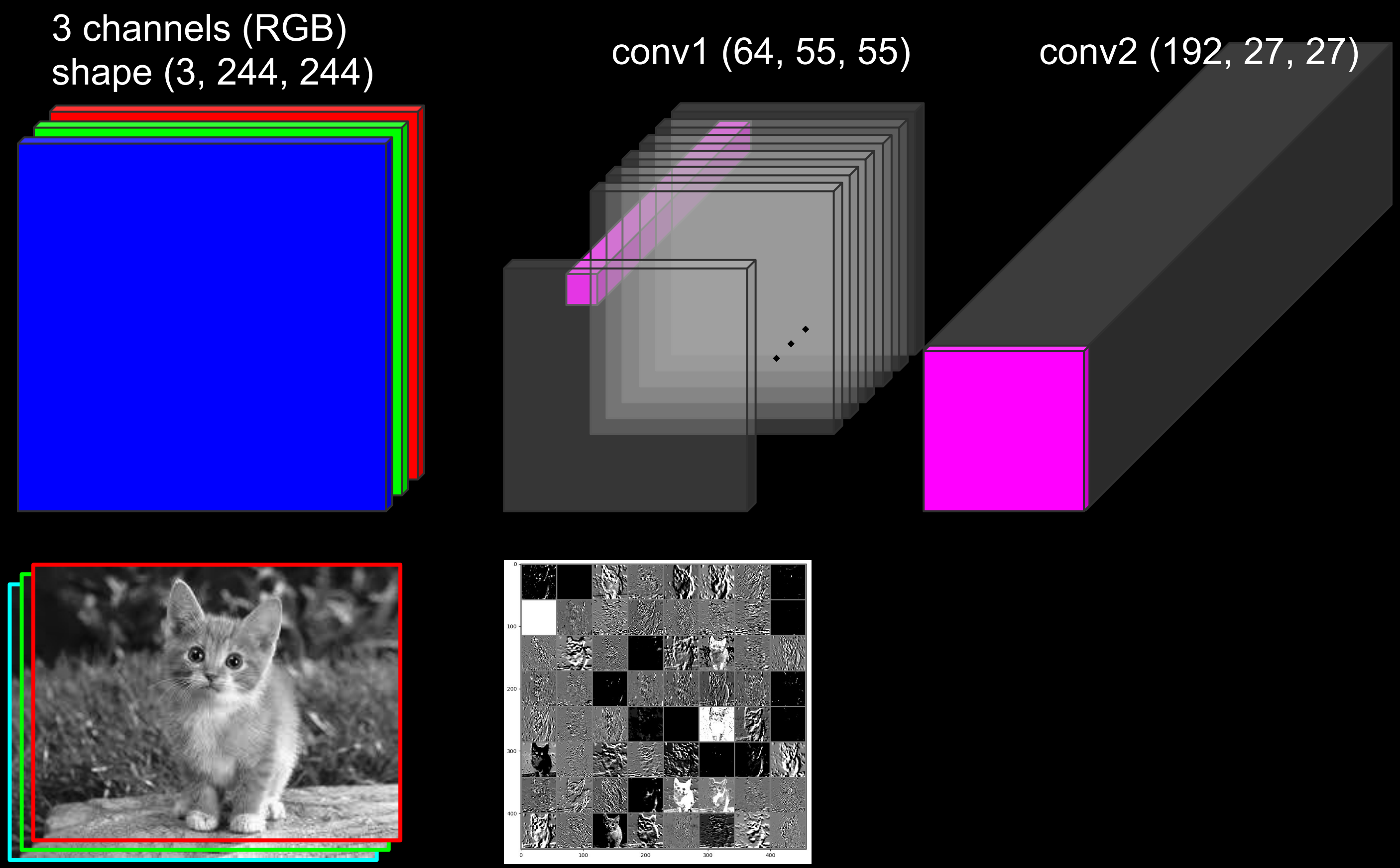
conv1 (64, 55, 55)

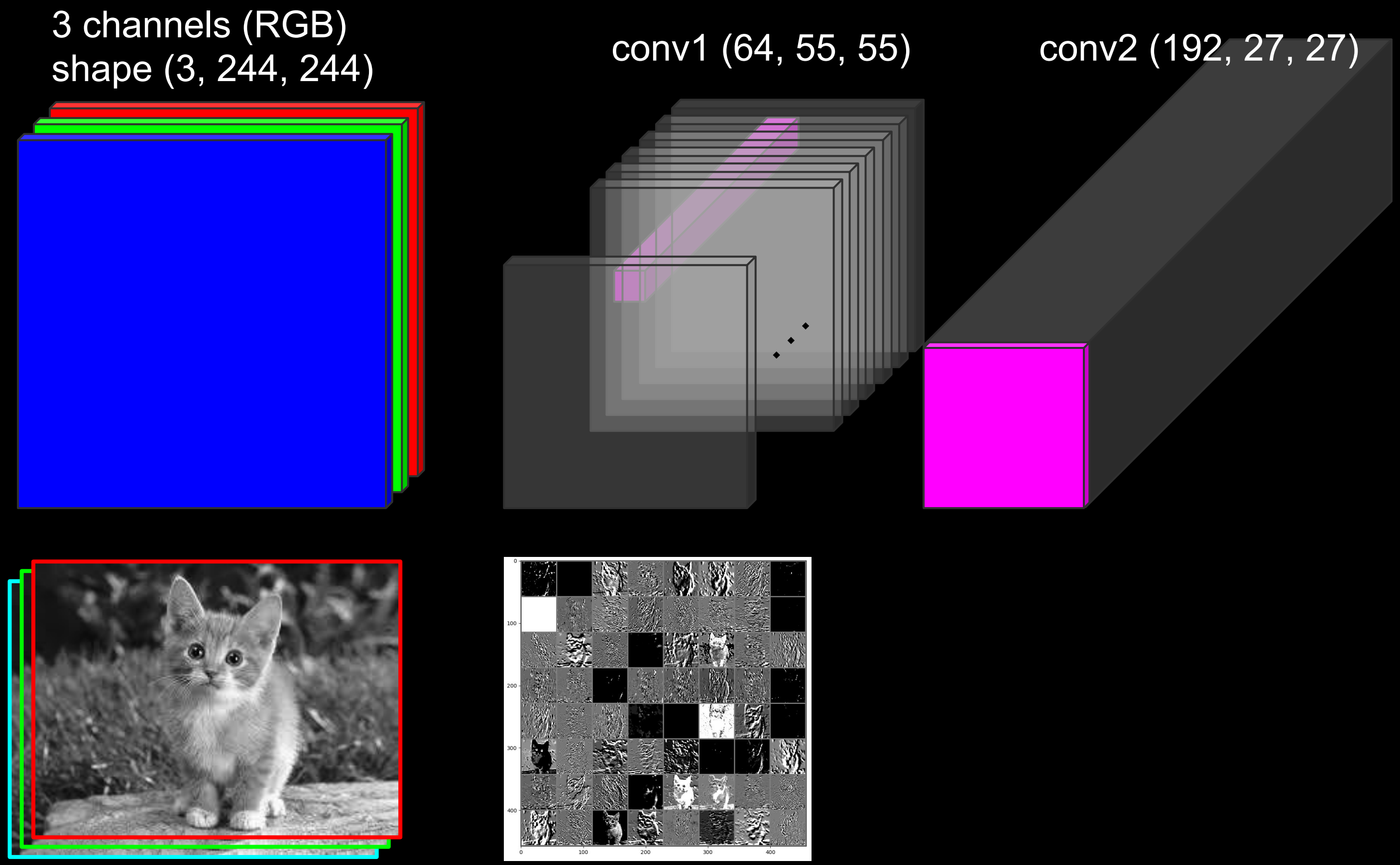


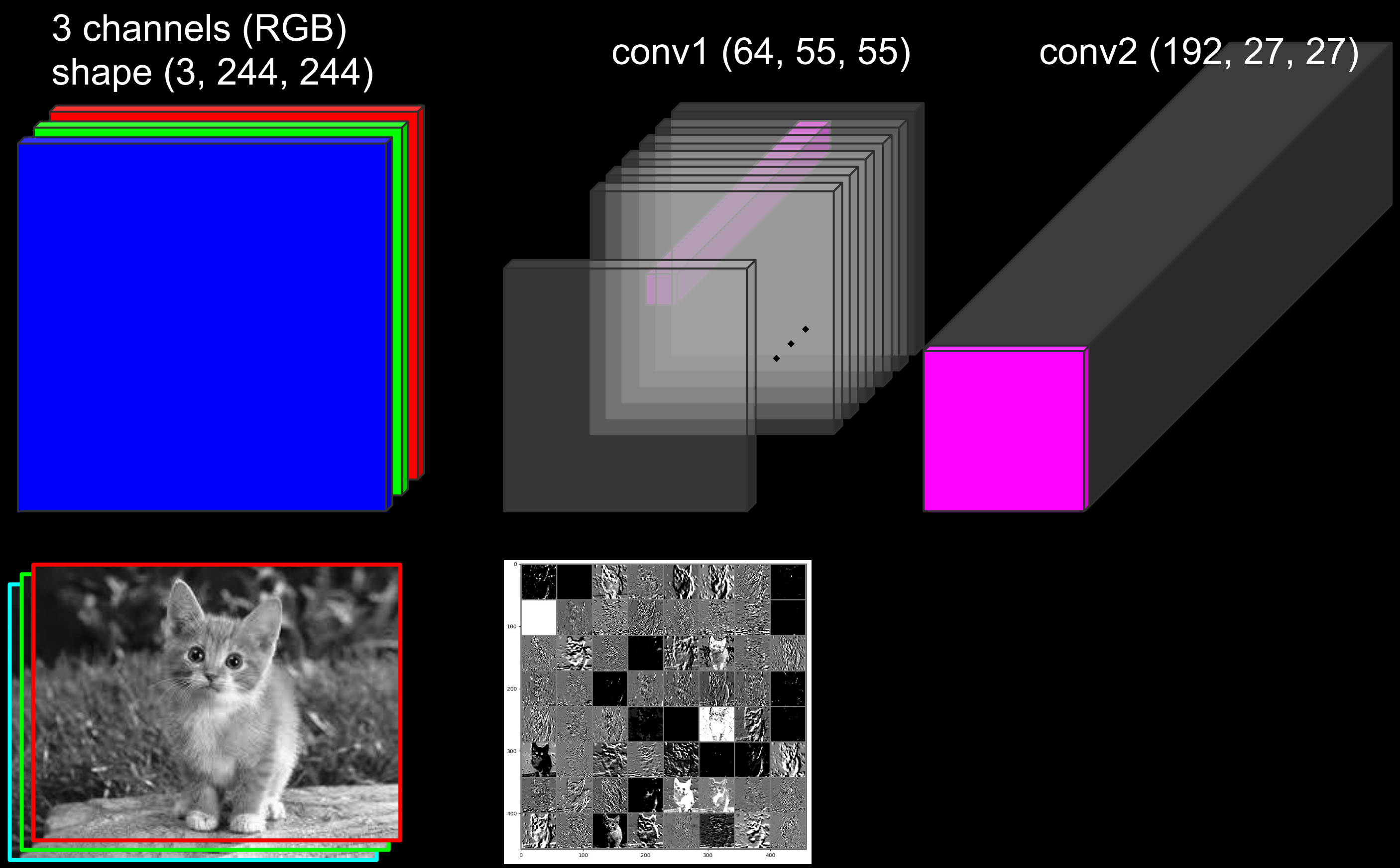


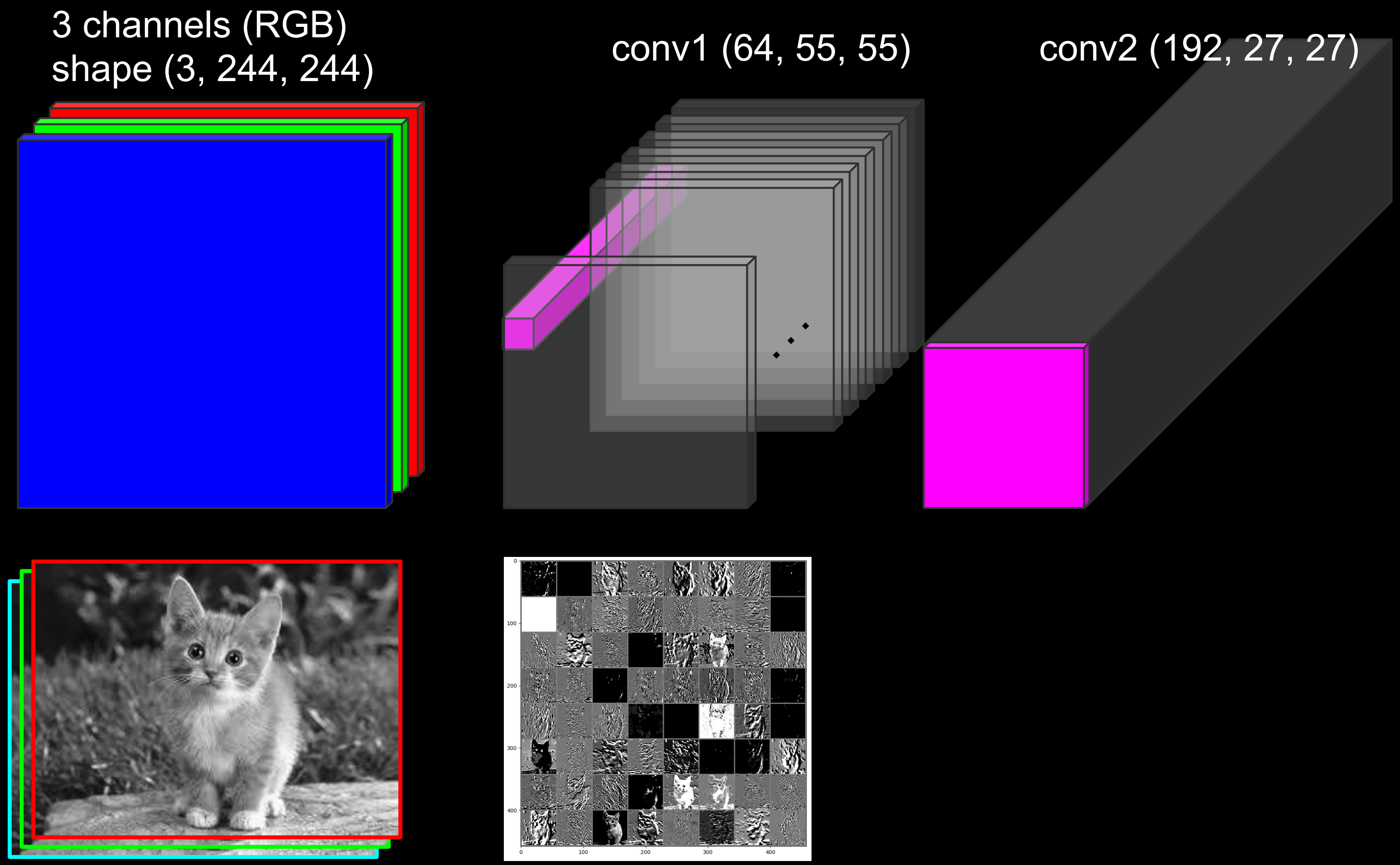

3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)

