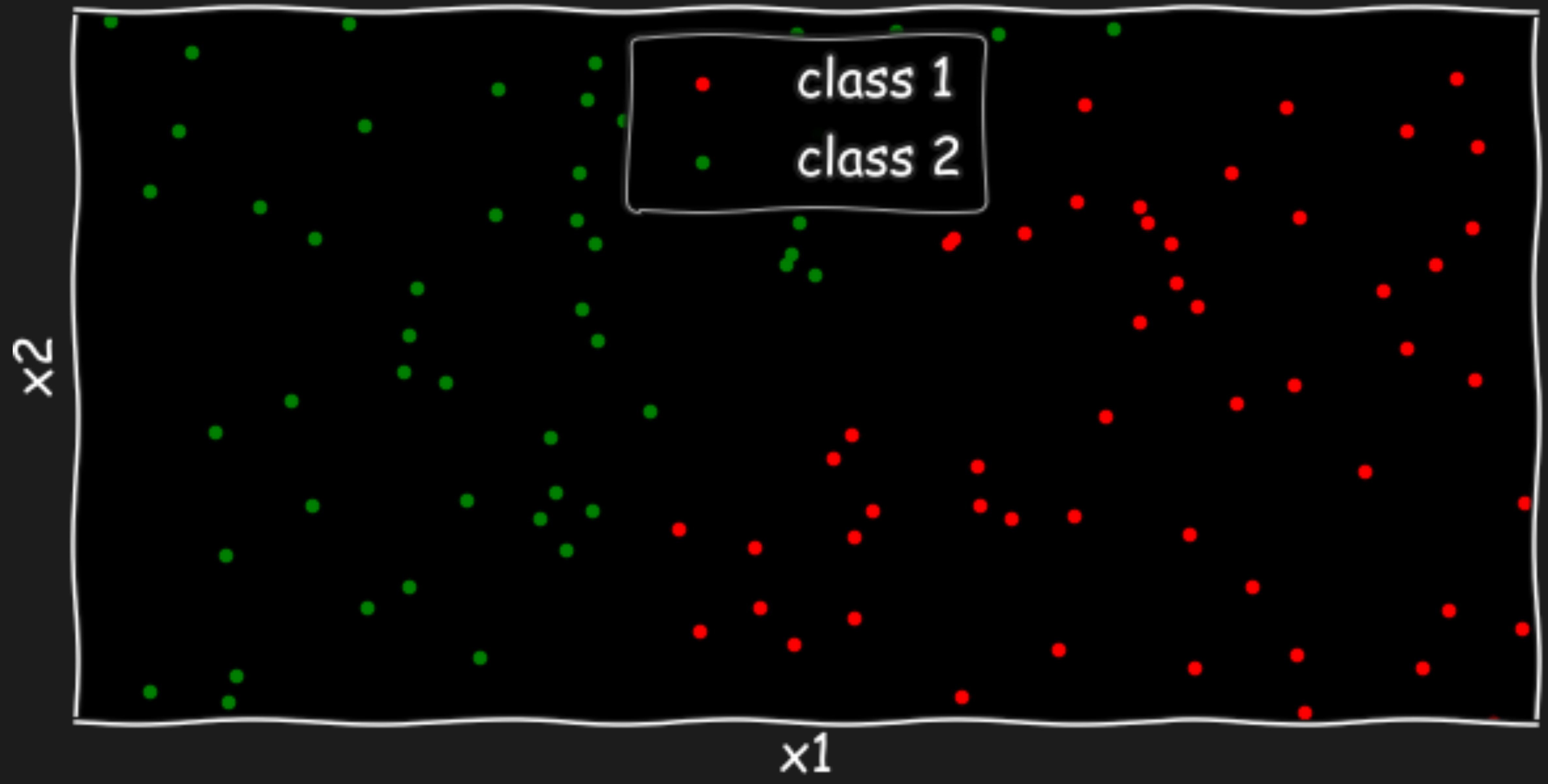
How to train an image classifier using PyTorch

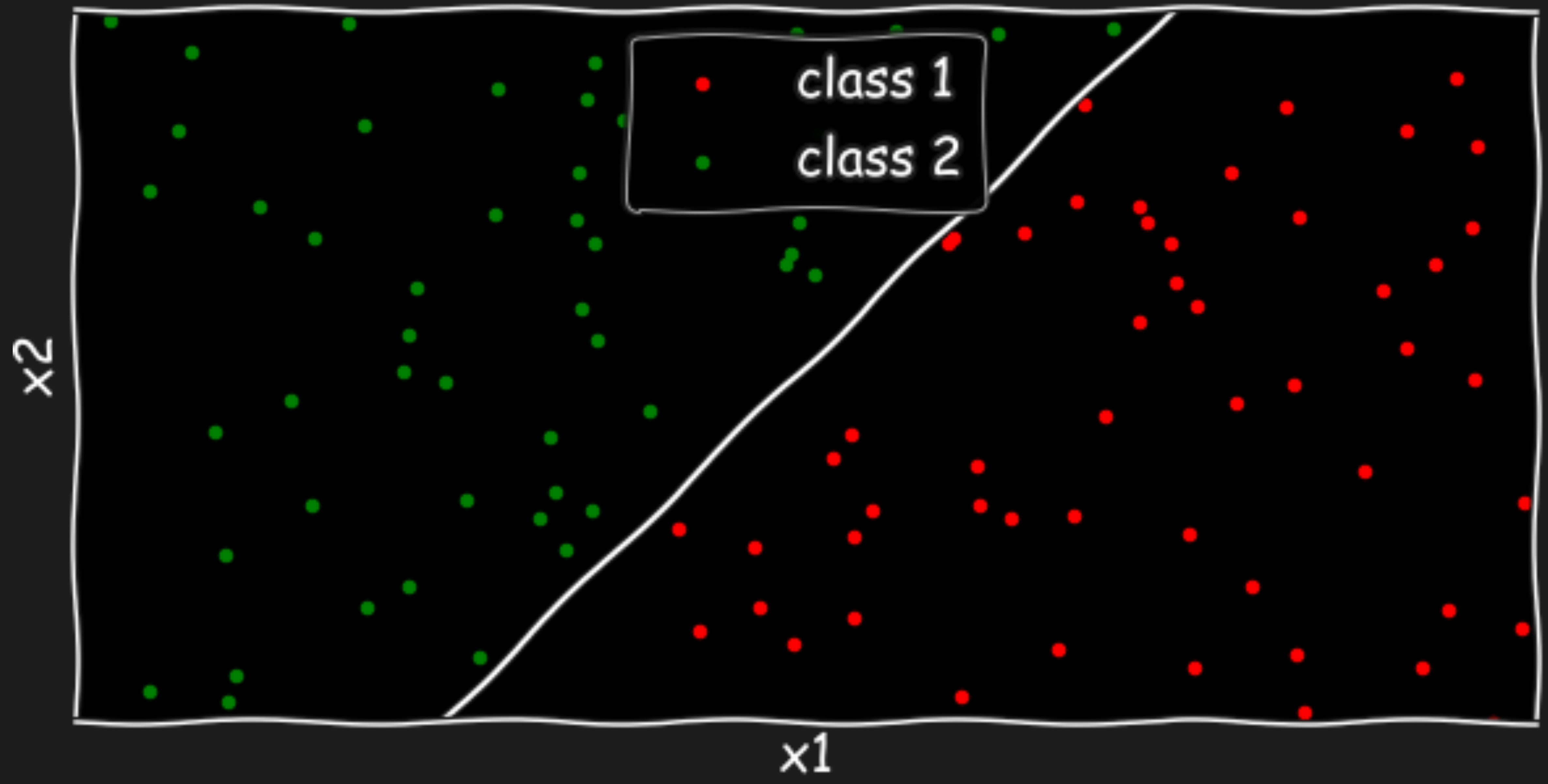
Rogier van der Geer -- GoDataDriven

- What is an image classifier?
- What is a neural network?
- How do you build one in PyTorch?
- What can you do with them?