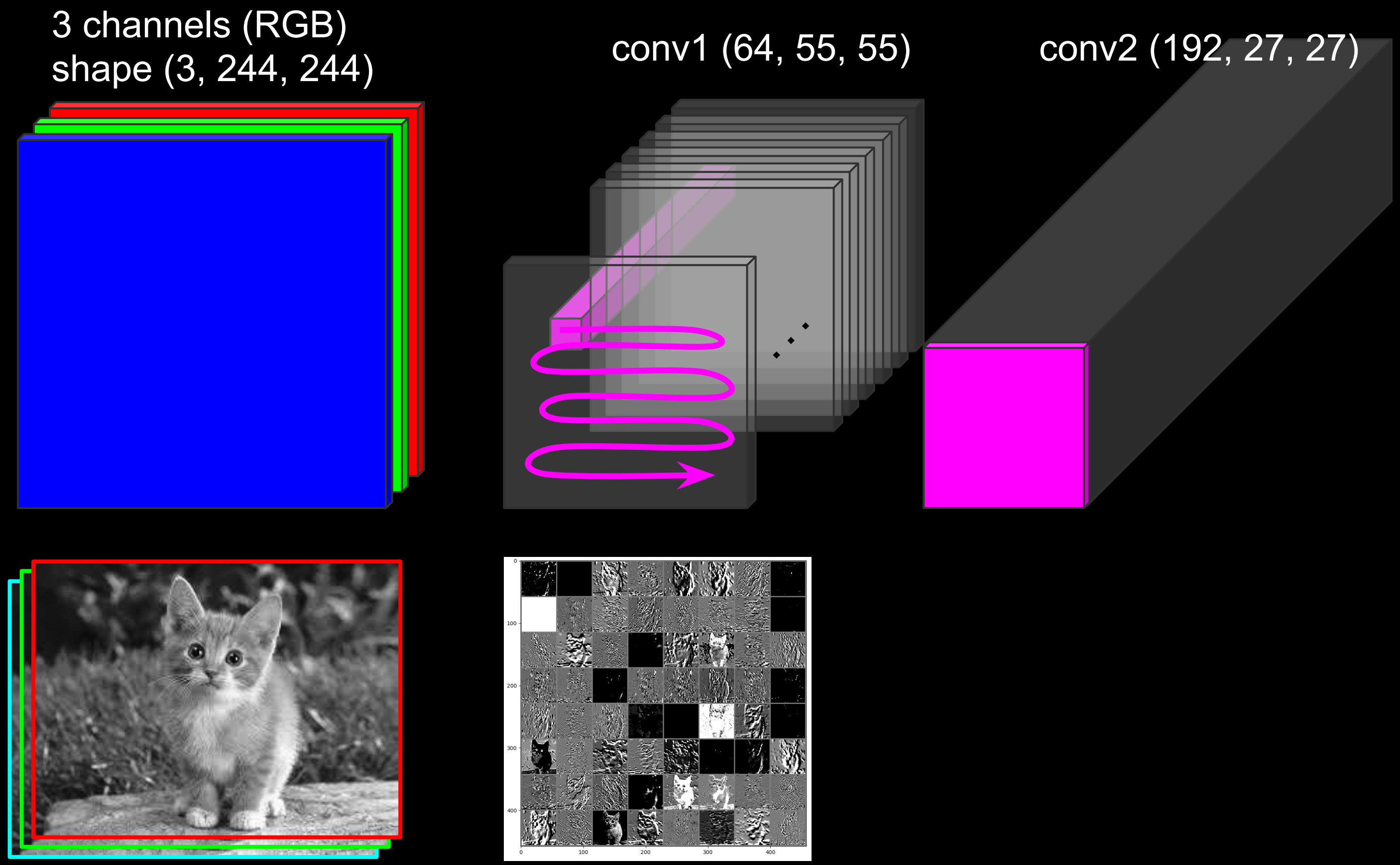


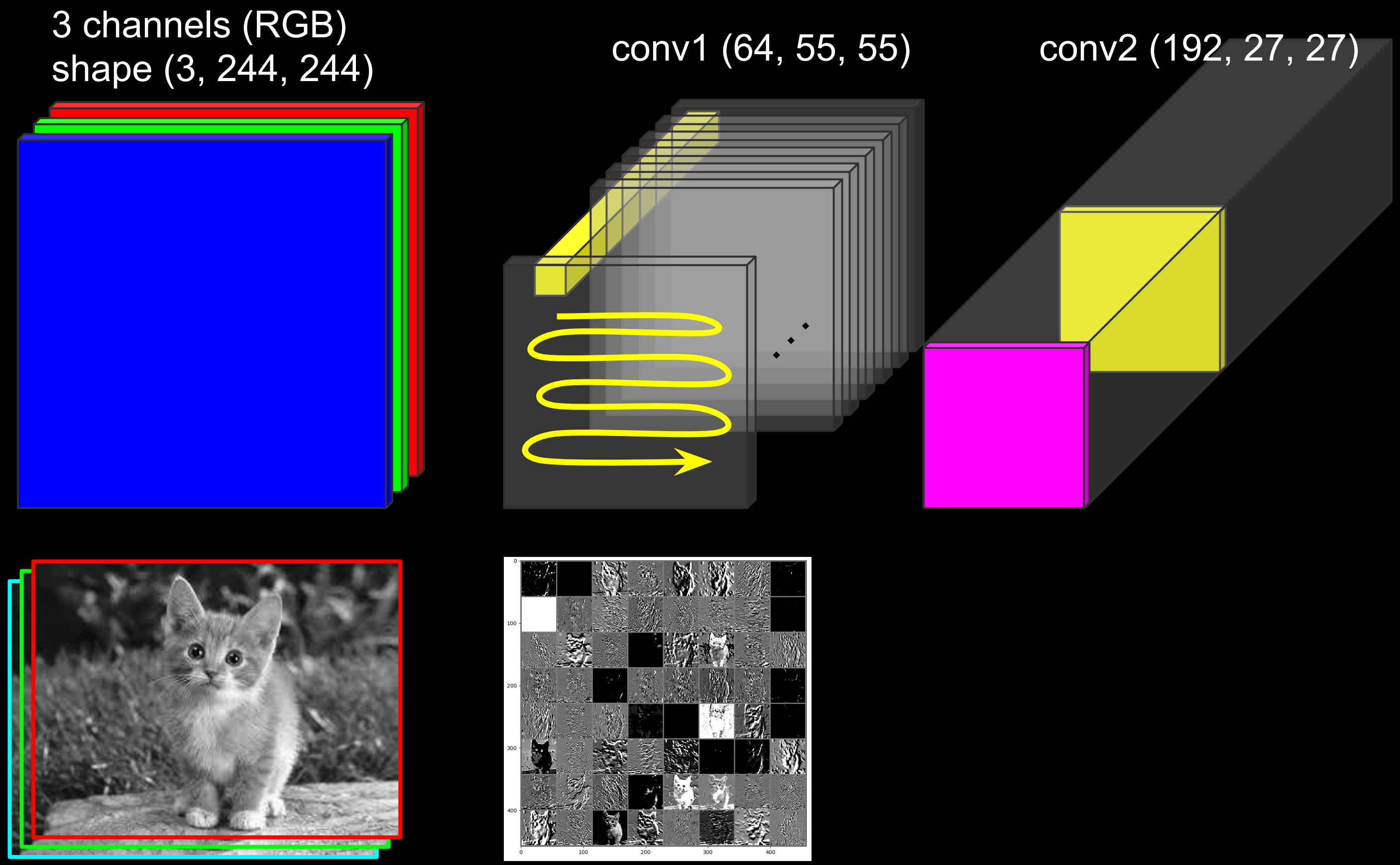

3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)

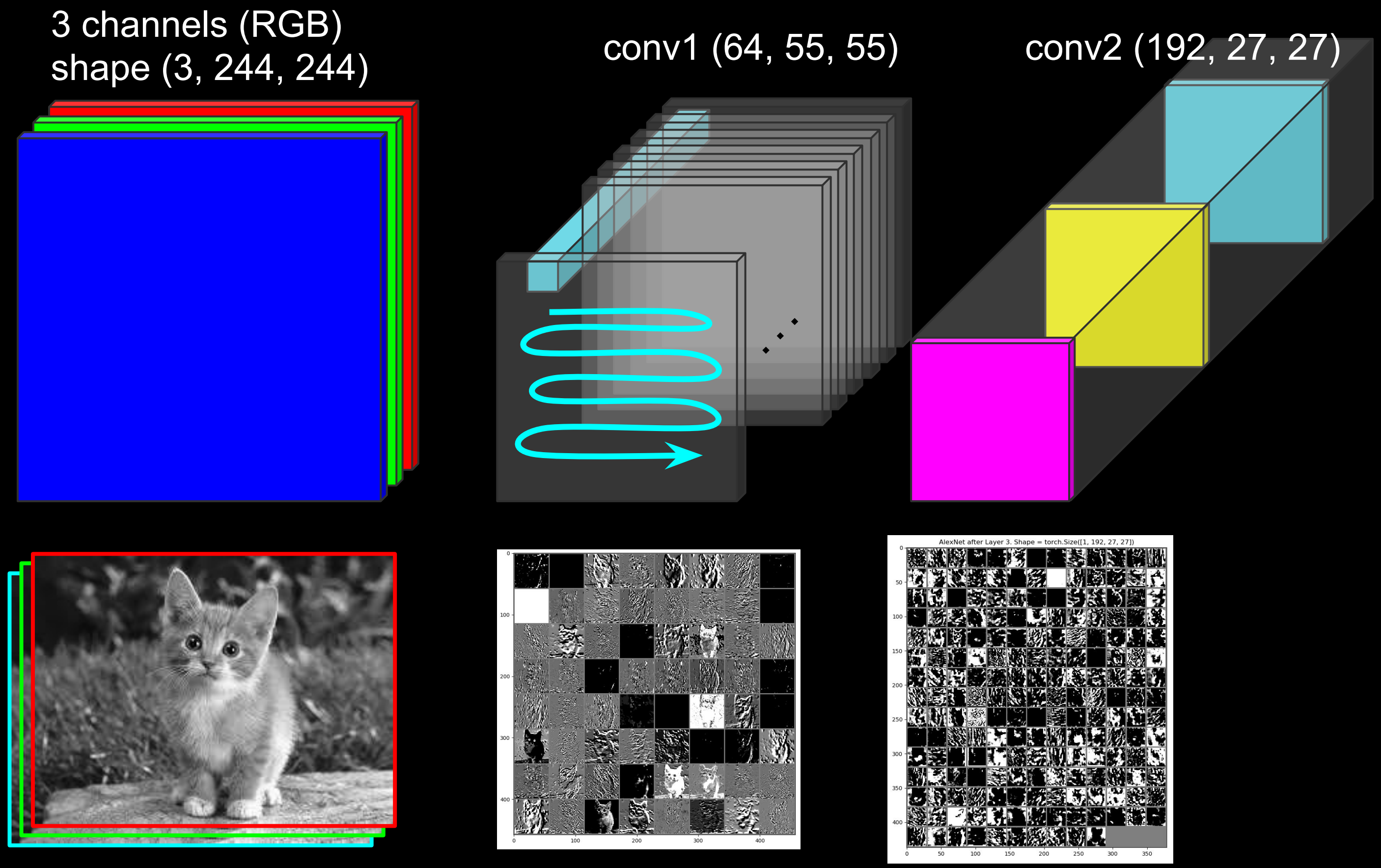

3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)


3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)


3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)


3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)


3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)


3 channels (RGB)
shape (3, 244, 244)
conv1 (64, 55, 55)
conv2 (192, 27, 27)
AlexNet after Layer 3. Shape = torch.Size([1, 192, 27, 27])

```python
alexnet = torchvision.models.alexnet(pretrained=True)
print(alexnet)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace)
    (3): Dropout(p=0.5)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

AlexNet

```python
resnext = torchvision.models.resnext101_32x8d(pretrained=True)
print(resnext)
```

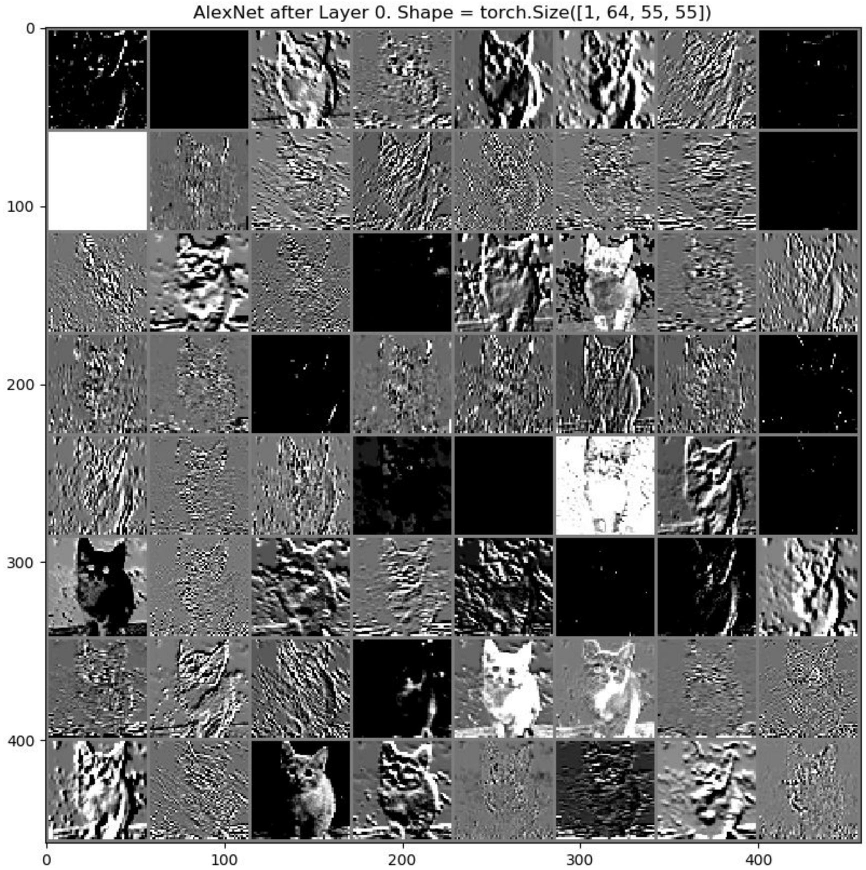
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=F
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=F
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      )
    )
```

```
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padd
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (relu): ReLU(inplace)
)
(2): Bottleneck(
  (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padd
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (relu): ReLU(inplace)
)
(3): Bottleneck(
  (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padd
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
  (relu): ReLU(inplace)
)
)
layer3): Sequential(
(0): Bottleneck(
  (conv1): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bia
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(2, 2), pa
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bi
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (relu): ReLU(inplace)
  (downsample): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  )
)
(1): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bi
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), pa
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bi
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (relu): ReLU(inplace)
)
(2): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bi
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), pa
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
```

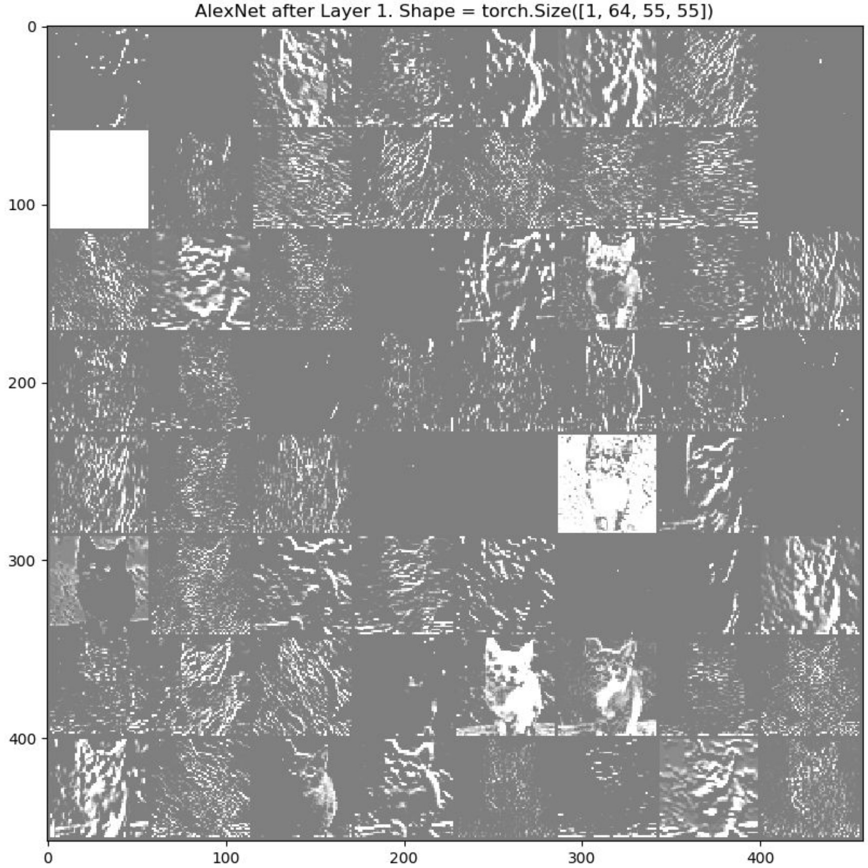
```
ine=True, track_running_stats=True)
le=(1, 1), bias=False)
ine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(7): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(8): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(9): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(10): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(11): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(12): Bottleneck(
  (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
  (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

ResNeXt

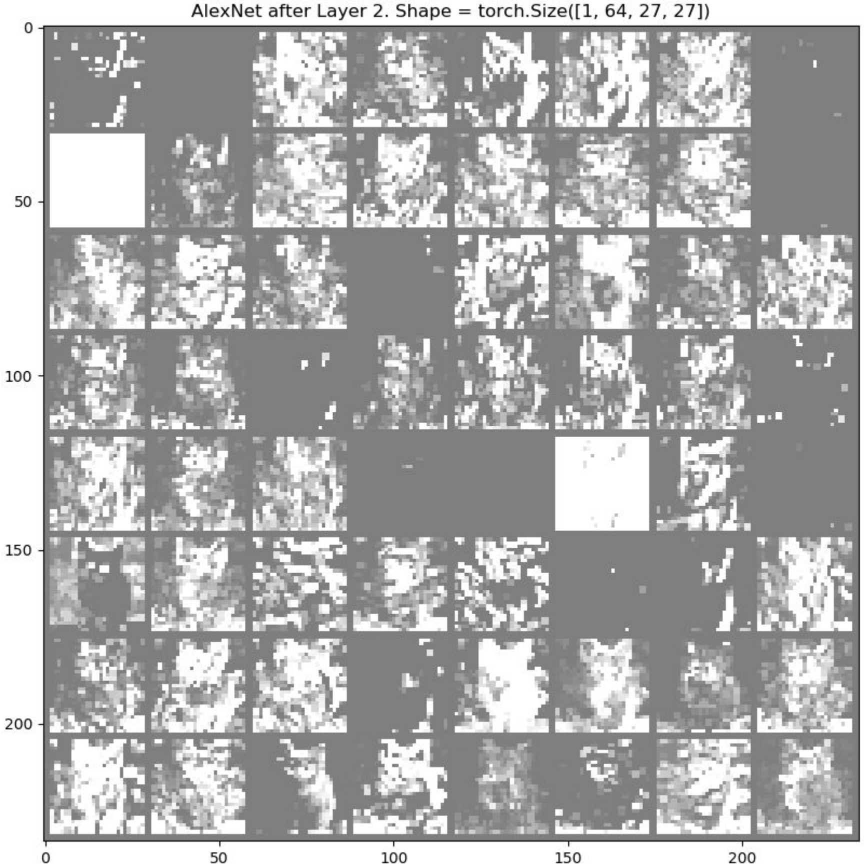
3 channels (RGB)
shape (3, 244, 244)



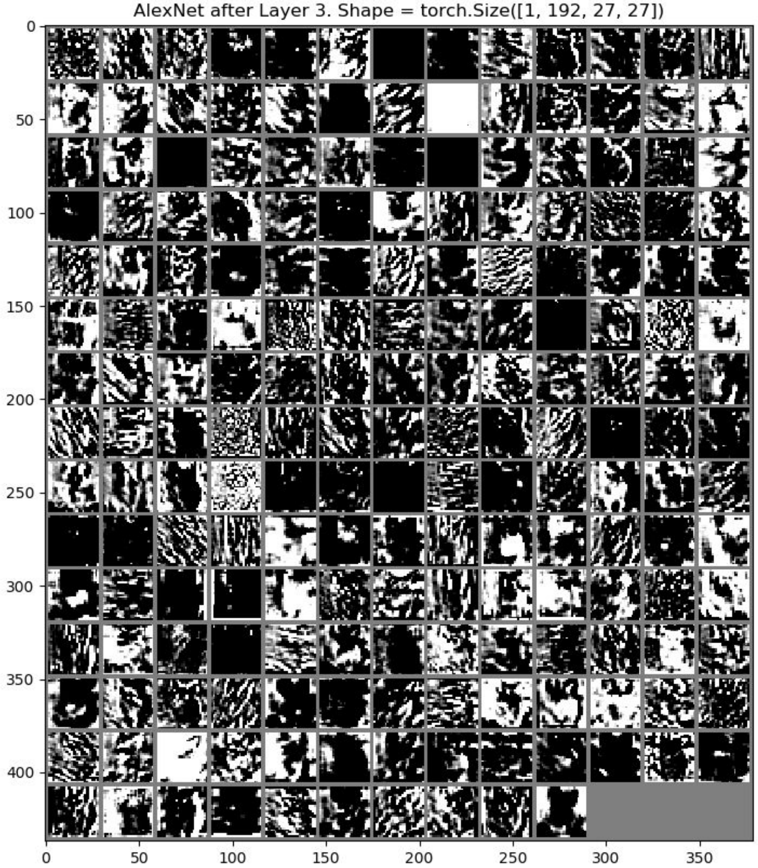

3 channels (RGB)
shape (3, 244, 244)


AlexNet after Layer 1. Shape = torch.Size([1, 64, 55, 55])
0
100
200
300
400
0
100
200
300
400

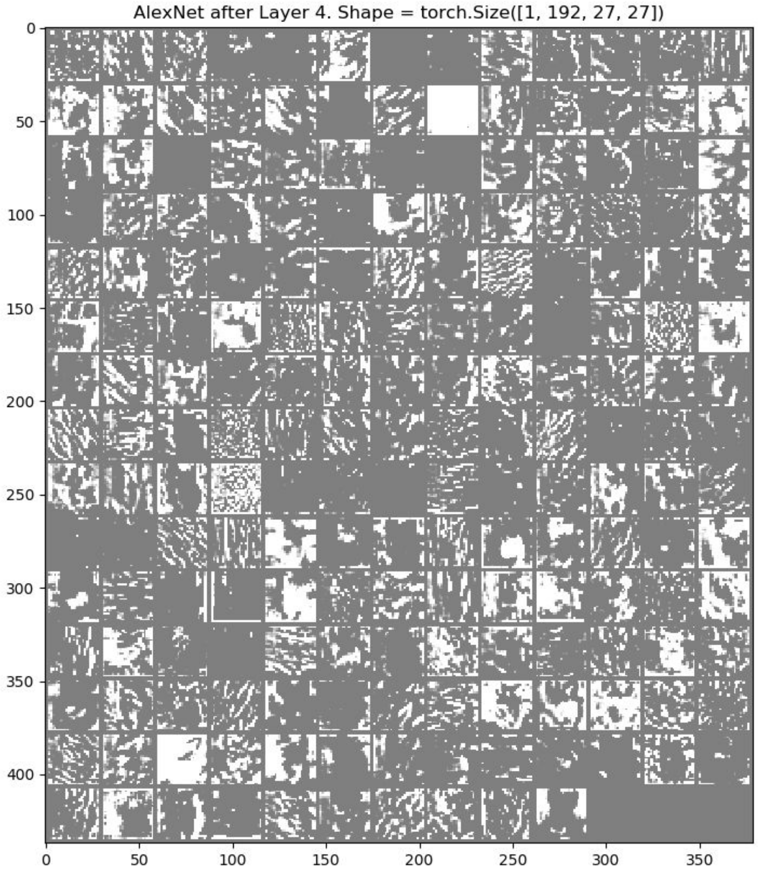
3 channels (RGB)
shape (3, 244, 244)



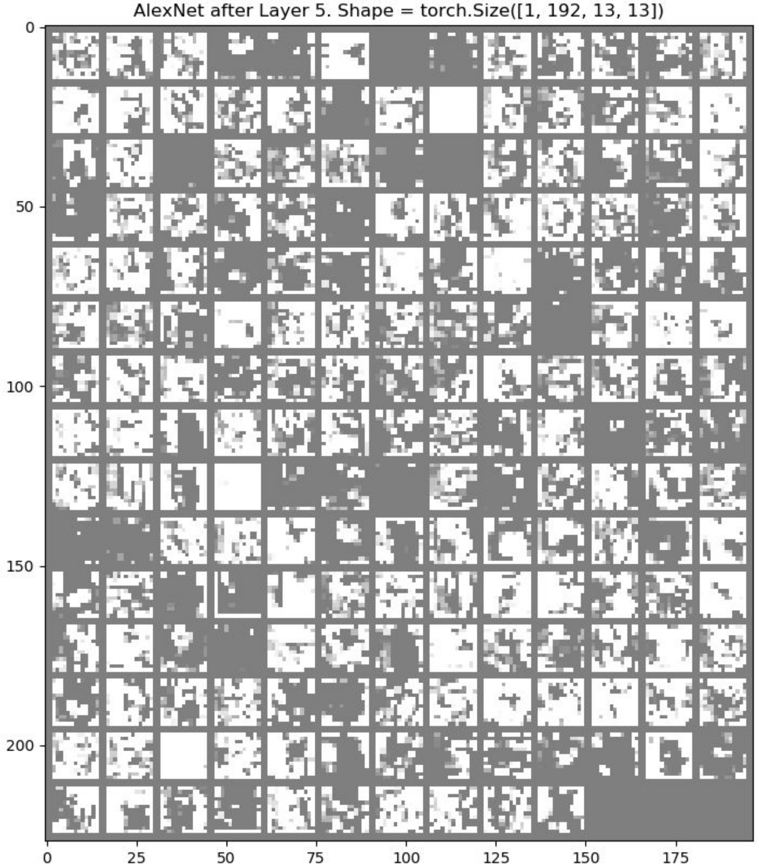
3 channels (RGB)
shape (3, 244, 244)



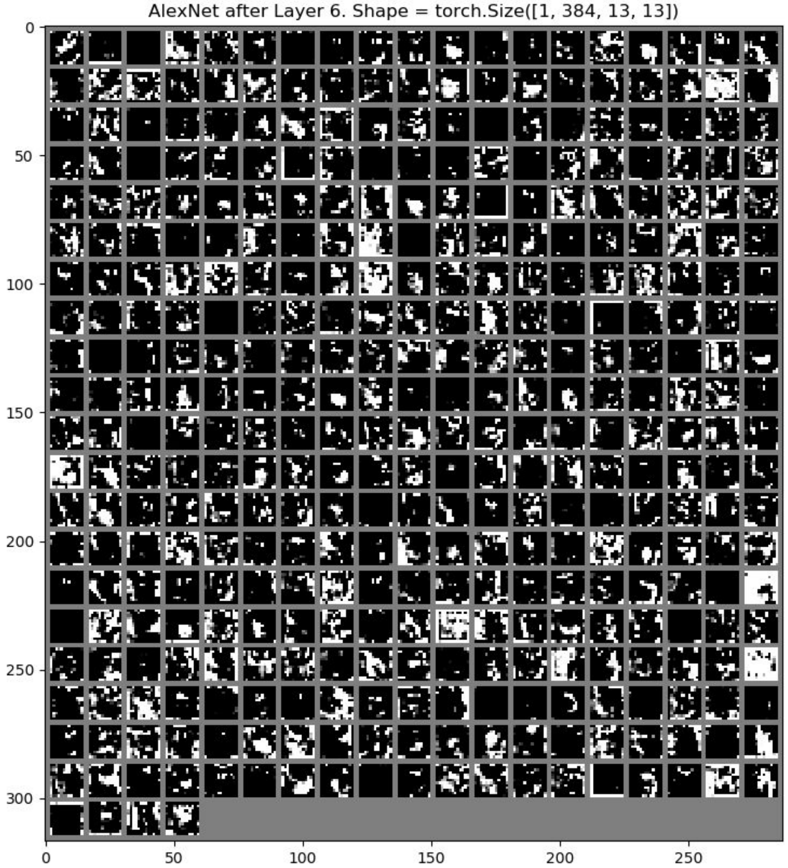
3 channels (RGB)
shape (3, 244, 244)

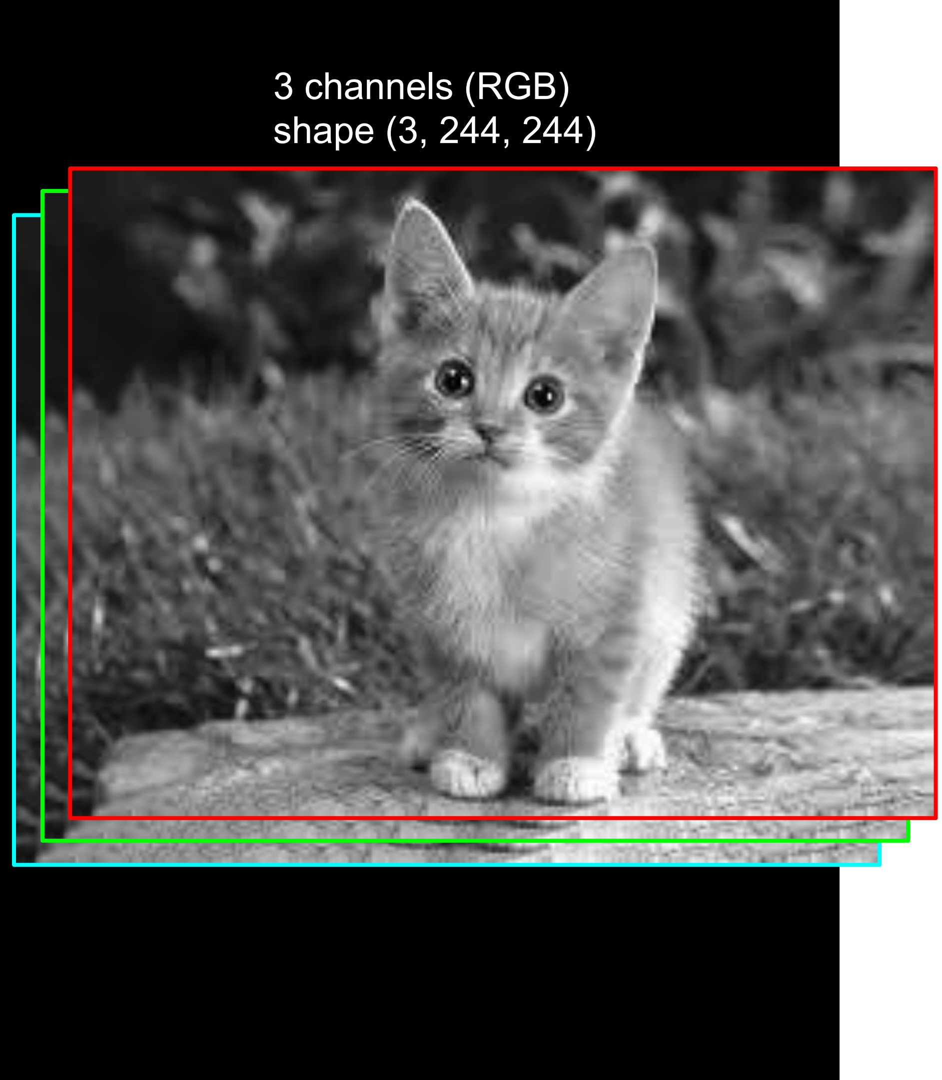