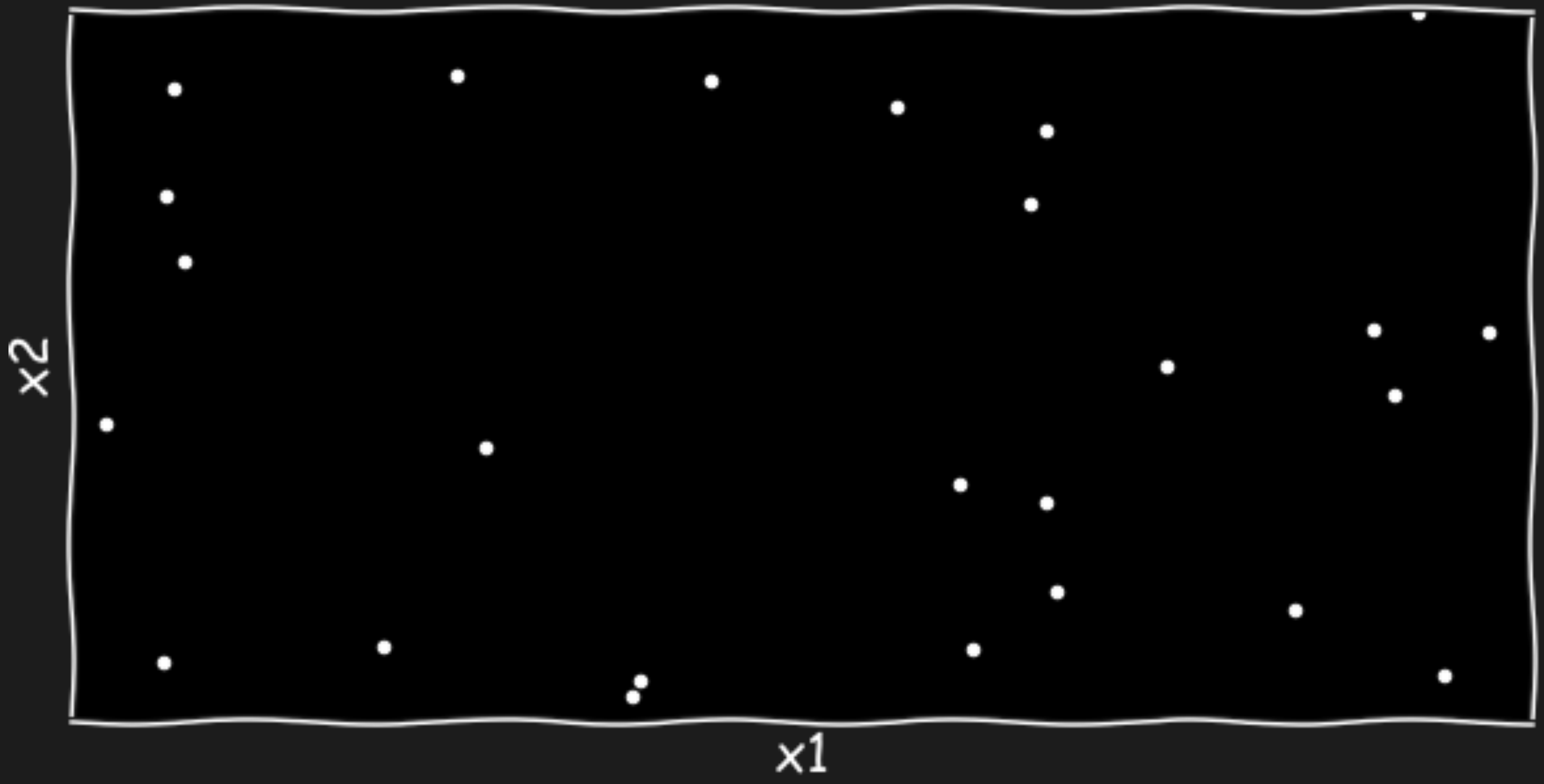
Labelled training **data set**



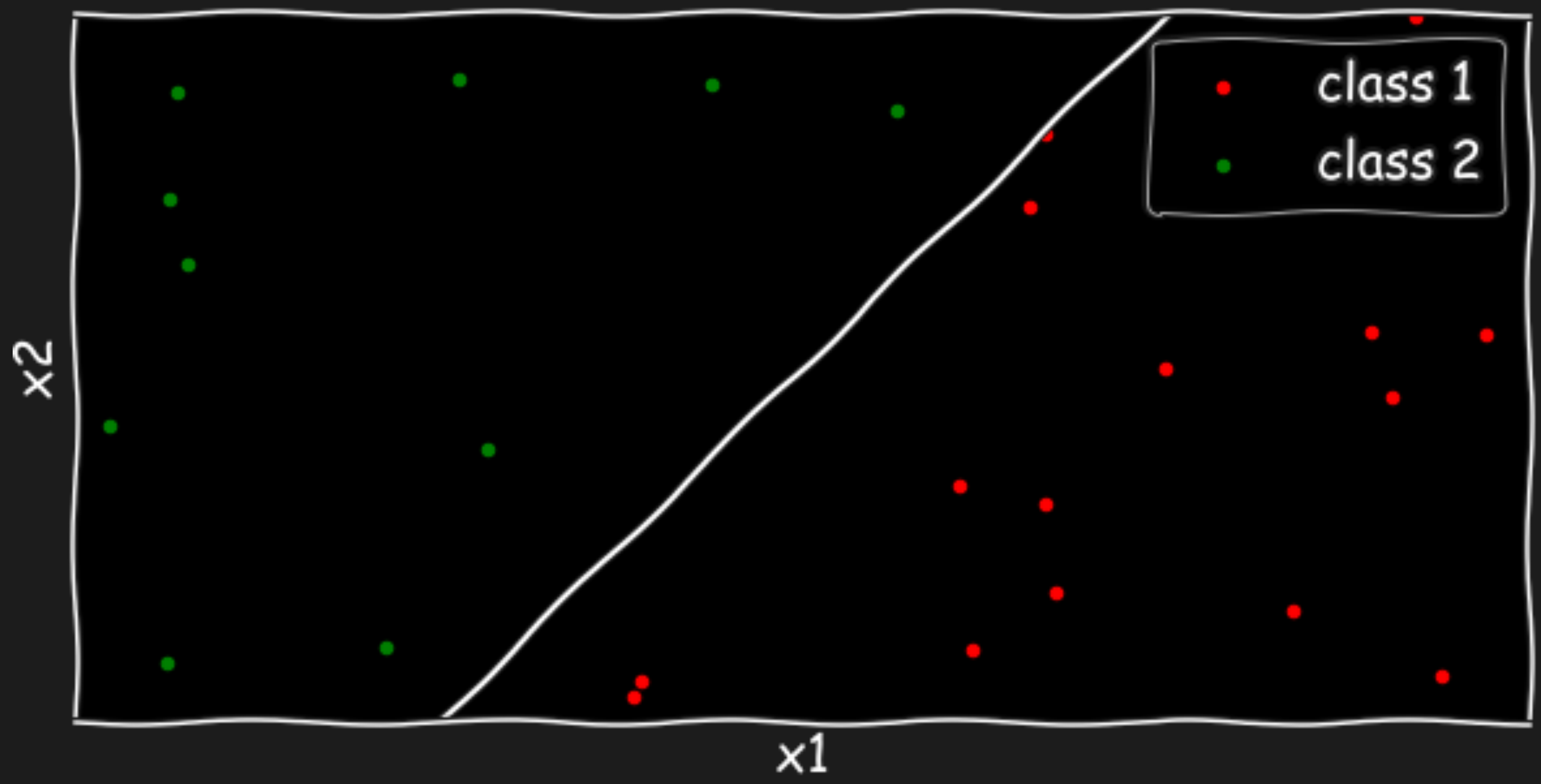
Simple classifier



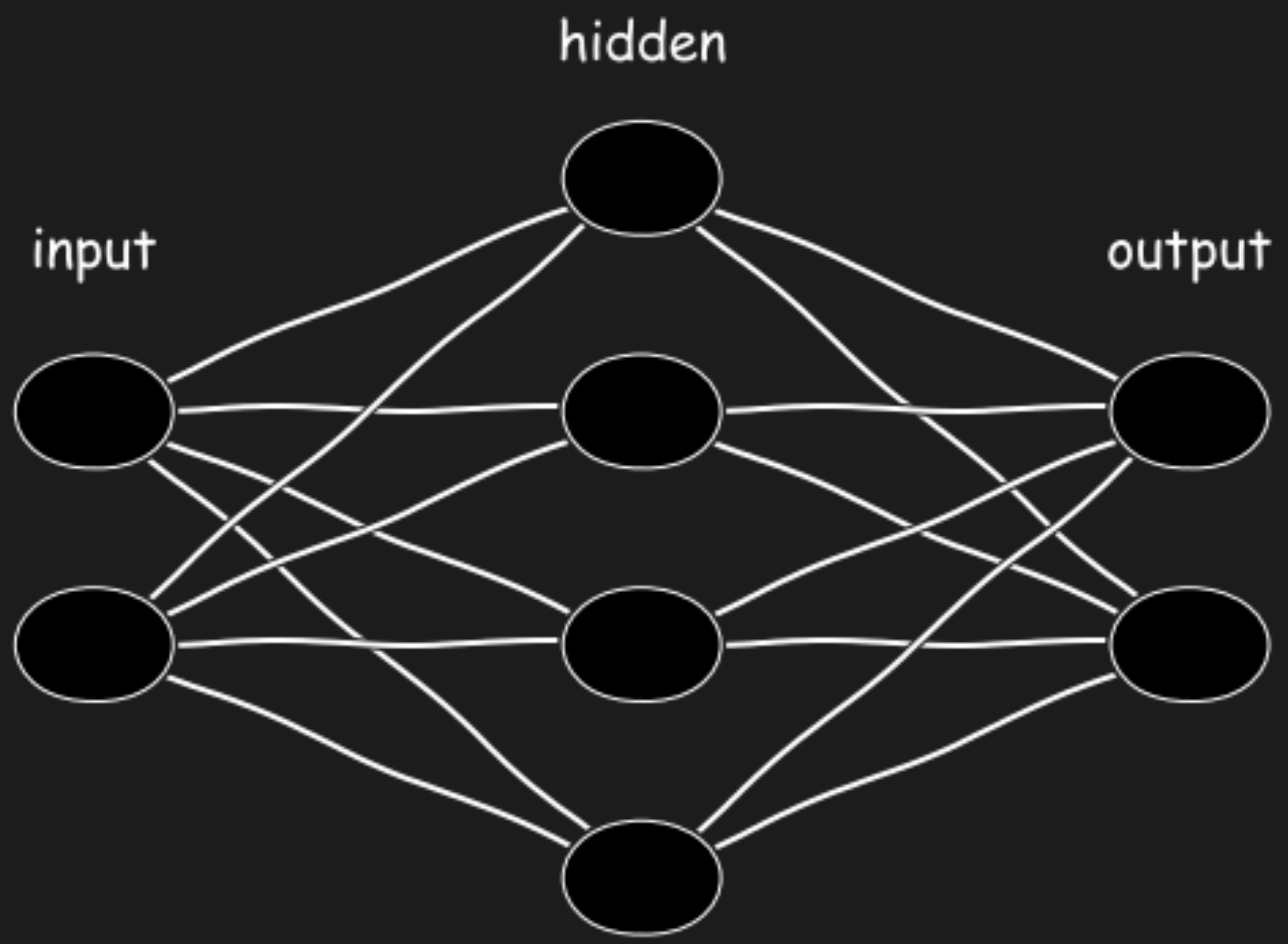
Unlabelled data set



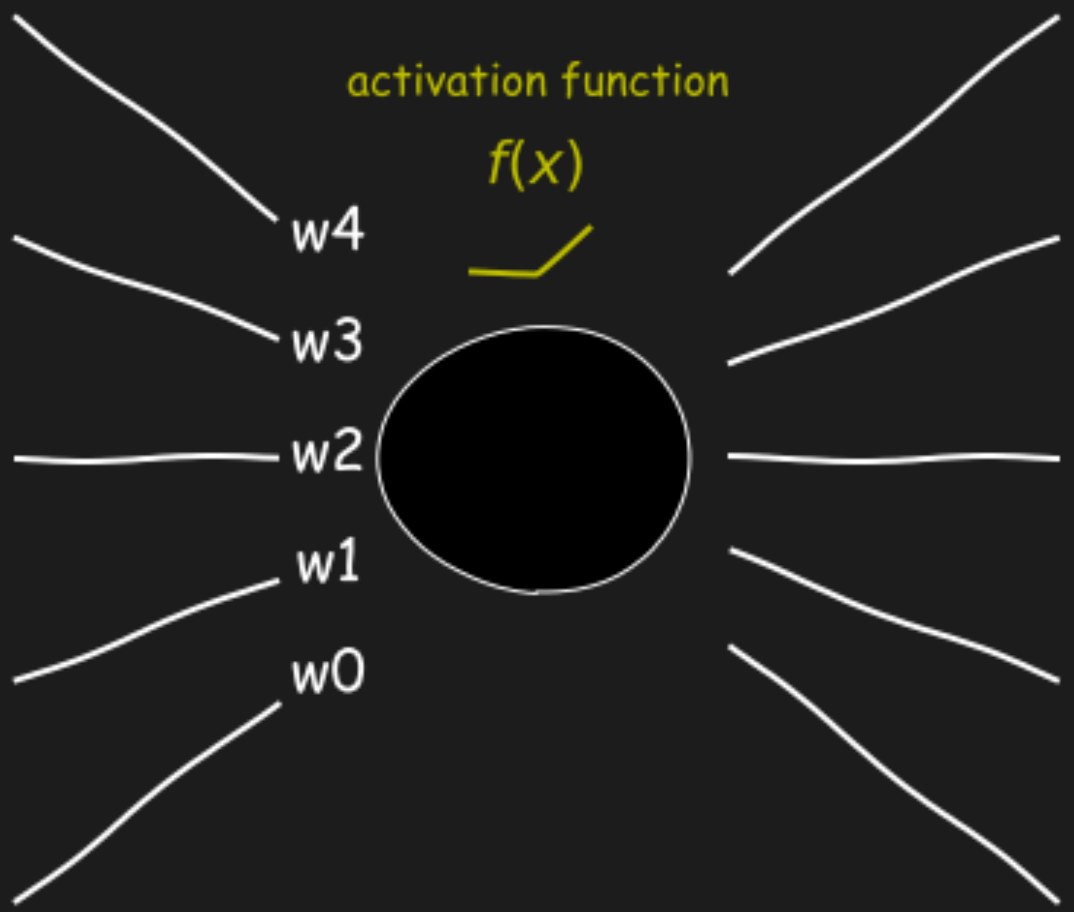
Classifications based on classifier