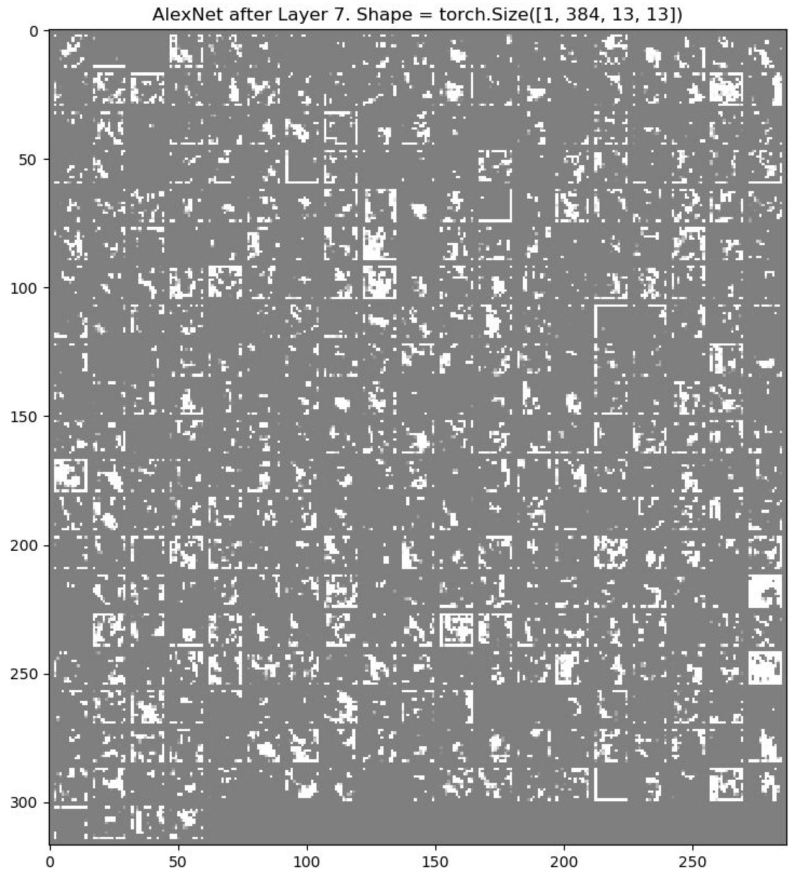
3 channels (RGB)
shape (3, 244, 244)



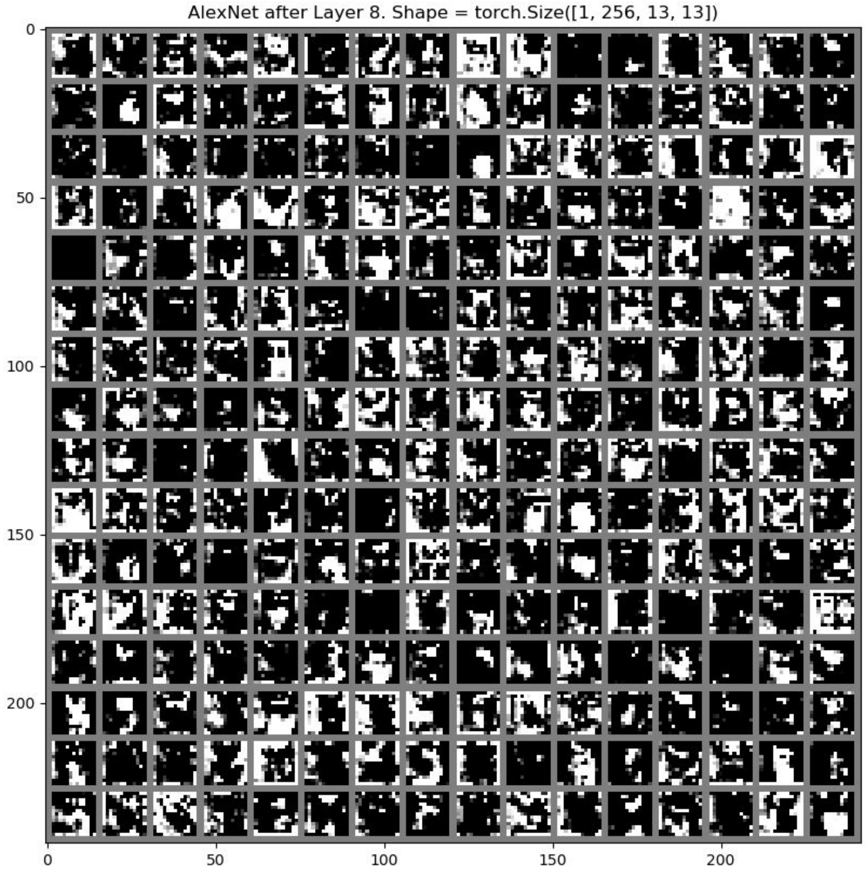

3 channels (RGB)
shape (3, 244, 244)


AlexNet after Layer 6. Shape = torch.Size([1, 384, 13, 13])
0
50
100
150
200
250
300
0
50
100
150
200
250


3 channels (RGB)
shape (3, 244, 244)

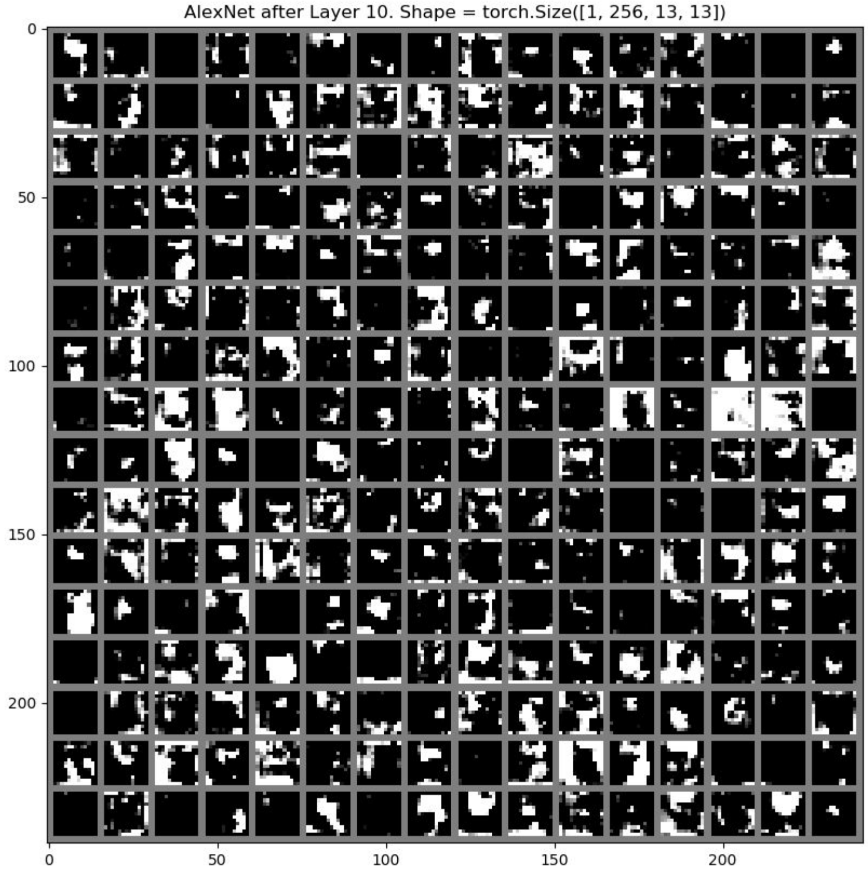

AlexNet after Layer 7. Shape = torch.Size([1, 384, 13, 13])
0
50
100
150
200
250
300
0
50
100
150
200
250

3 channels (RGB)
shape (3, 244, 244)

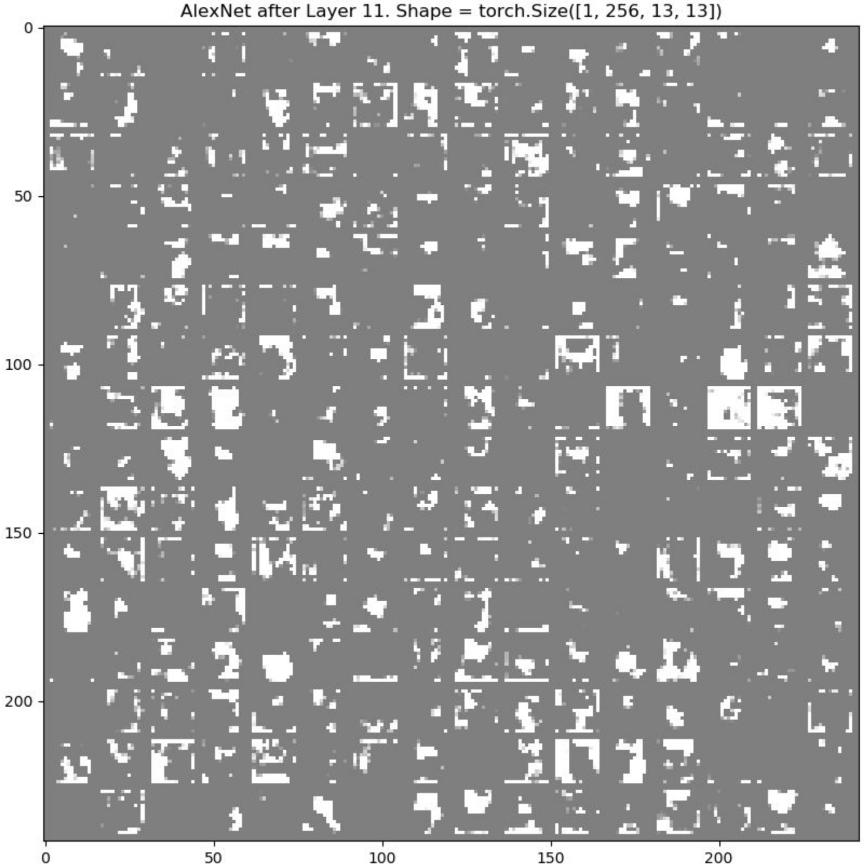


3 channels (RGB)
shape (3, 244, 244)

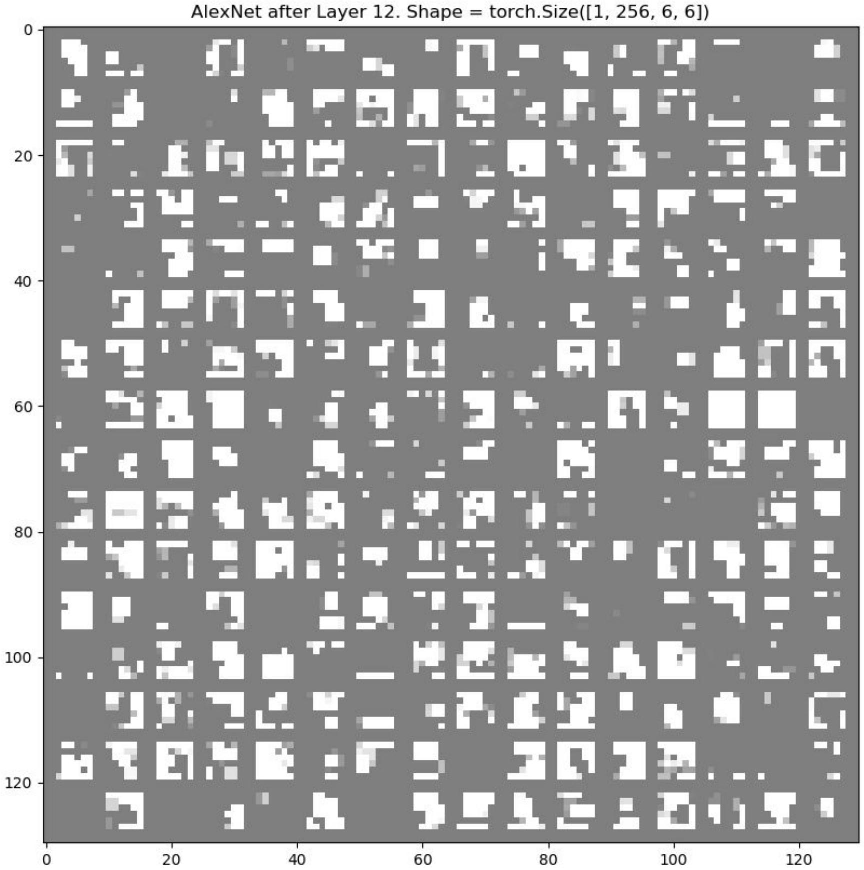
3 channels (RGB)
shape (3, 244, 244)

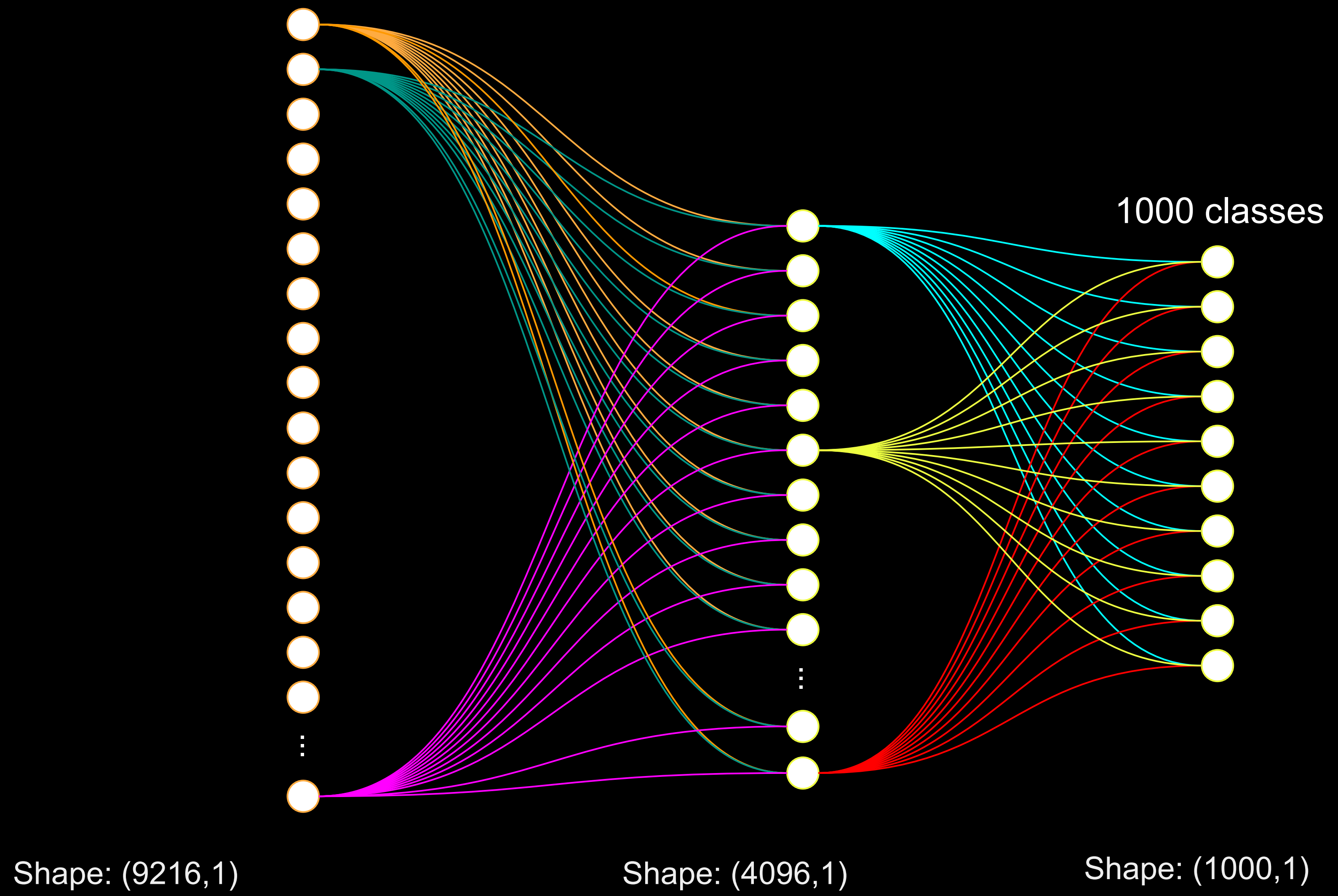


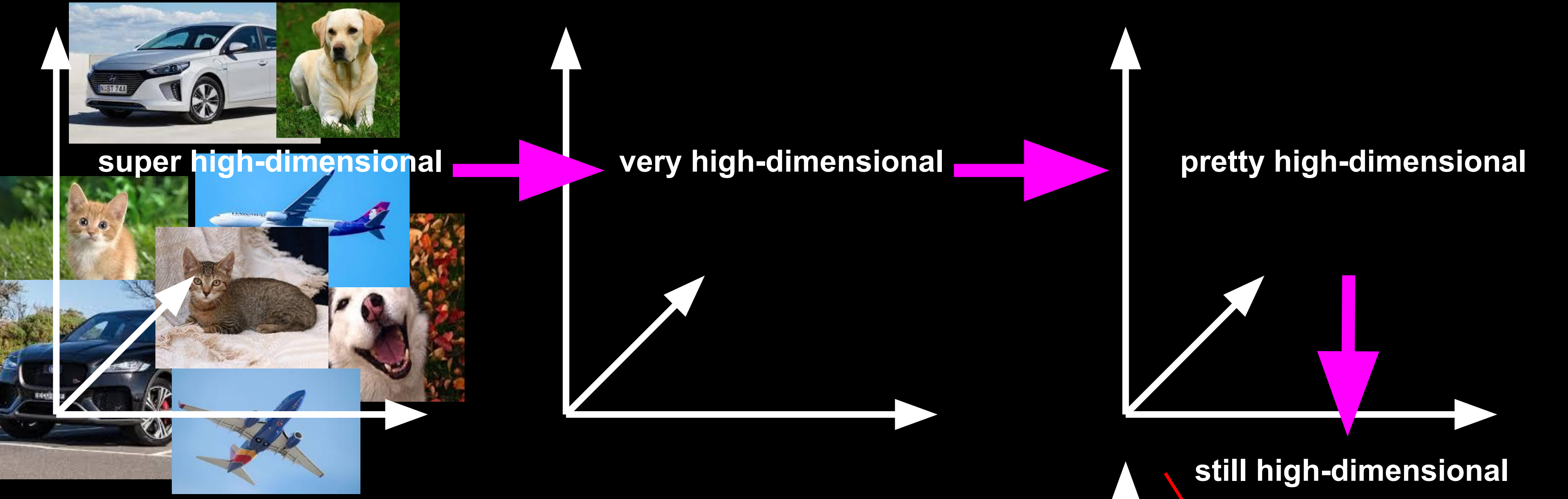
3 channels (RGB)
shape (3, 244, 244)

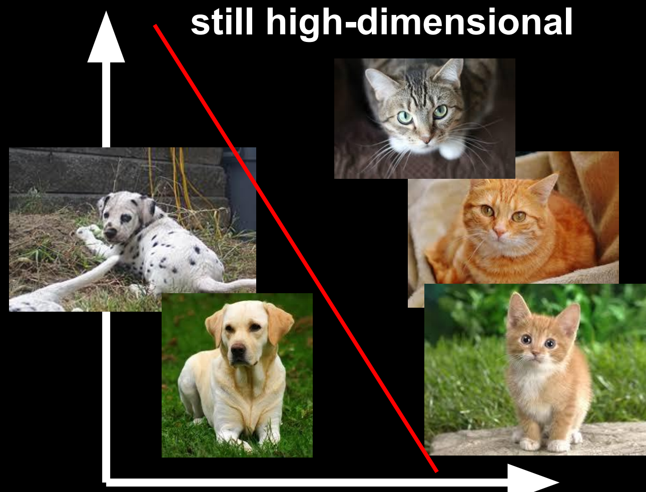


3 channels (RGB)
shape (3, 244, 244)



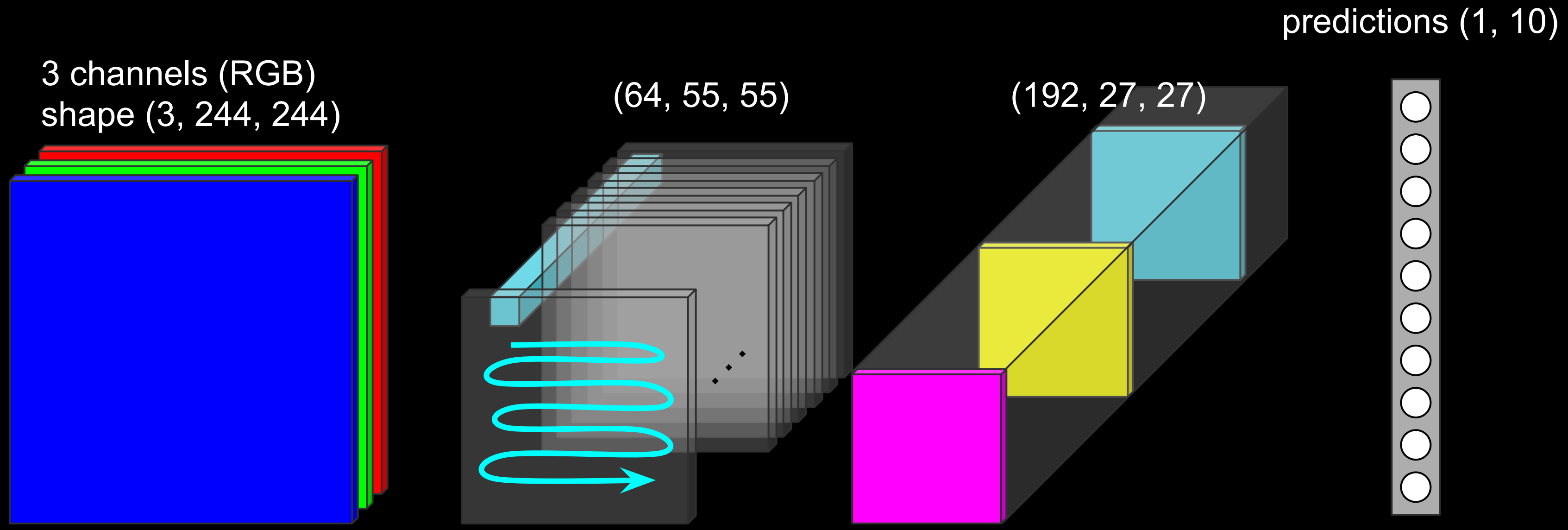

1000 classes
Shape: (9216,1)
Shape: (4096,1)
Shape: (1000,1)

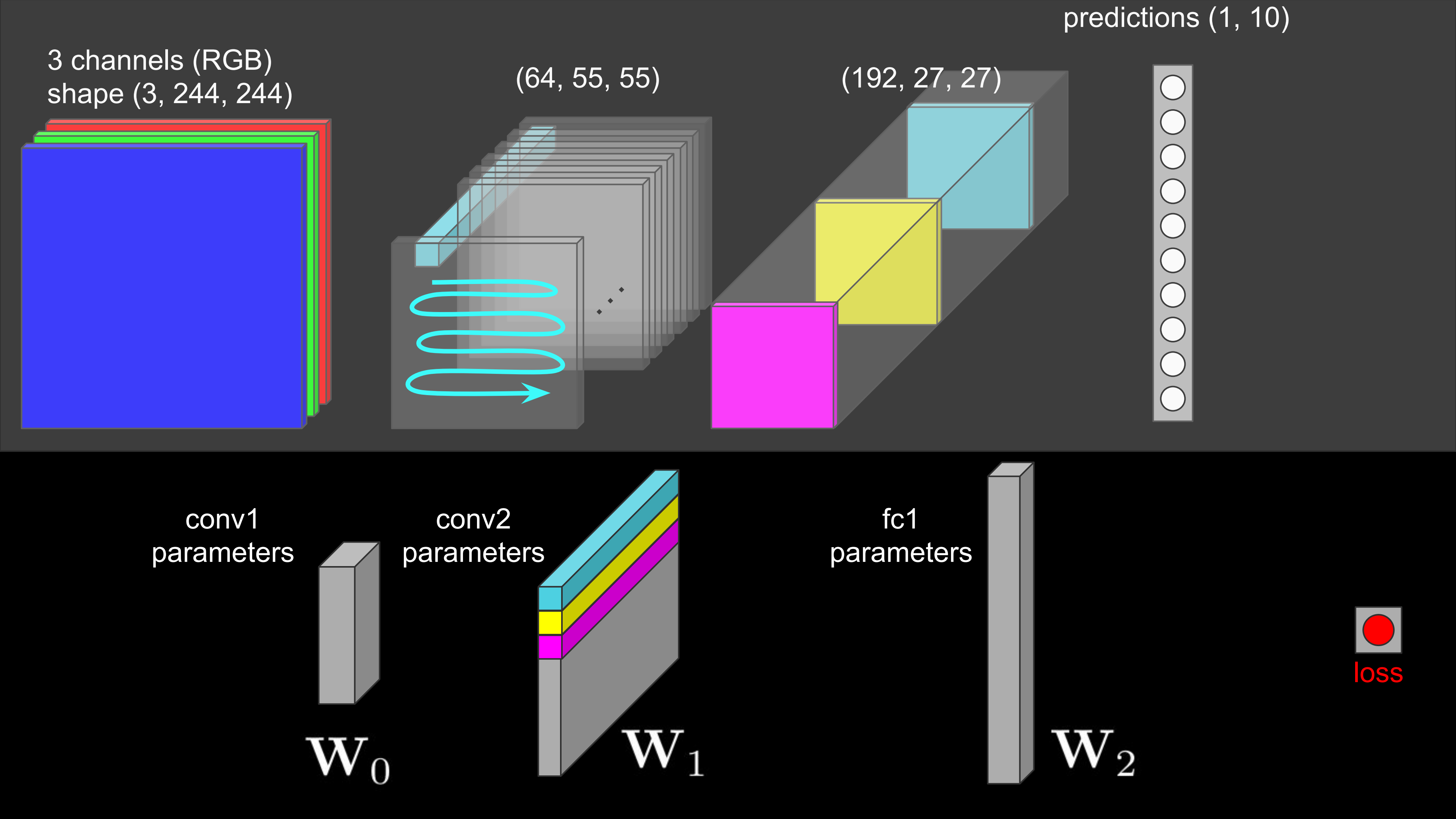




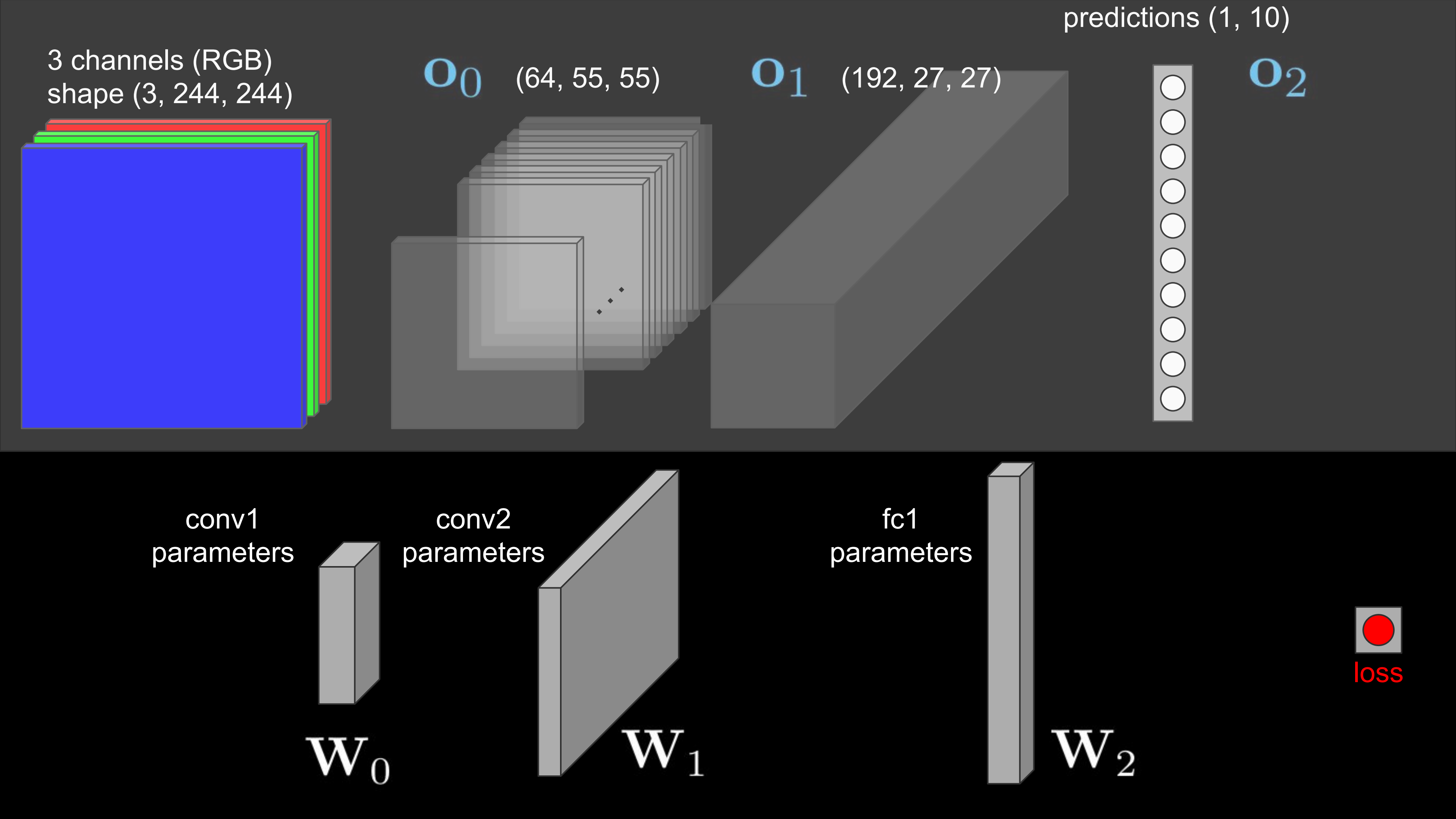
Nonlinear projections from one space into another.
Until classes are linearly separable.

Backpropagation

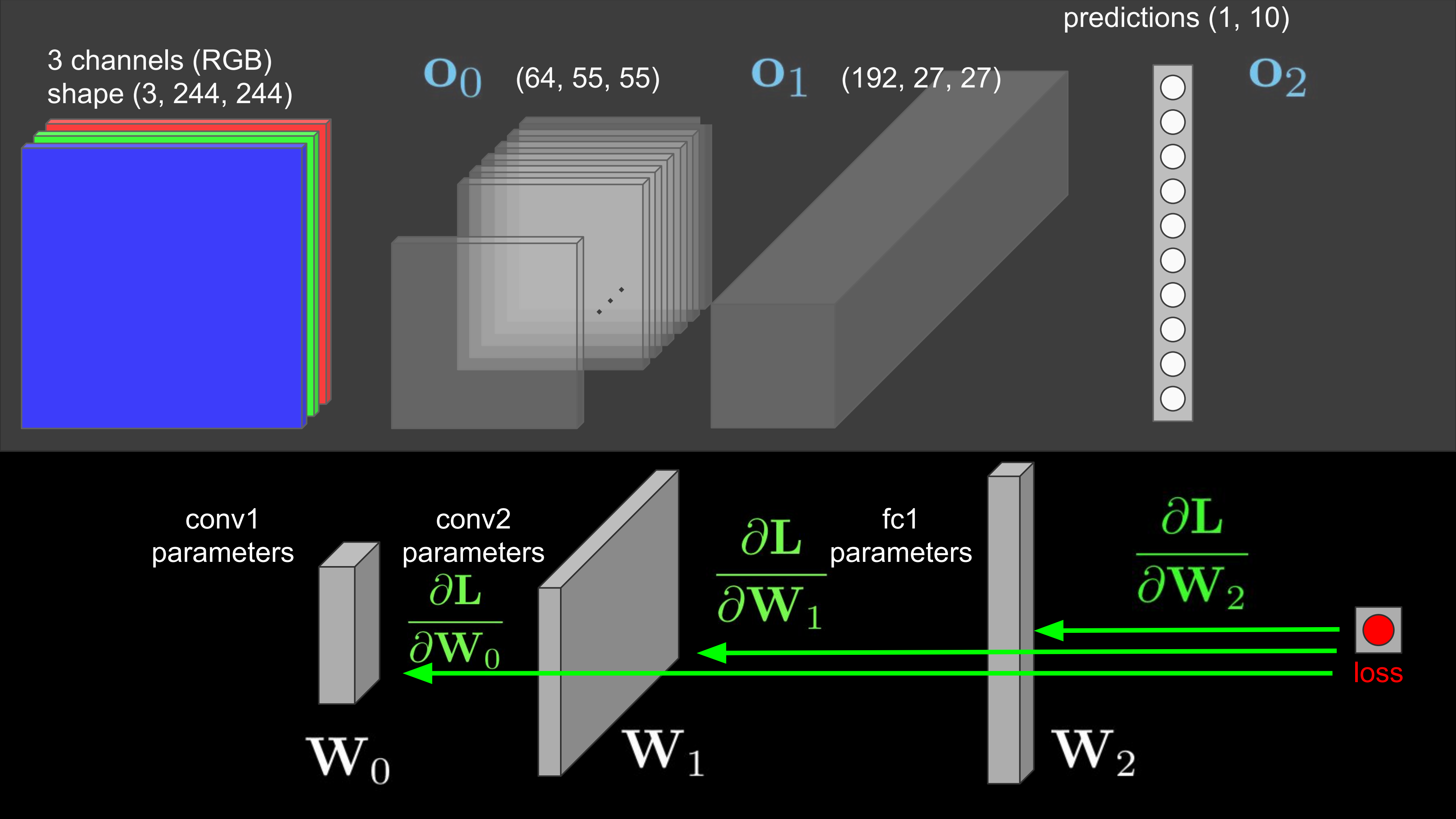

predictions (1, 10)
3 channels (RGB)
shape (3, 244, 244)
(64, 55, 55)
(192, 27, 27)



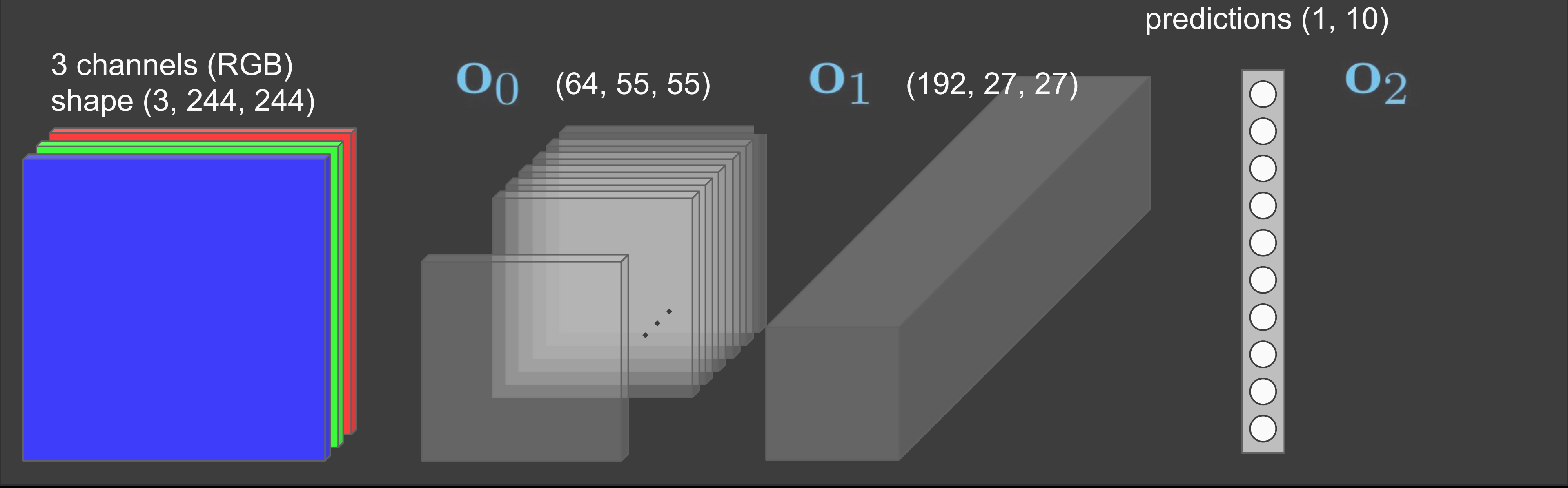
predictions (1, 10)
3 channels (RGB)
shape (3, 244, 244)
(64, 55, 55)
(192, 27, 27)
conv1
parameters
conv2
parameters
fc1
parameters
loss
W_0
W_1
W_2



3 channels (RGB)
shape (3, 244, 244)
$\mathbf{o}_0$
(64, 55, 55)
$\mathbf{o}_1$
(192, 27, 27)
predictions (1, 10)
$\mathbf{o}_2$
conv1
parameters
conv2
parameters
fc1
parameters
$\mathbf{W}_0$
$\mathbf{W}_1$
$\mathbf{W}_2$
loss



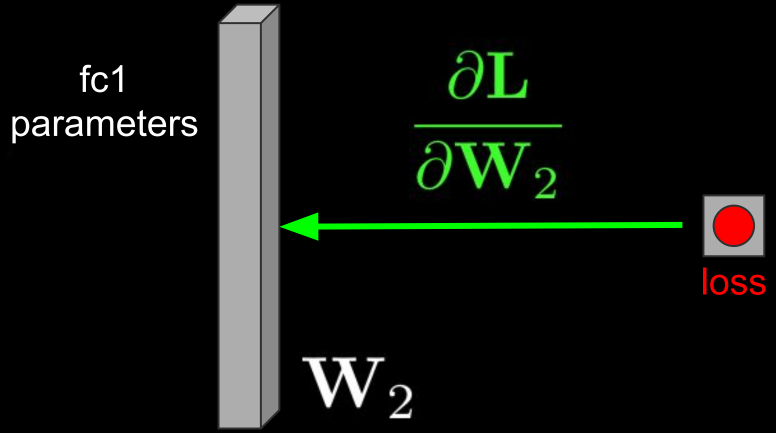
3 channels (RGB)
shape (3, 244, 244)
$\mathbf{o}_0$
(64, 55, 55)
$\mathbf{o}_1$
(192, 27, 27)
predictions (1, 10)
$\mathbf{o}_2$
conv1
parameters
conv2
parameters
$\frac{\partial L}{\partial \mathbf{W}_0}$
$\frac{\partial L}{\partial \mathbf{W}_1}$
fc1
parameters
$\frac{\partial L}{\partial \mathbf{W}_2}$
loss
$\mathbf{W}_0$
$\mathbf{W}_1$
$\mathbf{W}_2$

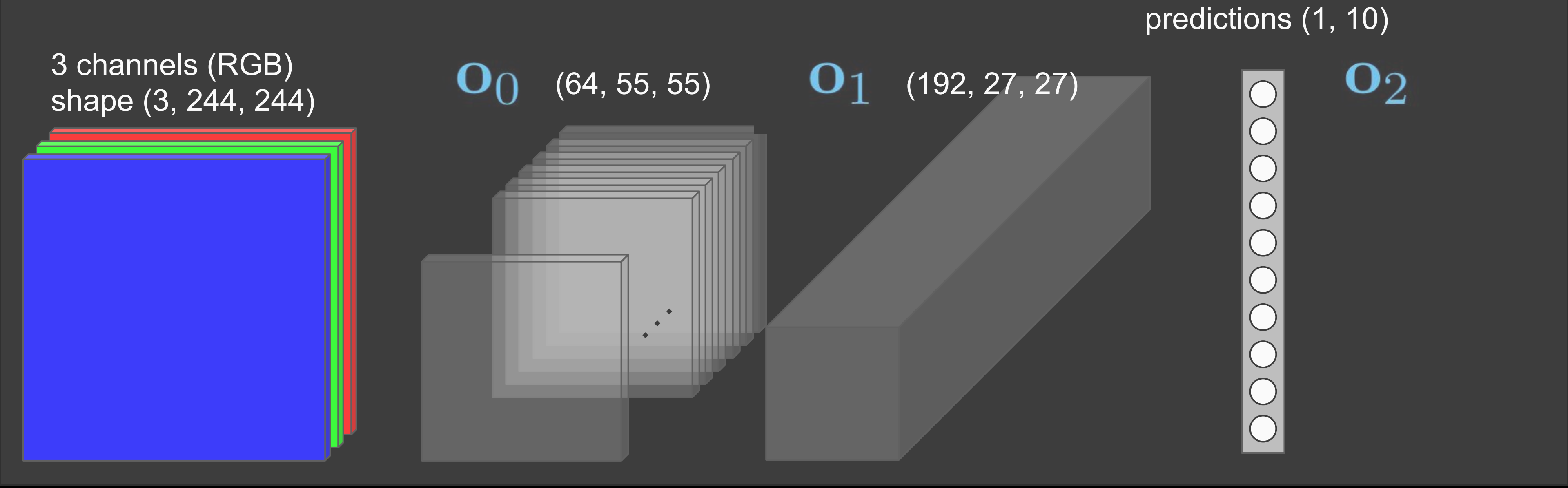


$$\mathbf{L} = f(\mathbf{o}_2)$$

$$\mathbf{o}_2 = g(\mathbf{o}_1, \mathbf{W}_2)$$