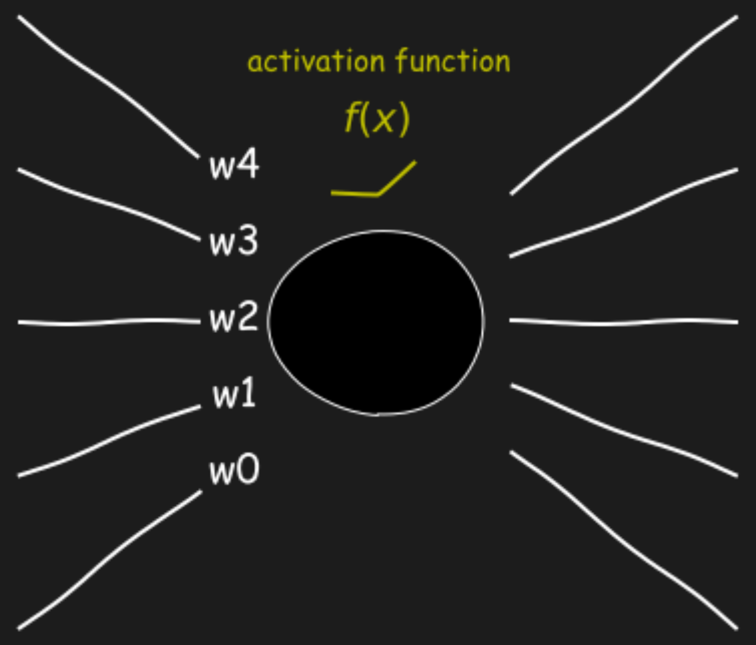
Neural network



Neuron



Neuron



Rectified linear unit:

$$y = max(0, x)$$

Neural network

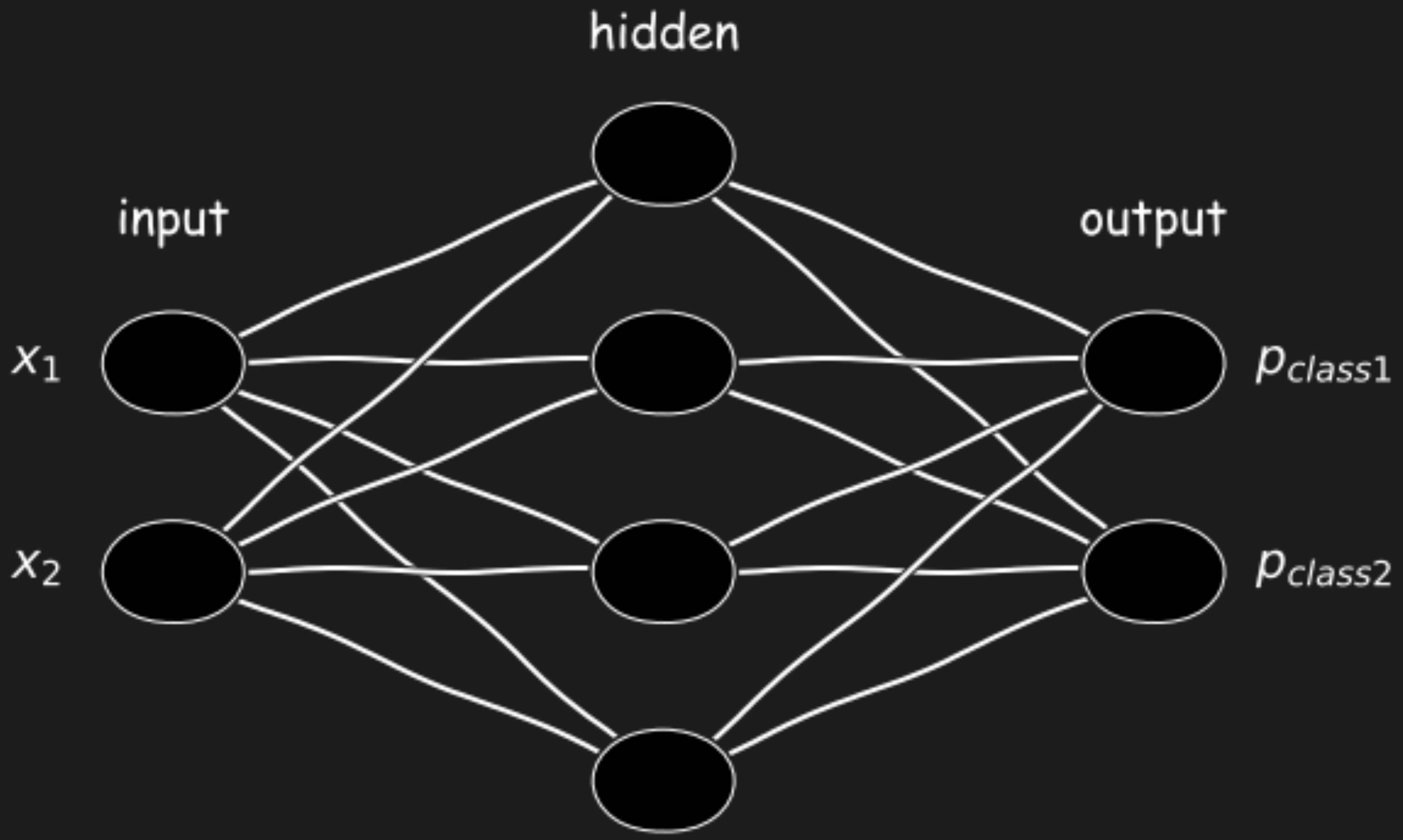


Image classification

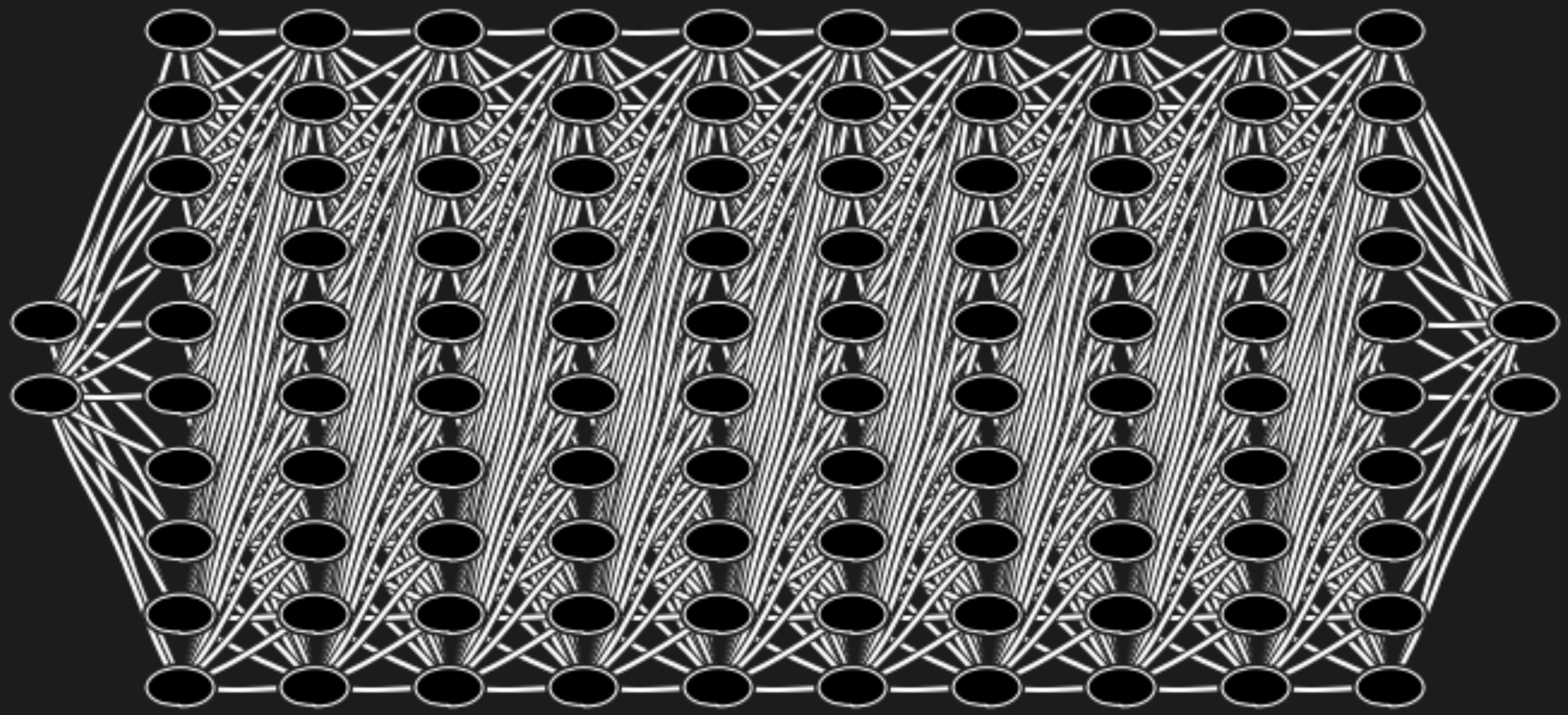
"Dog"

"Cat"

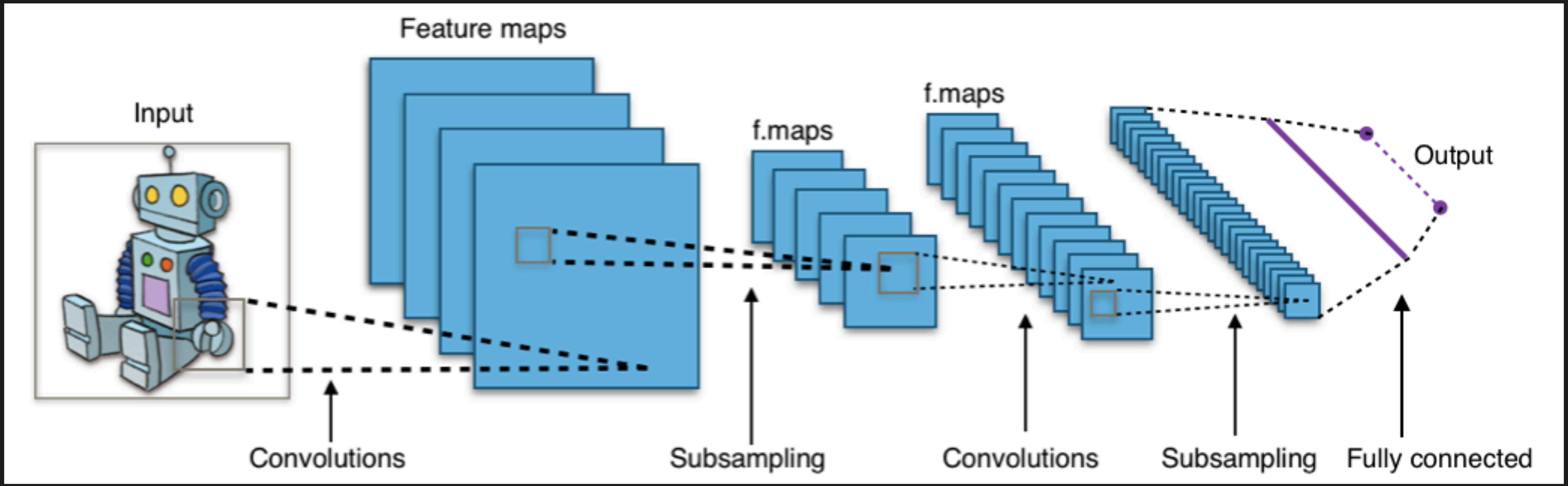
deep convolutional networks

Rogier van der Geer -- GoDataDriven

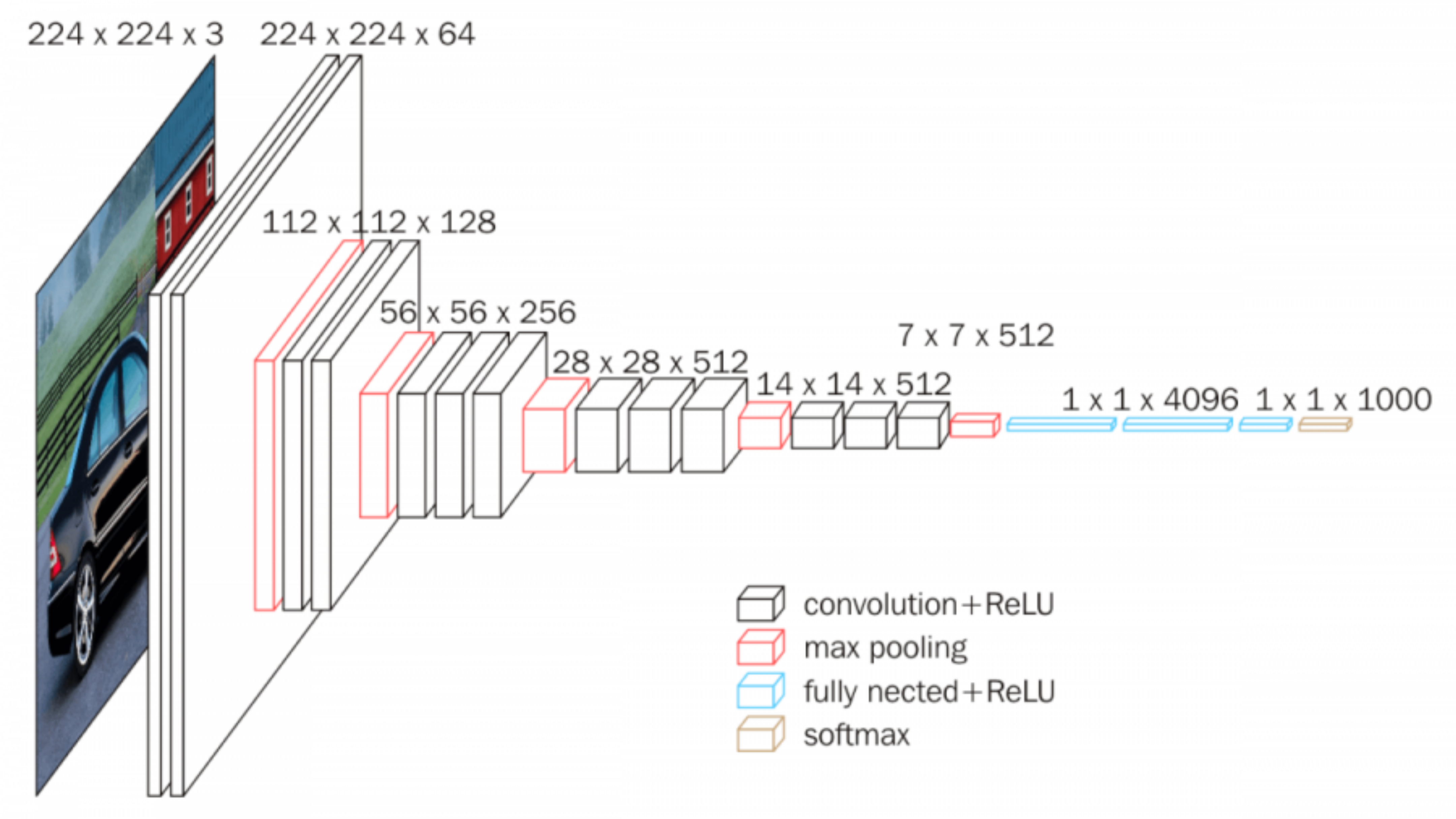
deep neural network



convolutional neural network



VGG16: 16 layers, 144 million weights

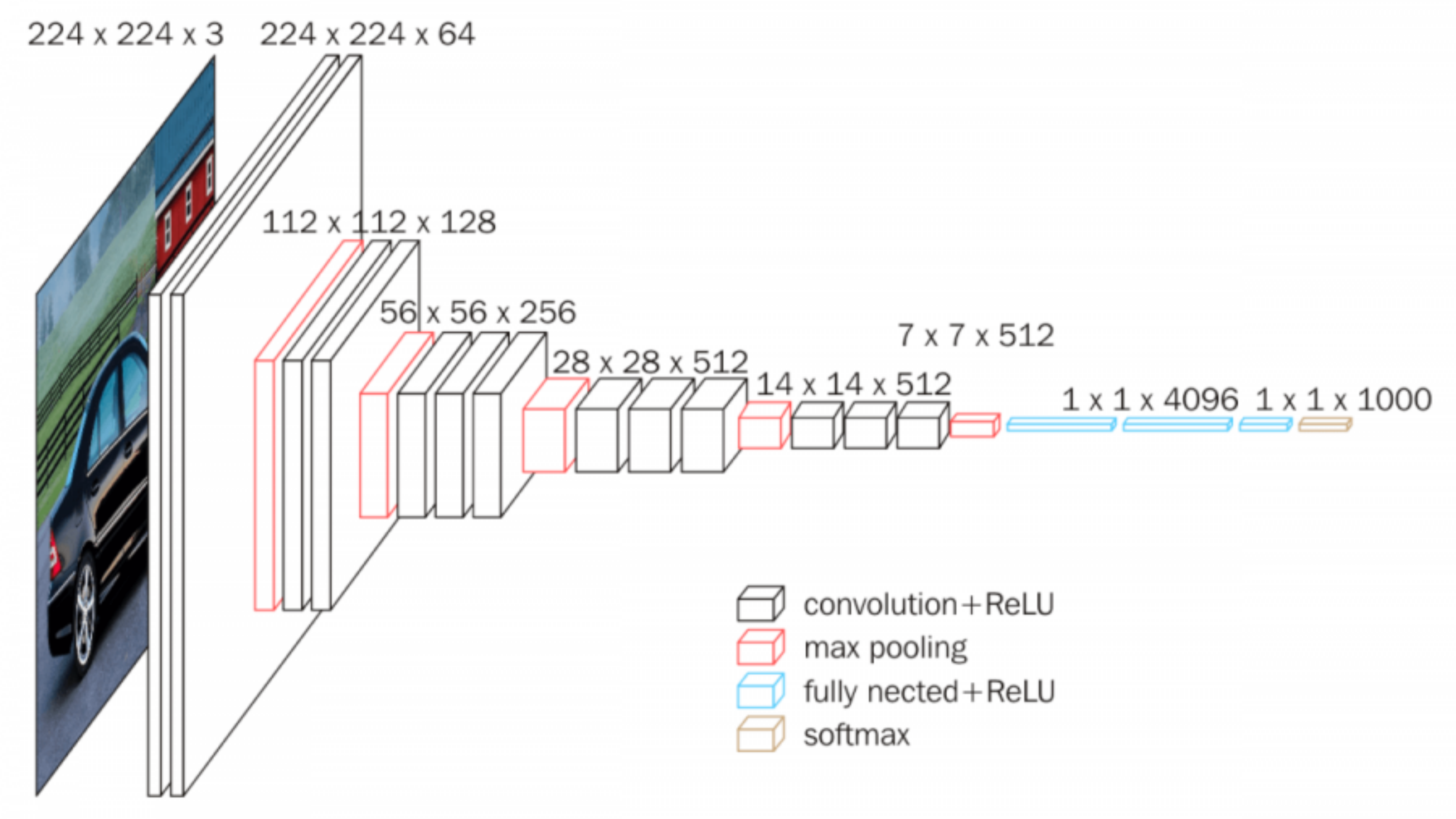


ImageNet

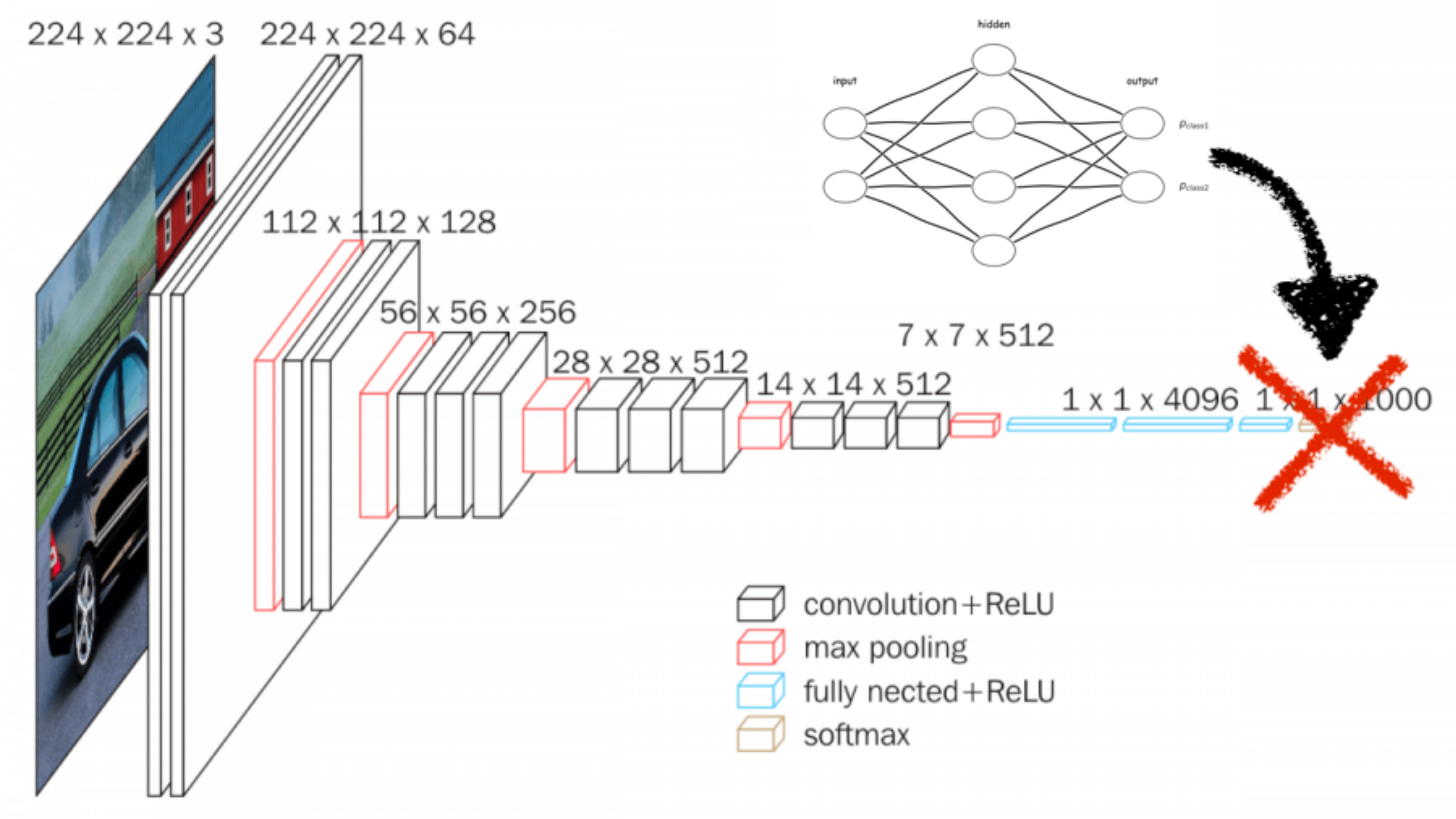
- 14 million images
- annotated into 1000 classes

VGG16: ~ 90% accuracy on 1000 classes

Transfer learning



Transfer learning



Why PyTorch not Keras?

- Keras was there first
- PyTorch is more flexible
- Keras is faster
- PyTorch lets you play with the internals

You learn more from PyTorch

PyTorch: define a model

```
from torch import nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(18 * 16 * 16, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):  # Input: 3 channels, 32x32
        x = F.relu(self.conv(x))  # Converts to 18 channels, 32x32
        x = self.pool(x)  # Pooling reduces to 18 channels, 16x16
        x = x.view(-1, 18 * 16 * 16)  # Reshape to a 1D vector of size 4608
        x = F.relu(self.fc1(x))  # Apply first FC layer, output has size 64
        x = self.fc2(x)  # Apply second FC layer, output has size 10
        return x
```

PyTorch: loading a pre-trained model

```
from torchvision.models import squeezenet1_0  # Or VGG

model = squeezenet1_0(pretrained=True)
```

```python
from torchvision.models import squeezenet1_0

print(squeezenet1_0(pretrained=True))
```

```
SqueezeNet(
  (features): Sequential(
    (0): Conv2d(3, 96, kernel_size=(7, 7), stride=(2, 2))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
    ...
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5)
    (1): Conv2d(512, 1000, kernel_size=(1, 1), stride=(1, 1))
    (2): ReLU(inplace)
    (3): AvgPool2d(kernel_size=13, stride=1, padding=0)
  )
)
```

PyTorch: pre-trained model