$$\mathbf{L} = f(g(\mathbf{o}_1, \mathbf{W}_2))$$

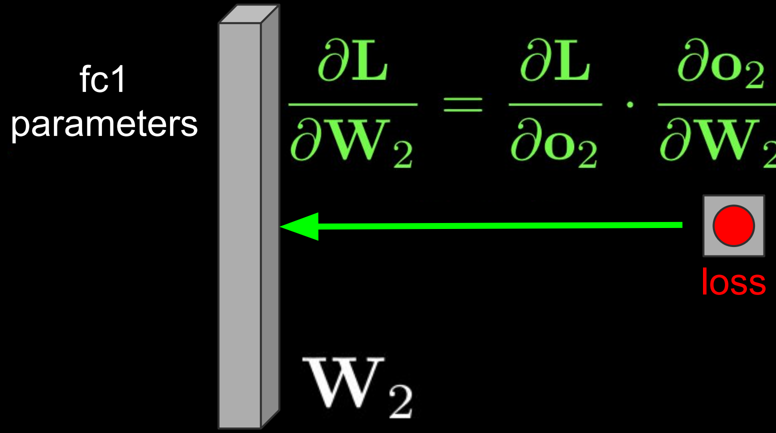




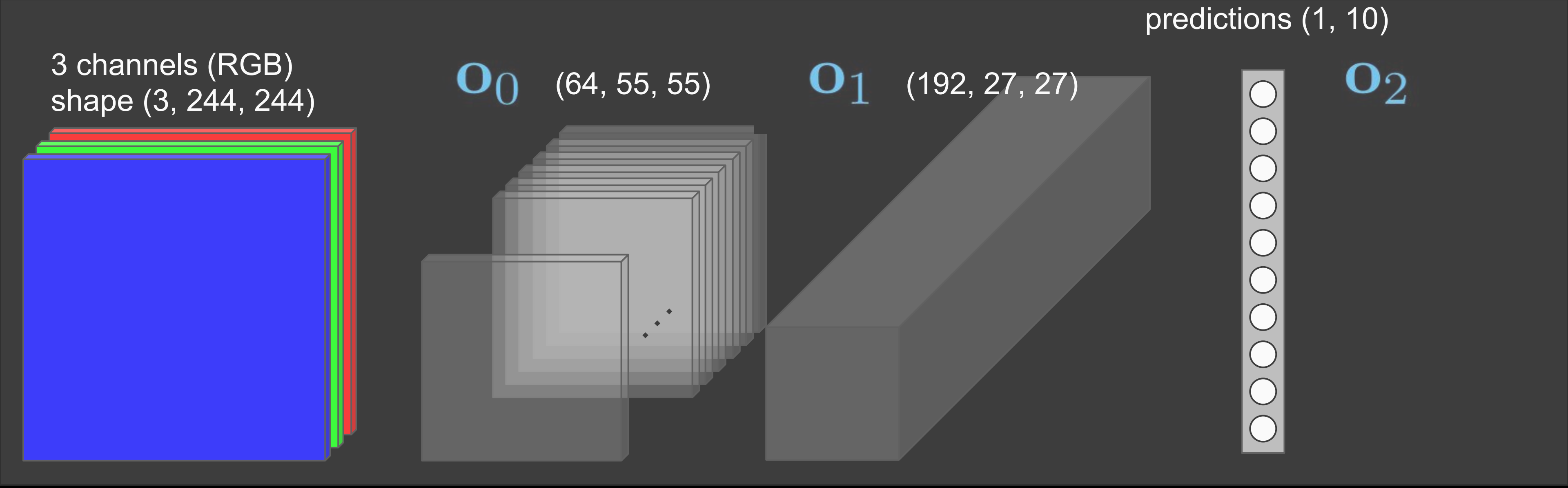
$$\mathbf{L} = f(\mathbf{o}_2)$$

$$\mathbf{o}_2 = g(\mathbf{o}_1, \mathbf{W}_2)$$

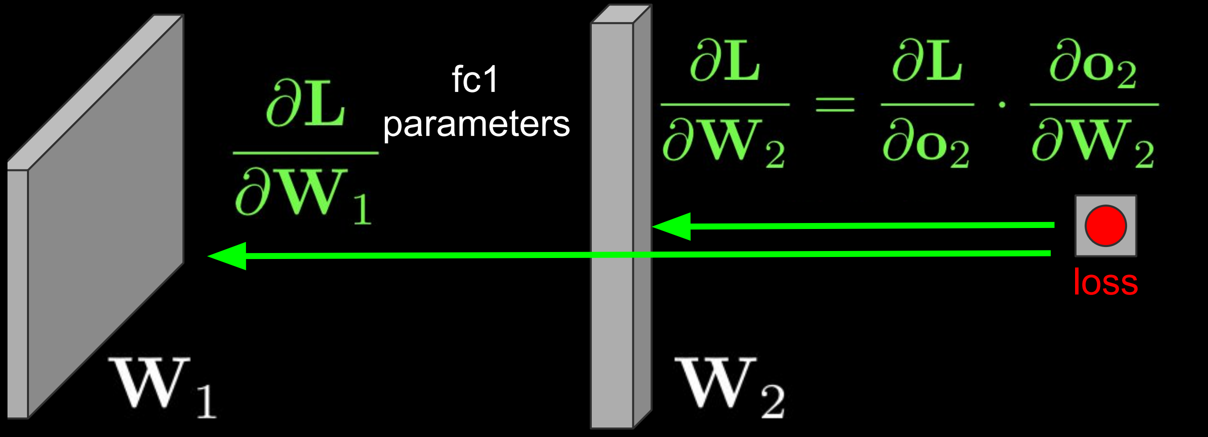
$$\mathbf{L} = f(g(\mathbf{o}_1, \mathbf{W}_2))$$

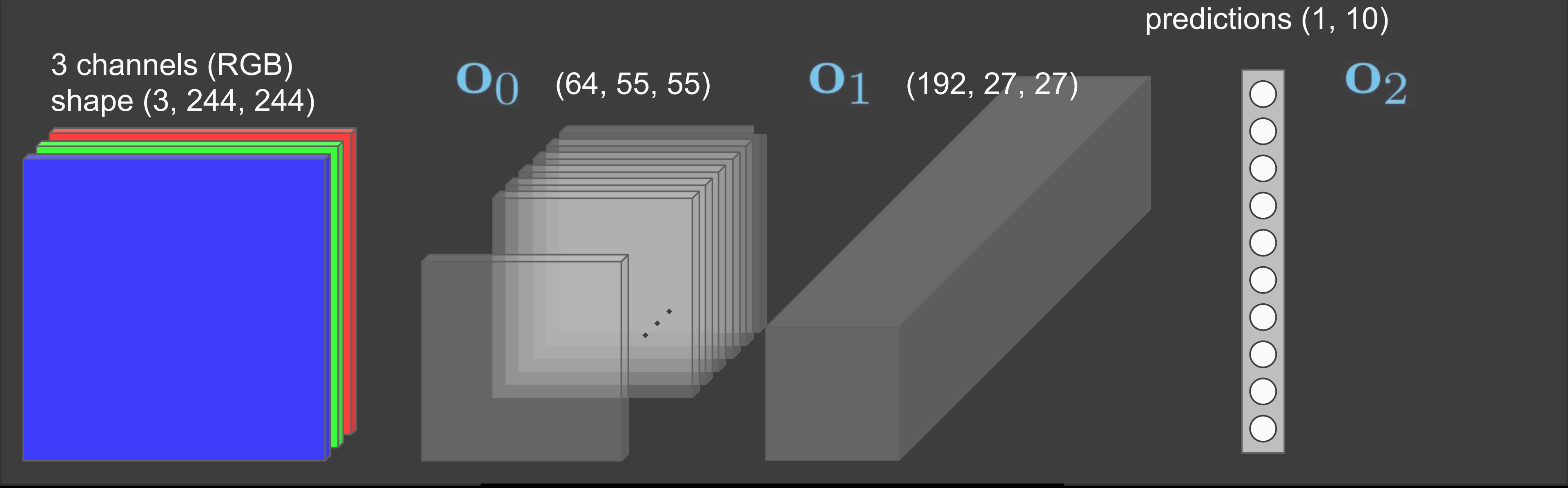


$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathbf{L}}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{W}_2}$$



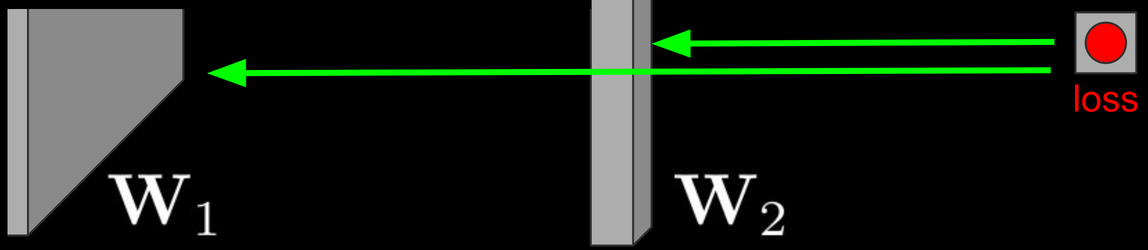
$$
\begin{aligned}
\mathbf{L} &= f(\mathbf{o}_2) \\
\mathbf{o}_2 &= g(\mathbf{o}_1, \mathbf{W}_2) \\
\mathbf{L} &= f(g(\mathbf{o}_1, \mathbf{W}_2)) \\
\mathbf{o}_1 &= h(\mathbf{o}_0, \mathbf{W}_1) \\
\mathbf{L} &= f(g(h(\mathbf{o}_0, \mathbf{W}_1), \mathbf{W}_2))
\end{aligned}
$$

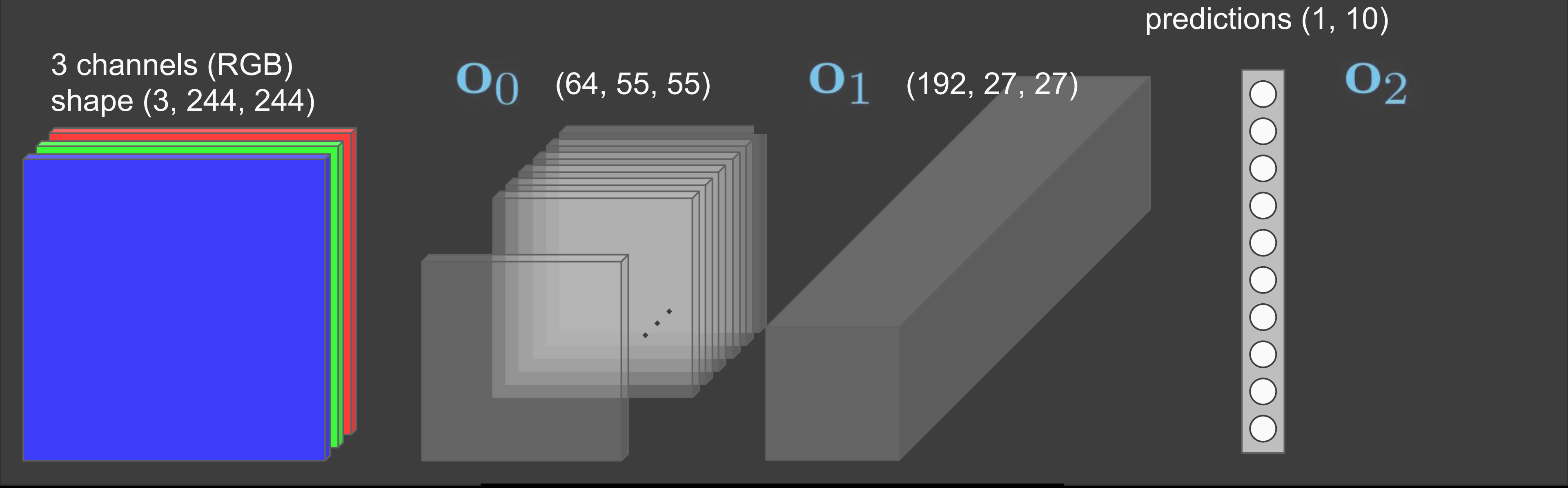




$$
\begin{aligned}
\mathbf{L} &= f(\mathbf{o}_2) \\
\mathbf{o}_2 &= g(\mathbf{o}_1, \mathbf{W}_2) \\
\mathbf{L} &= f(g(\mathbf{o}_1, \mathbf{W}_2)) \\
\mathbf{o}_1 &= h(\mathbf{o}_0, \mathbf{W}_1) \\
\mathbf{L} &= f(g(h(\mathbf{o}_0, \mathbf{W}_1), \mathbf{W}_2))