```python
from torch import nn
from torchvision.models import squeezenet1_0

n_classes = 4

model = squeezenet1_0(pretrained=True)
model.num_classes = n_classes
model.classifier[1] = nn.Conv2d(512, n_classes, kernel_size=(1, 1), stride=(1, 1))
```

Train your model

```python
from torch import nn, optim

model.train()  # Set your model to training mode

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1E-3, momentum=0.9)

for inputs, labels in loader:  # Multiple images at once
    optimizer.zero_grad()  # Reset the optimizer
    outputs = model(inputs)  # Forward pass

    loss = criterion(outputs, labels)  # Compute the loss

    loss.backward()  # Backward pass
    optimizer.step()  # Optimize the weights
```

One loop through all training images is an epoch.

Evaluation

```python
from torch import max, no_grad

model.eval()  # Set model to evaluation mode: disable dropout etc

loss = 0
with no_grad():
    for inputs, labels in loader:
        outputs = model(inputs)
        _, predictions = max(outputs.data, dim=1)  # Returns (values, indices)
        loss += criterion(outputs, labels)
```

Loading data: the dataset

```python
from torchvision import transforms
from torchvision.datasets import ImageFolder

train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])
test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()
])

train_set = ImageFolder(path_to_train_images, transform=train_transform)
test_set = ImageFolder(path_to_test_images, transform=test_transform)
```

Loading data: the loader

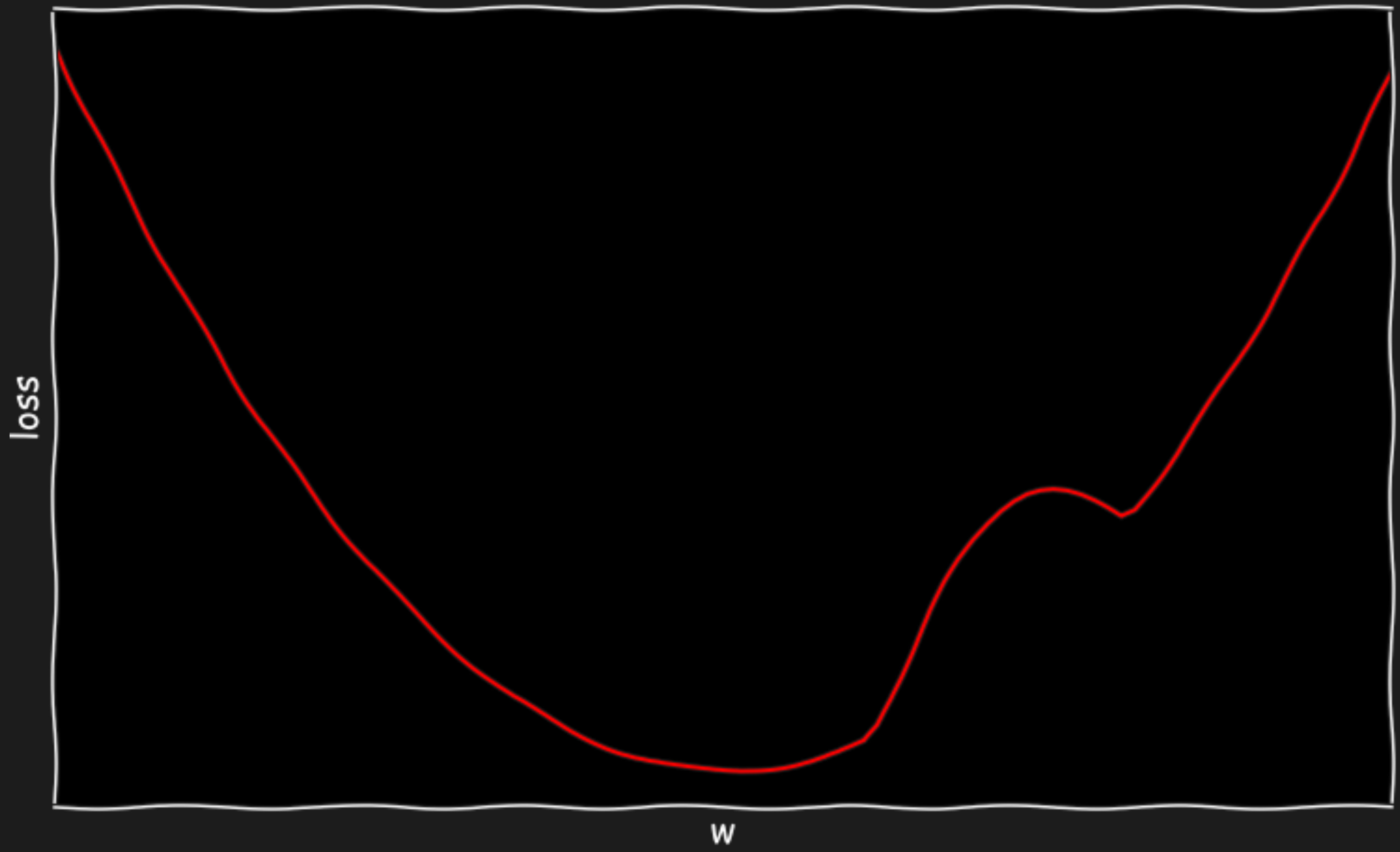
```python
from torch.utils.data import DataLoader

train_loader = DataLoader(
    dataset=train_set,
    batch_size=32,
    num_workers=4,
    shuffle=True,
)

test_loader = DataLoader(
    dataset=test_set,
    batch_size=32,
    num_workers=4,
    shuffle=True,
)
```

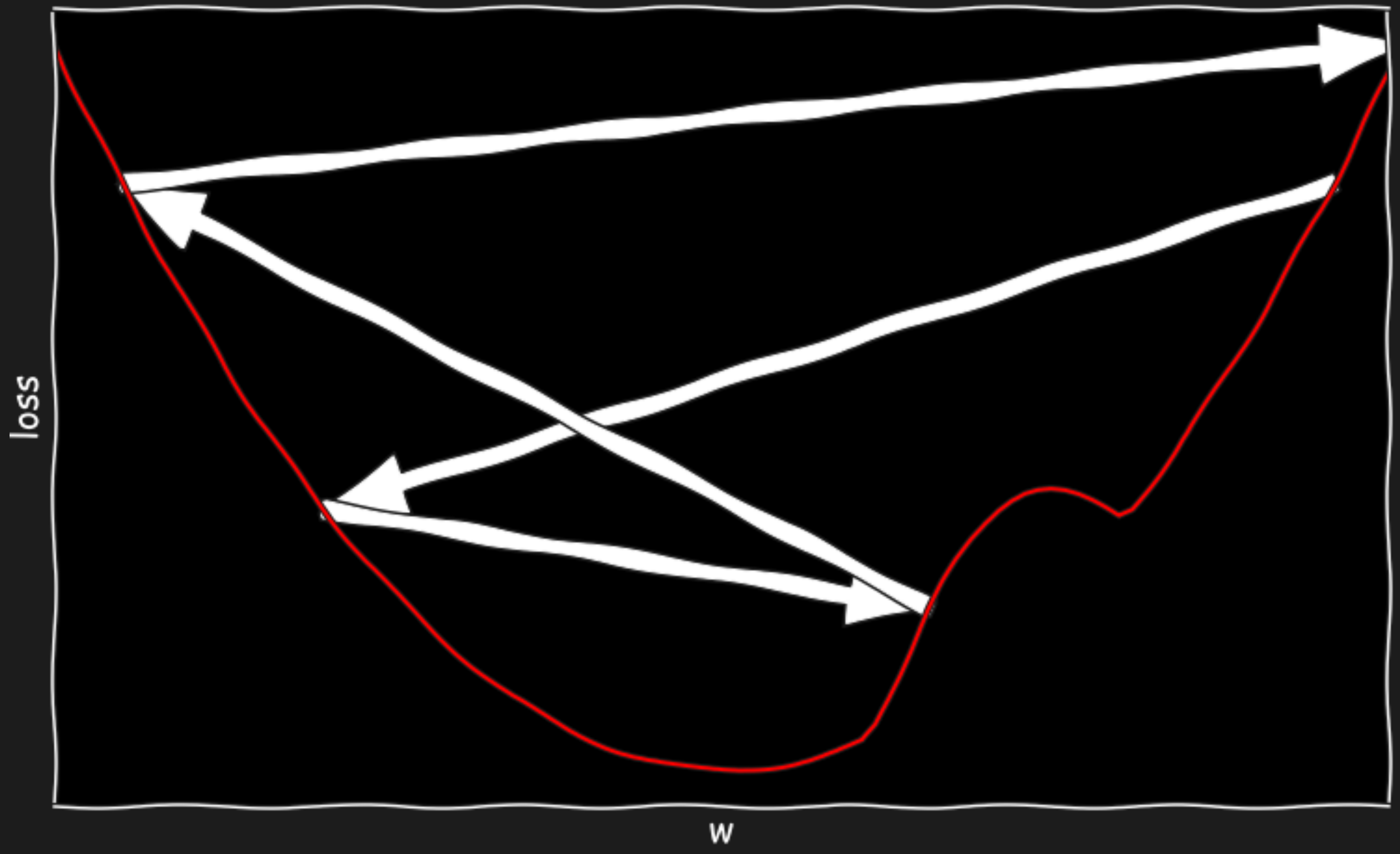
Learning rate

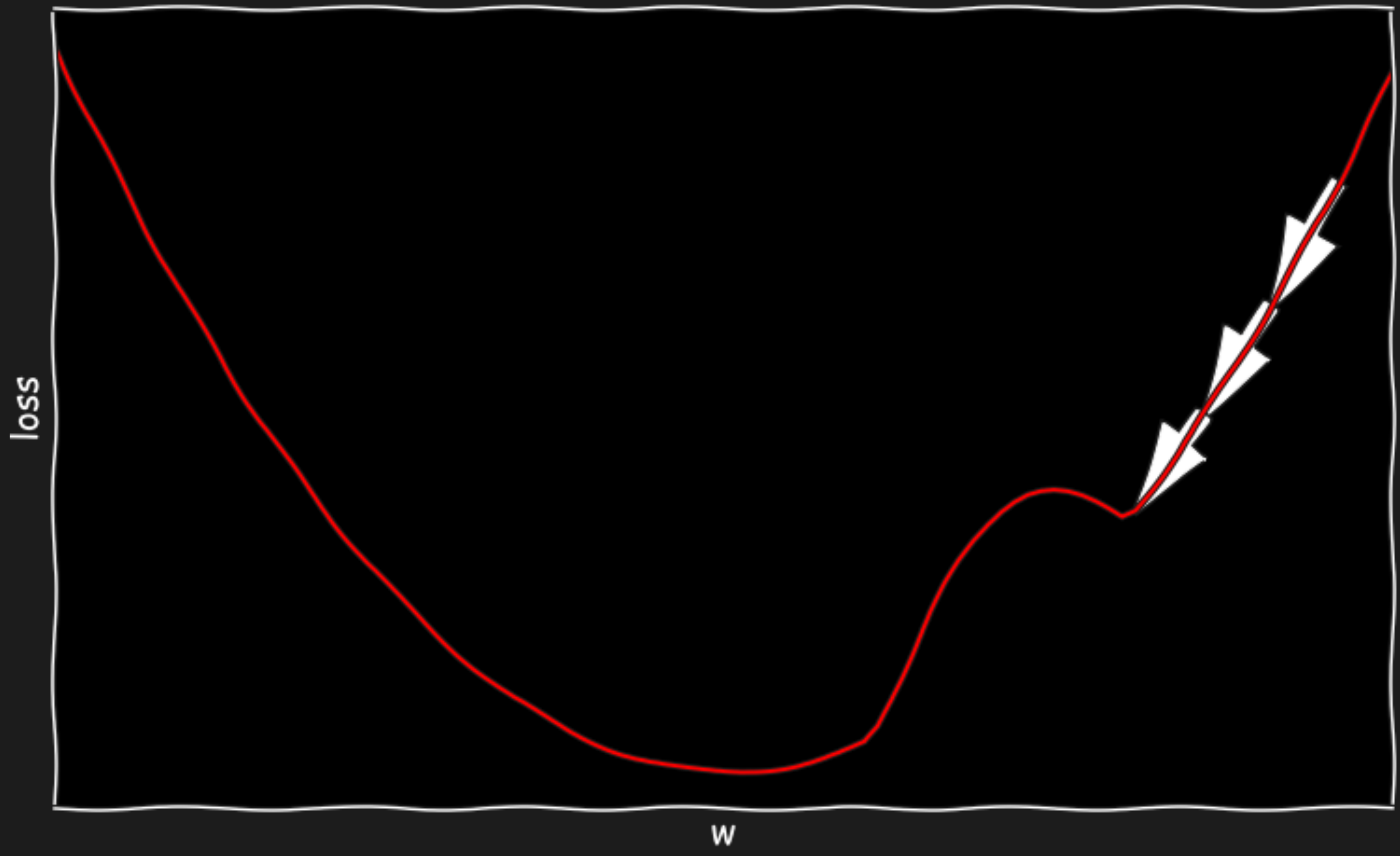
Remember our optimizer:

```
optimizer = SGD(model.parameters(), lr=1E-3, momentum=0.9)
```

Here `lr` is our learning rate, the rate at which we change the weights when training. What is a good value?


loss
w

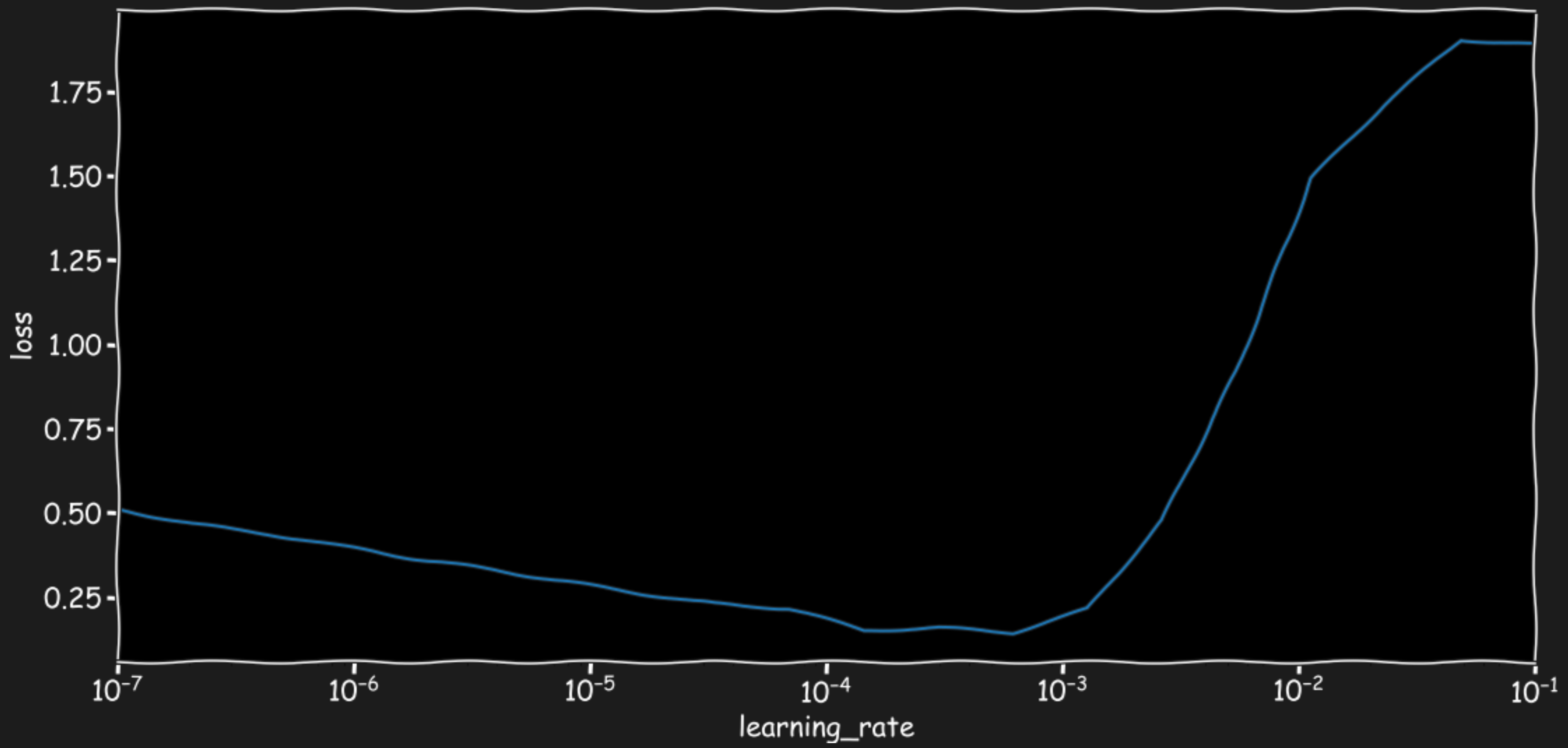

loss
w


loss
w

A learning rate sweep

```
def set_learning_rate(optimizer, learning_rate):
    for param_group in optimizer.param_groups:
        param_group['lr'] = learning_rate

learning_rates = np.logspace(min_lr, max_lr, num=n_steps)
results = []
for learning_rate in learning_rates:
    set_learning_rate(optimizer, learning_rate)
    train_batches(...)
    results.append(evaluate_batches(...))
```