\end{aligned}
$$

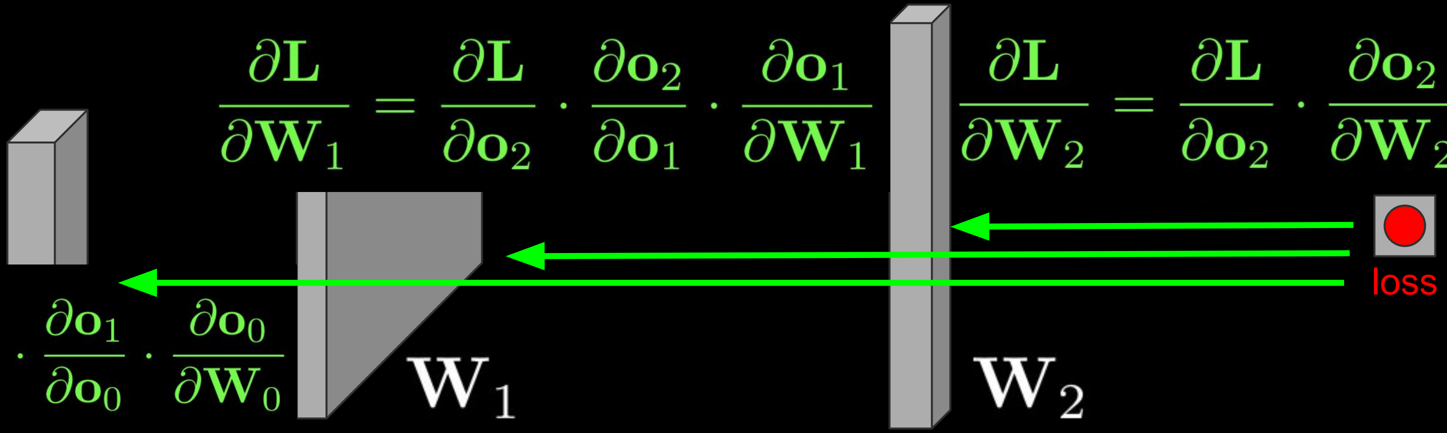
$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathbf{L}}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{o}_1} \cdot \frac{\partial \mathbf{o}_1}{\partial \mathbf{W}_1} \qquad \frac{\partial \mathbf{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathbf{L}}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{W}_2}$$



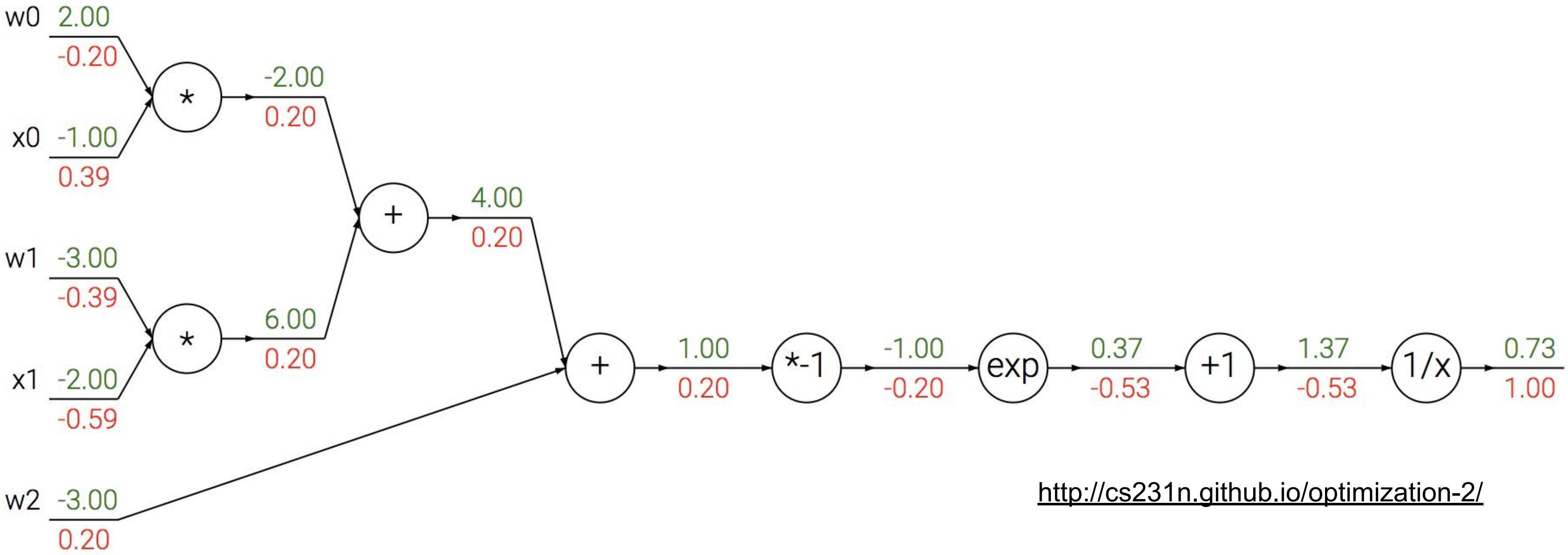


$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{o}_1} \cdot \frac{\partial \mathbf{o}_1}{\partial \mathbf{W}_1}$$

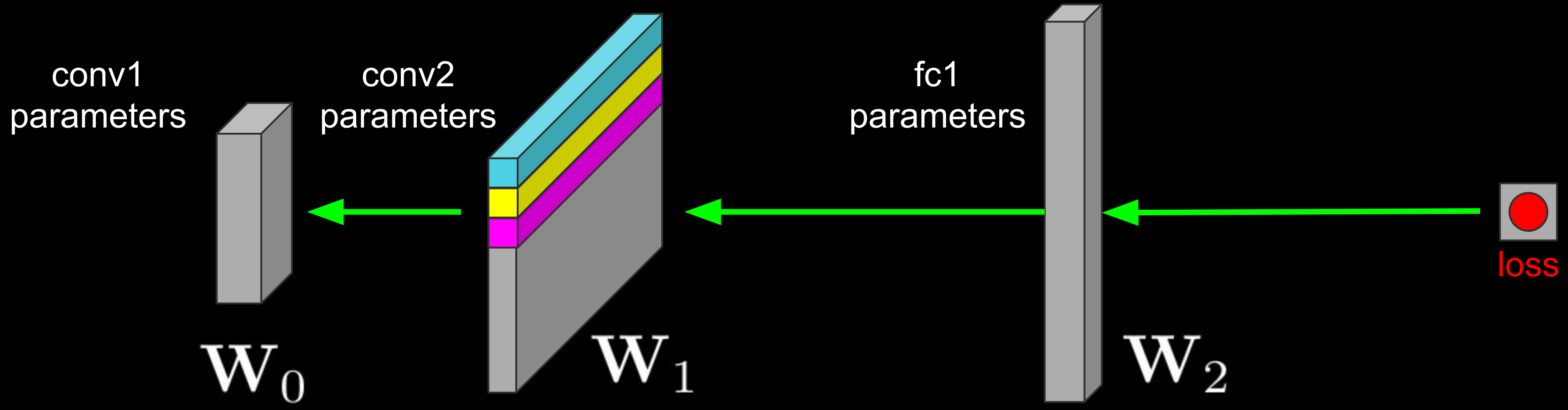
$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{W}_2}$$

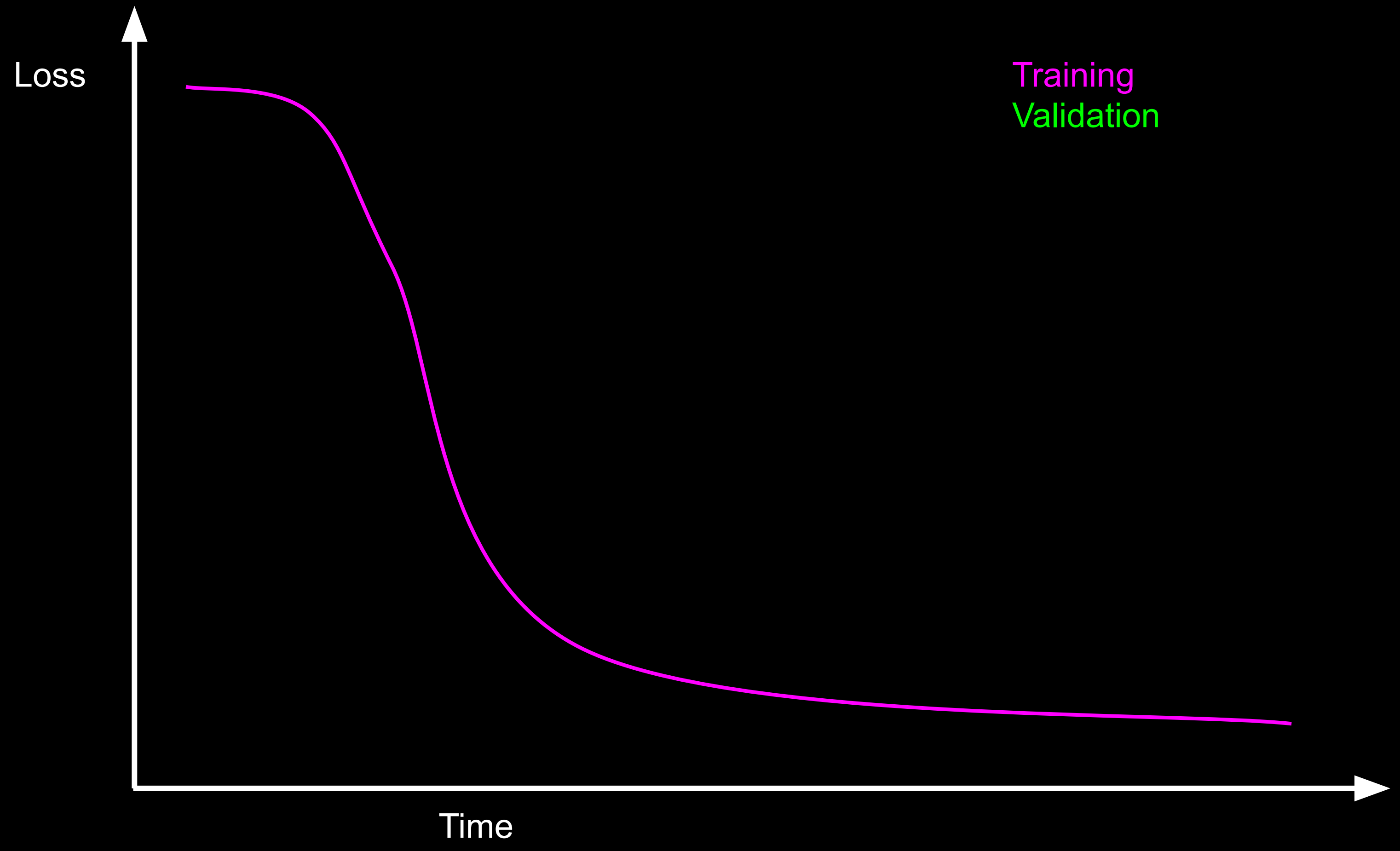


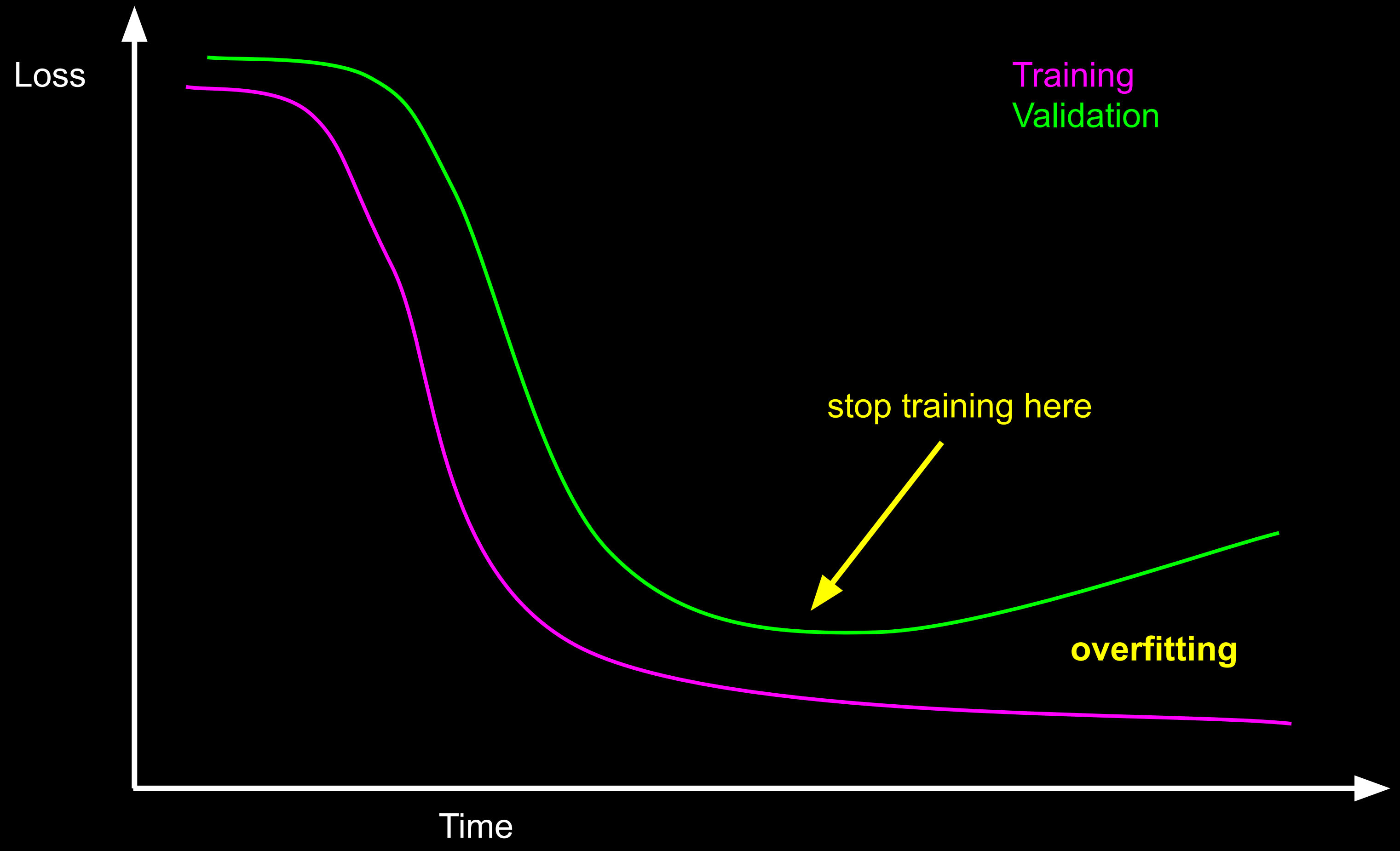
$$\frac{\partial L}{\partial \mathbf{W}_0} = \frac{\partial L}{\partial \mathbf{o}_2} \cdot \frac{\partial \mathbf{o}_2}{\partial \mathbf{o}_1} \cdot \frac{\partial \mathbf{o}_1}{\partial \mathbf{o}_0} \cdot \frac{\partial \mathbf{o}_0}{\partial \mathbf{W}_0}$$



http://cs231n.github.io/optimization-2/

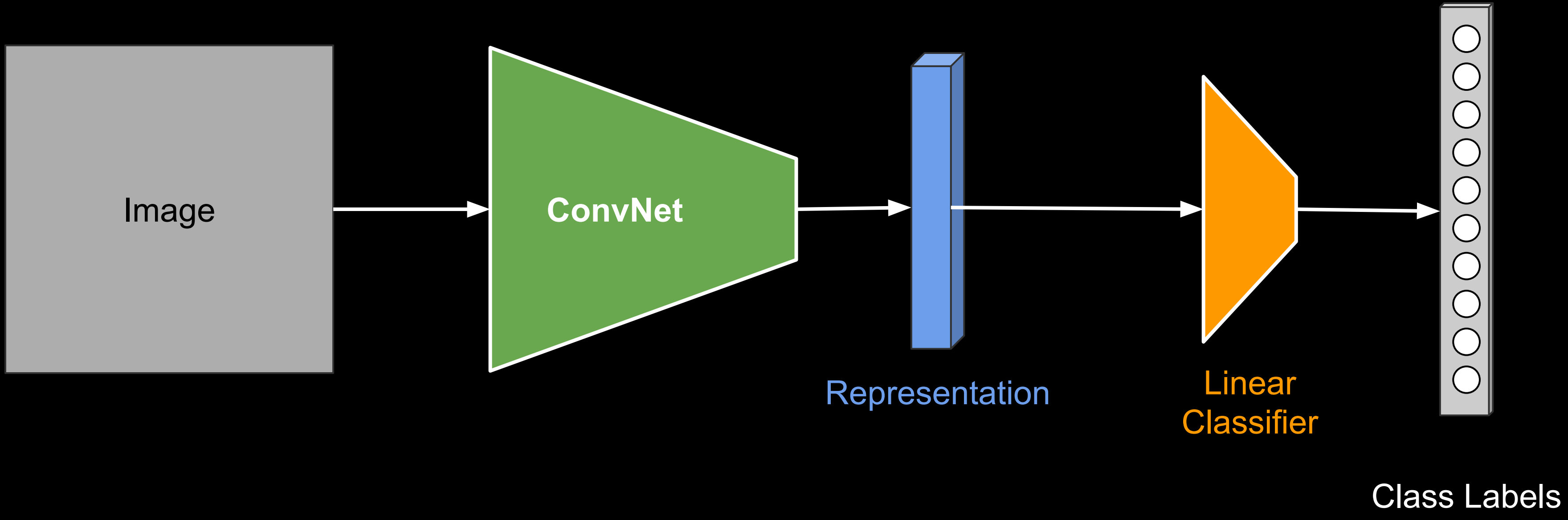