Learning rate sweep plot



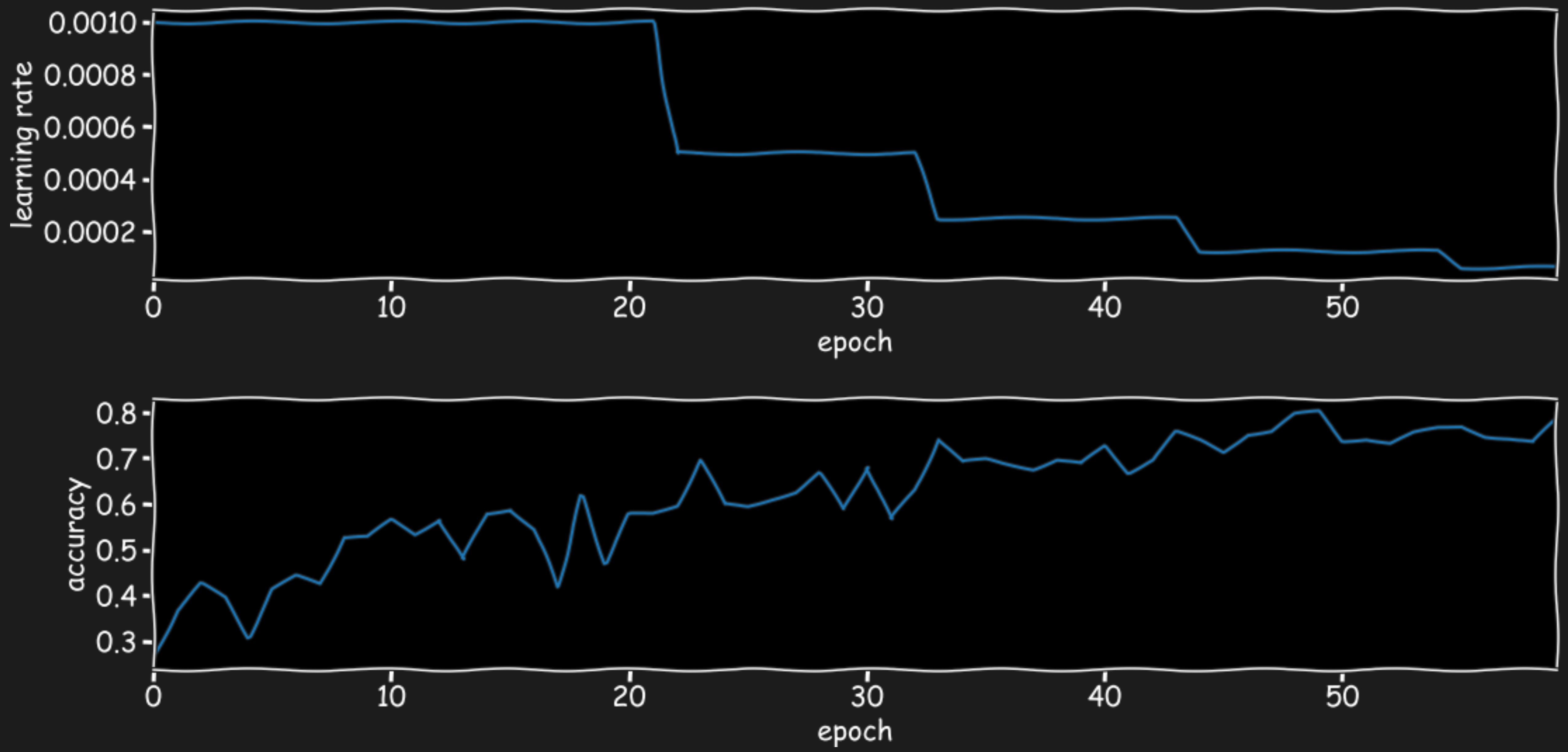
Learning rate scheduler

```
from torch.optim.lr_scheduler import ReduceLROnPlateau

scheduler = ReduceLROnPlateau(optimizer, factor=0.5, patience=25)
```

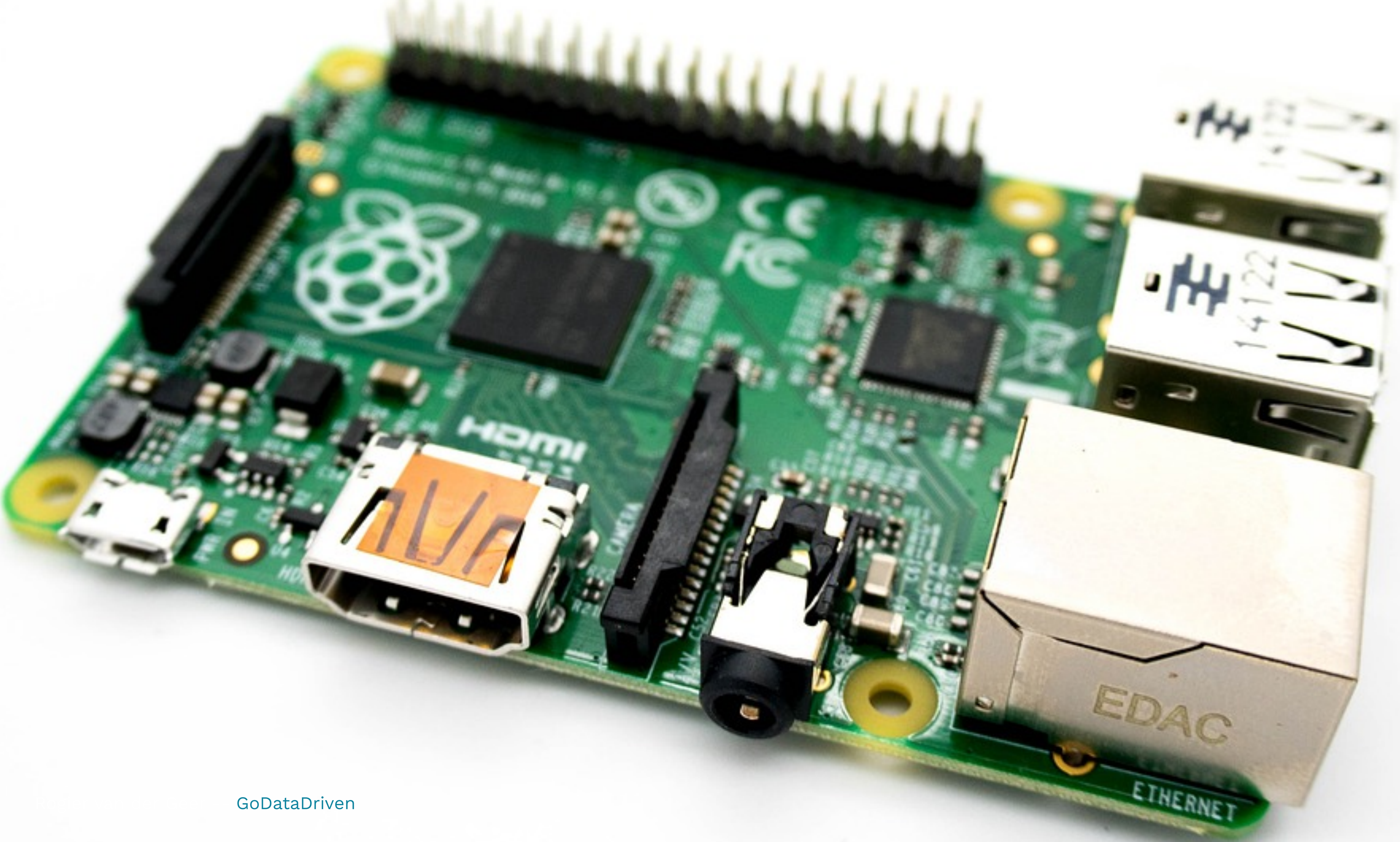
After every training epoch:

```
scheduler.step(test_loss)
```