conv1
parameters
conv2
parameters
fc1
parameters
loss
$\mathbf{W}_0$
$\mathbf{W}_1$
$\mathbf{W}_2$


Loss
Training
Validation
Time

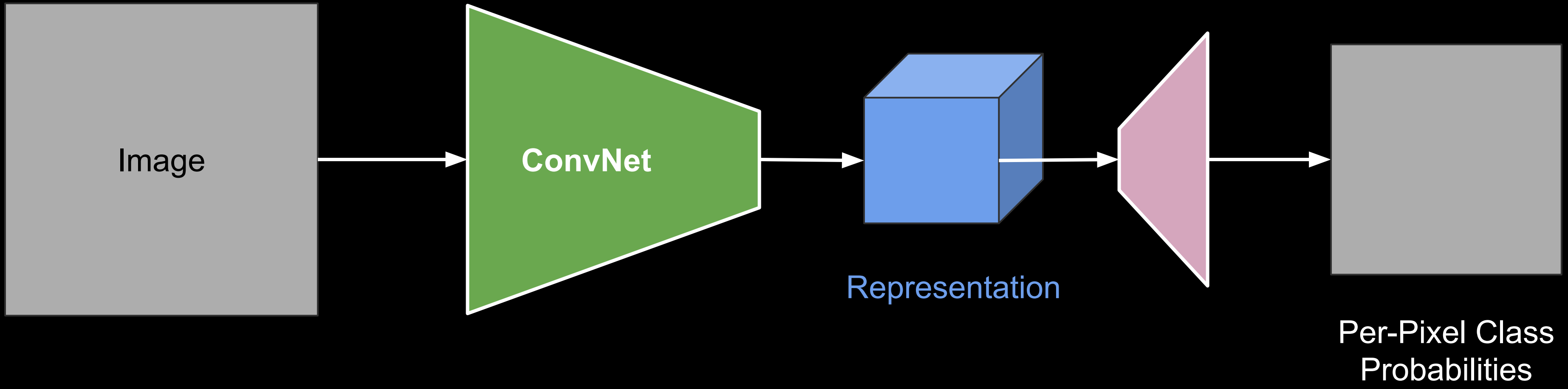

Loss
Training
Validation
stop training here
overfitting
Time

Applications

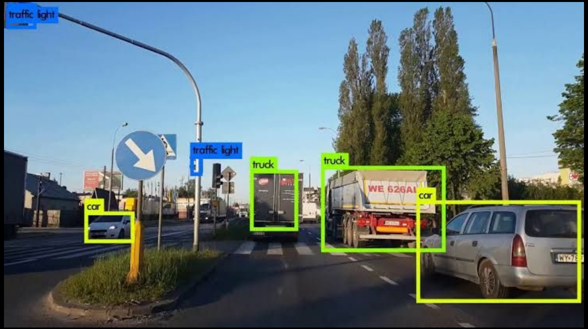
Image Classification

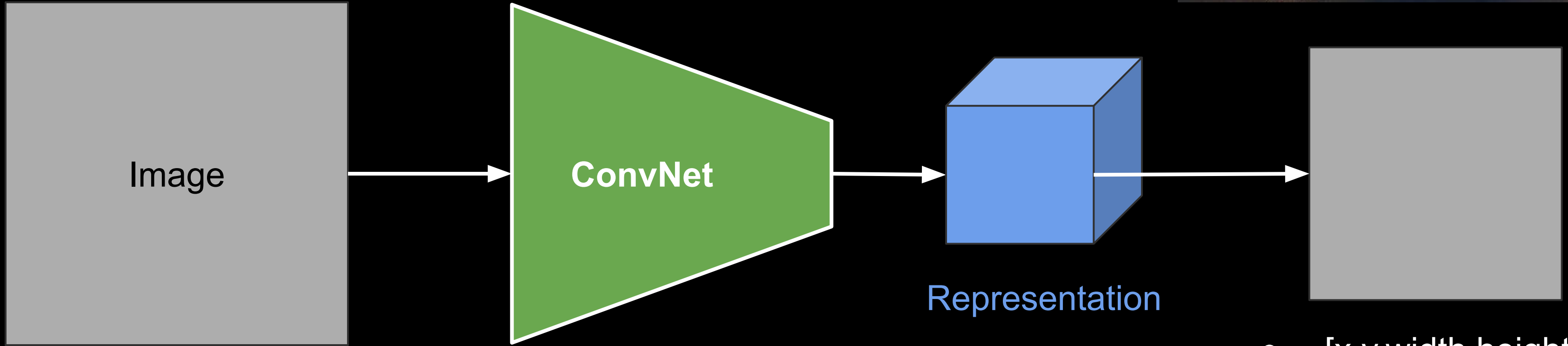


Semantic Segmentation



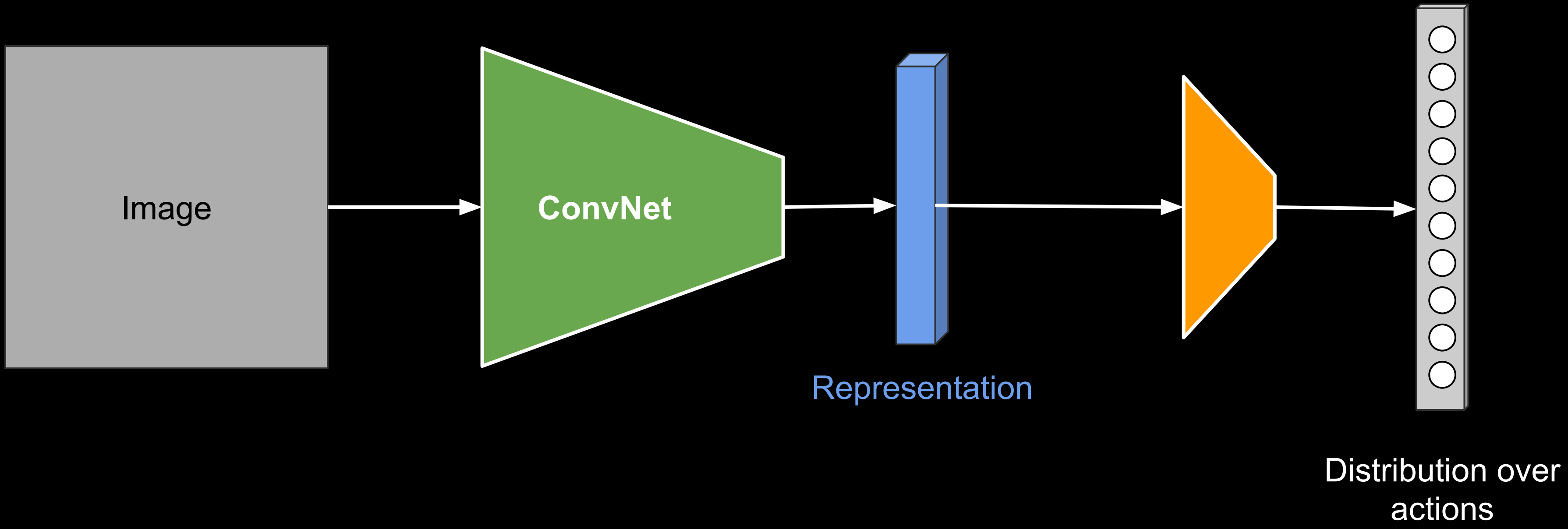
Object Detection



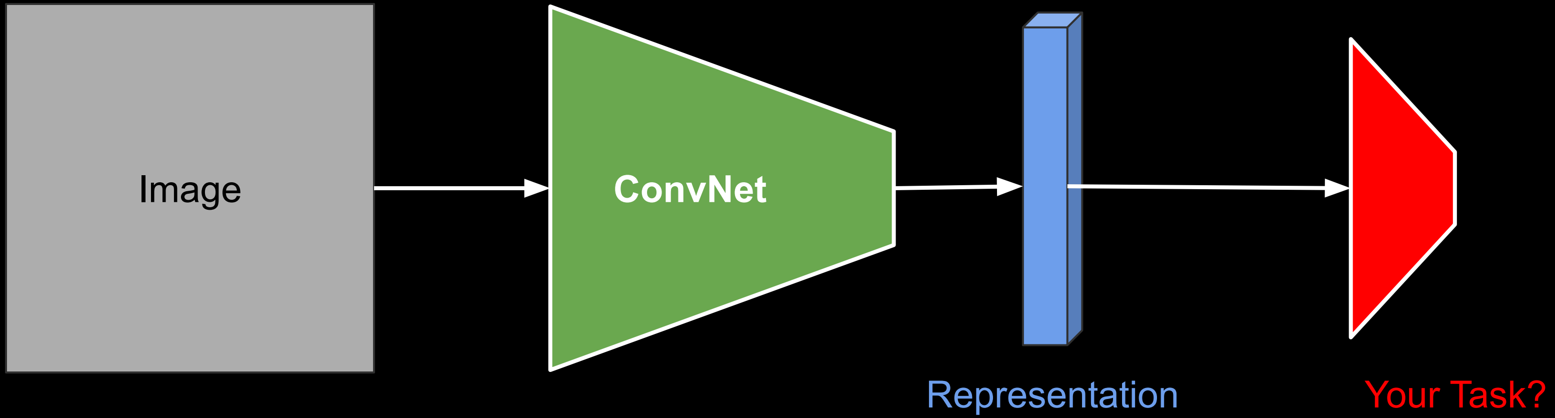


- [x,y,width,height]
- confidence
- class label

Reinforcement Learning



What is your task?



Fine Tuning

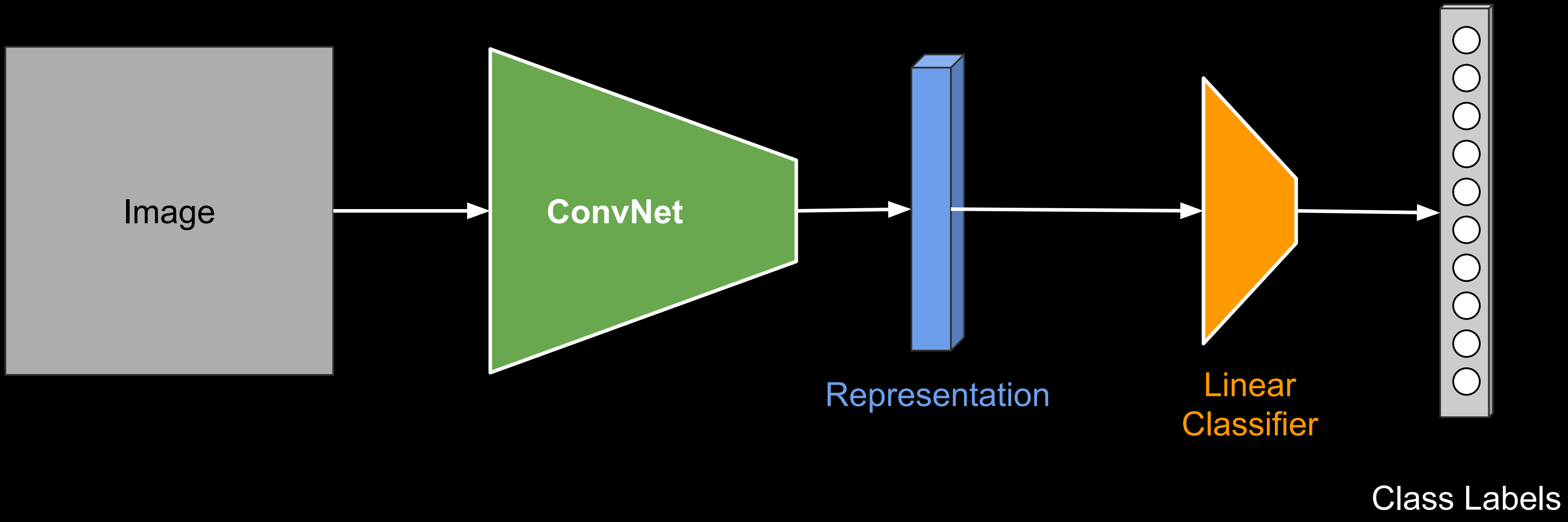
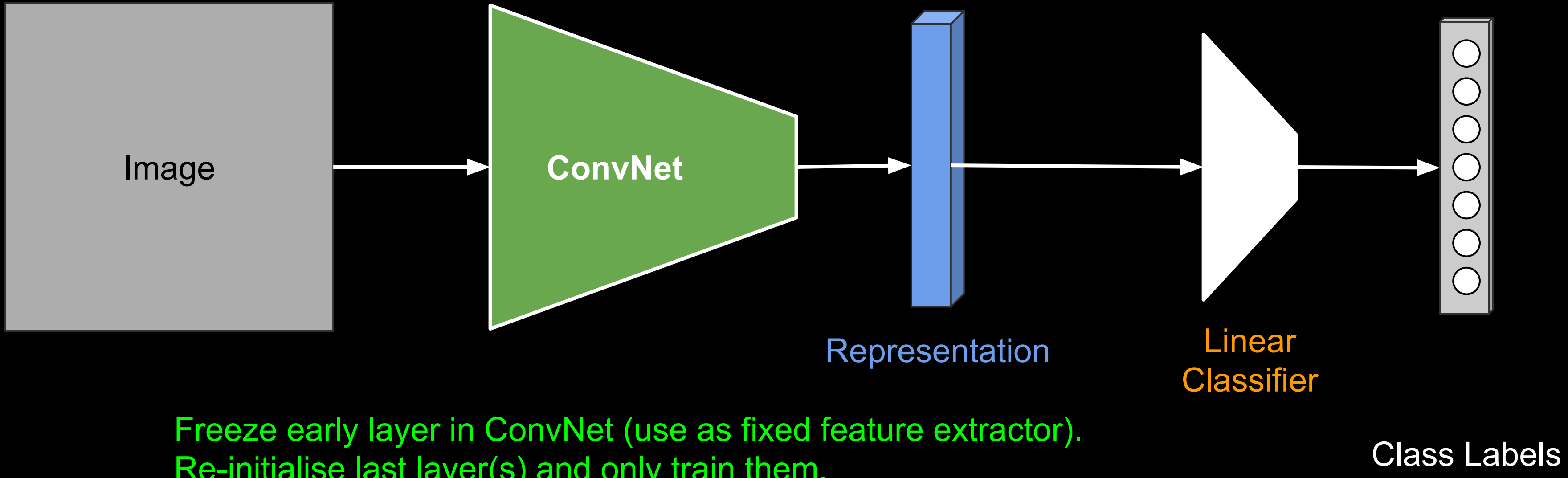


Image
ConvNet
Representation
Linear
Classifier
Class Labels

Fine Tuning



Freeze early layer in ConvNet (use as fixed feature extractor).
Re-initialise last layer(s) and only train them.

Tips and Tricks

http://karpathy.github.io/2019/04/25/recipe/

http://cs231n.github.io/neural-networks-3/

Deep Learning for Robotic Vision

An Introduction

Niko Suenderhauf

Queensland University of Technology
Australian Centre for Robotic Vision