Learning rate step plot



Data


HDMI
EDAC
ETHERNET

Data set

Photos taken in the worlds largest cities

- 72 cities
- ~ 0.5M images
- 10k photographers
- ~ 30 GB
- licensed for reuse

Rogier van der Geer -- GoDataDriven

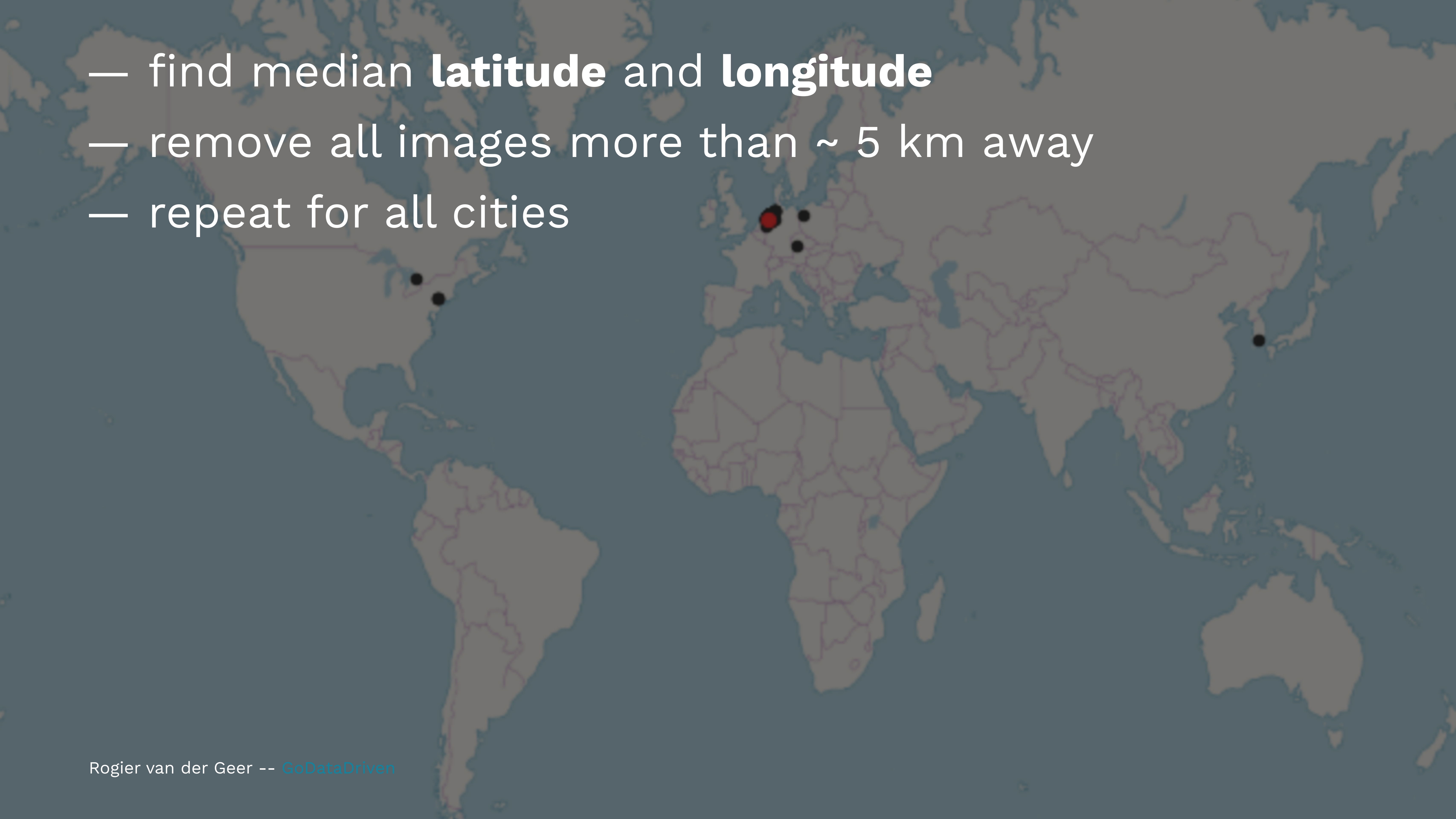
Rogier van der Geer -- GoDataDriven

— Amsterdam

- Amsterdam
- Dublin

Dublin, Terminal 2, Amsterdam, Schiphol, Seoul, Incheon, Taipei, Taoyuan, Hong Kong, Airport, Citygate, Aer Lingus, KLM, Korean Air, Eva Air, Cathay Pacific, Jeju, Gimpo, Hyatt Regency, Grand Hyatt, The Sherwood Hotel, Regent Hotel, Park Hyatt, Intercontinental, COEX, Taipei 101, Elite Concepts, cars, ICC, Ritz Carlton, W Hotel Hong Kong, breakfast, lunch, dinner, room service, french toast, ice cream, birthday, Mercedes, Hyundai, Kia, BMW, Bentley, Bongeunsa, Buddhist temple, Shilla, Lotte, cocktails, Taxis, transport, traffic, landmark, watch, bed, bathroom, suite, rooms, facades, architecture, street art, candid, men, girls, people, Jungmun beach, Teddy Bear Museum, Grand Club, Regency Club, irish love...

- find median **latitude** and **longitude**
- remove all images more than ~ 5 km away
- repeat for all cities



Other tags

city, street, sony, square, belgium, squareformat, architecture, london, photography, australia, brussels, 2016, art, urban, tokyo, bruxelles, japan, park, berlin, paris, night, travel, 2018, sky, ilce6500, sonyilce6500, california, sydney, streetphotography, nikon, chicago, people, building, belgique, spain, de, new, barcelona, nyc, losangeles, 2015, music, highiso, europe, museum, usa, amsterdam, concert, toronto, 日本, england, skyline, bxl, bru, france, switzerland, 東京, live, manhattan, canada, downtown, photoderue, sport, outdoor, china, rome, uk

Other tags

city, street, sony, square, belgium, squareformat, architecture, london, photography, australia, brussels, 2016, art, urban, tokyo, bruxelles, japan, park, berlin, paris, night, travel, 2018, sky, ilce6500, sonyilce6500, california, sydney, streetphotography, nikon, chicago, people, building, belgique, spain, de, new, barcelona, nyc, losangeles, 2015, music, highiso, europe, museum, usa, amsterdam, concert, toronto, 日本, england, skyline, bxl, bru, france, switzerland, 東京, live, manhattan, canada, downtown, photoderue, sport, outdoor, china, rome, uk

Top 10 most common cities

city	# images
london	1677
new york city	1320
chicago	909
toronto	521
sydney	203
los angeles	201
tokyo	191
philadelphia	175
houston	173
shanghai	151

Top 10 most common cities

city	train images	test images
london	1509	168
new york city	1188	132
chicago	818	91
toronto	469	52
sydney	183	20
los angeles	182	19
tokyo	172	19
philadelphia	157	18
houston	157	16
shanghai	136	15

wait...

or get a fast gpu


Rogier van der Geer -- GoDataDriven

London

Rogier van der Geer -- GoDataDriven

Rogier van der Geer -- GoDataDriven

Sydney

Toronto

Rogier van der Geer -- GoDataDriven

Los Angeles

Chicago


Rogier van der Geer -- GoDataDriven

Philadelphia

Rogier van der Geer -- GoDataDriven

Tokyo

Rogier van der Geer -- GoDataDriven

Rogier van der Geer -- GoDataDriven

Houston

Shanghai

Chicago


~~Chicago~~
what?

More mistagged images

Train set

Test set

Plan

Assign photographers to train/test splits

city	train images	test images	train photographers	test photographers
london	1509	168	161	18
new york city	1188	132	253	26
chicago	818	91	170	19
toronto	469	52	90	11
sydney	183	20	54	7
los angeles	182	19	50	5
tokyo	172	19	37	4
philadelphia	157	18	30	4
houston	157	16	24	3
shanghai	136	15	38	4

wait...

Result

- Awful performance:
  - train: ~ 90% accuracy
  - test: ~ 50% accuracy
  - Very overtrained!
- Too few photographers per city
- Too many mistagged photos

Another Plan

Other plan

1. Build a model:
   — Classes skyline and no skyline
2. Train on all data
   — Labels: has `skyline` tag or not
3. Make predictions for all data
4. Only use data with positive prediction

wait...

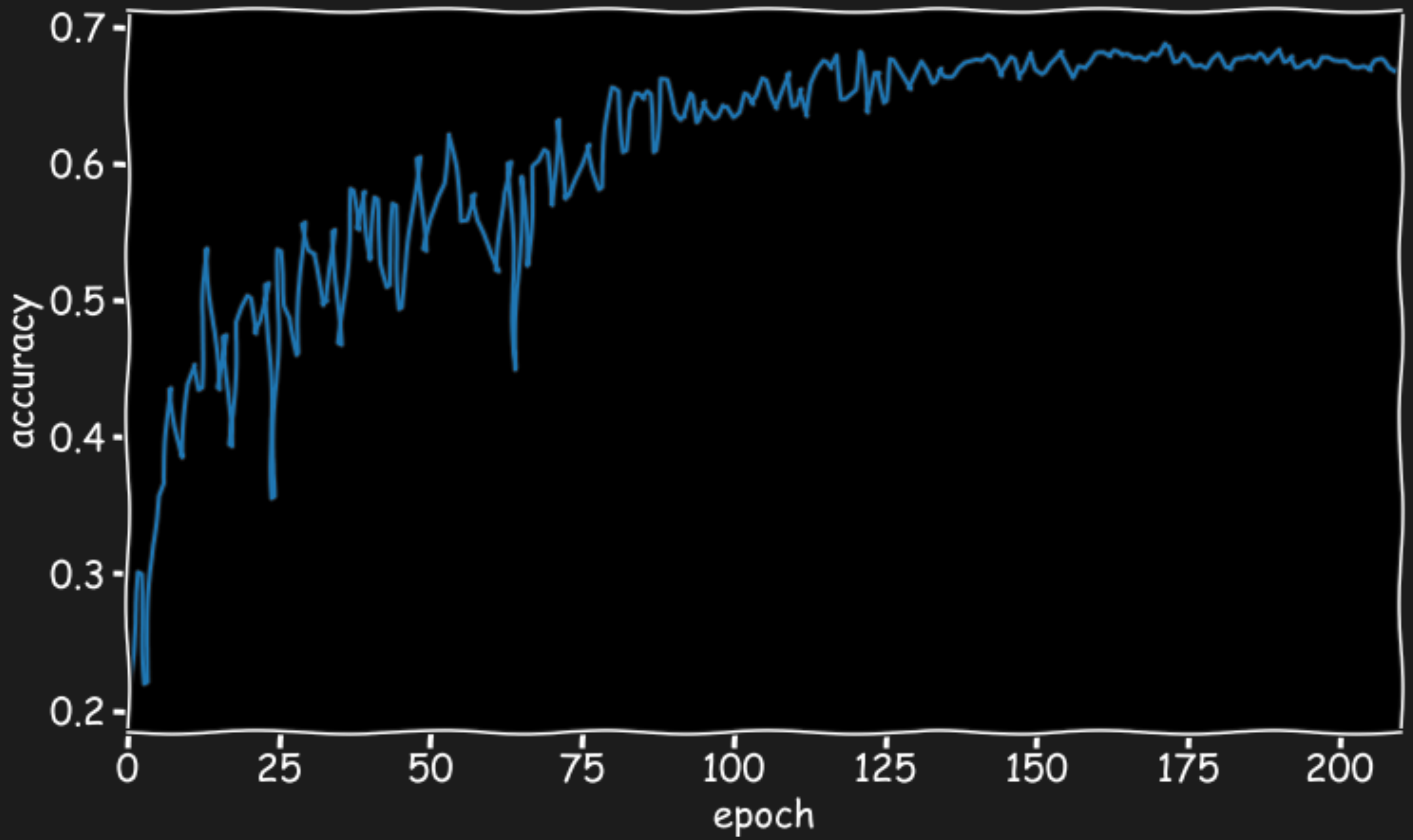
Result

prediction:	no skyline	skyline
no tag	467452	1070
with tag	1181	6204

Now: re-create train/test split

wait...

Yet more results



Chicago

Los Angeles

prediction: ~~New York City~~, label: Philadelphia

prediction: ~~London~~, label: Toronto

prediction: ~~Philadelphia~~, label: Shanghai

Final remarks

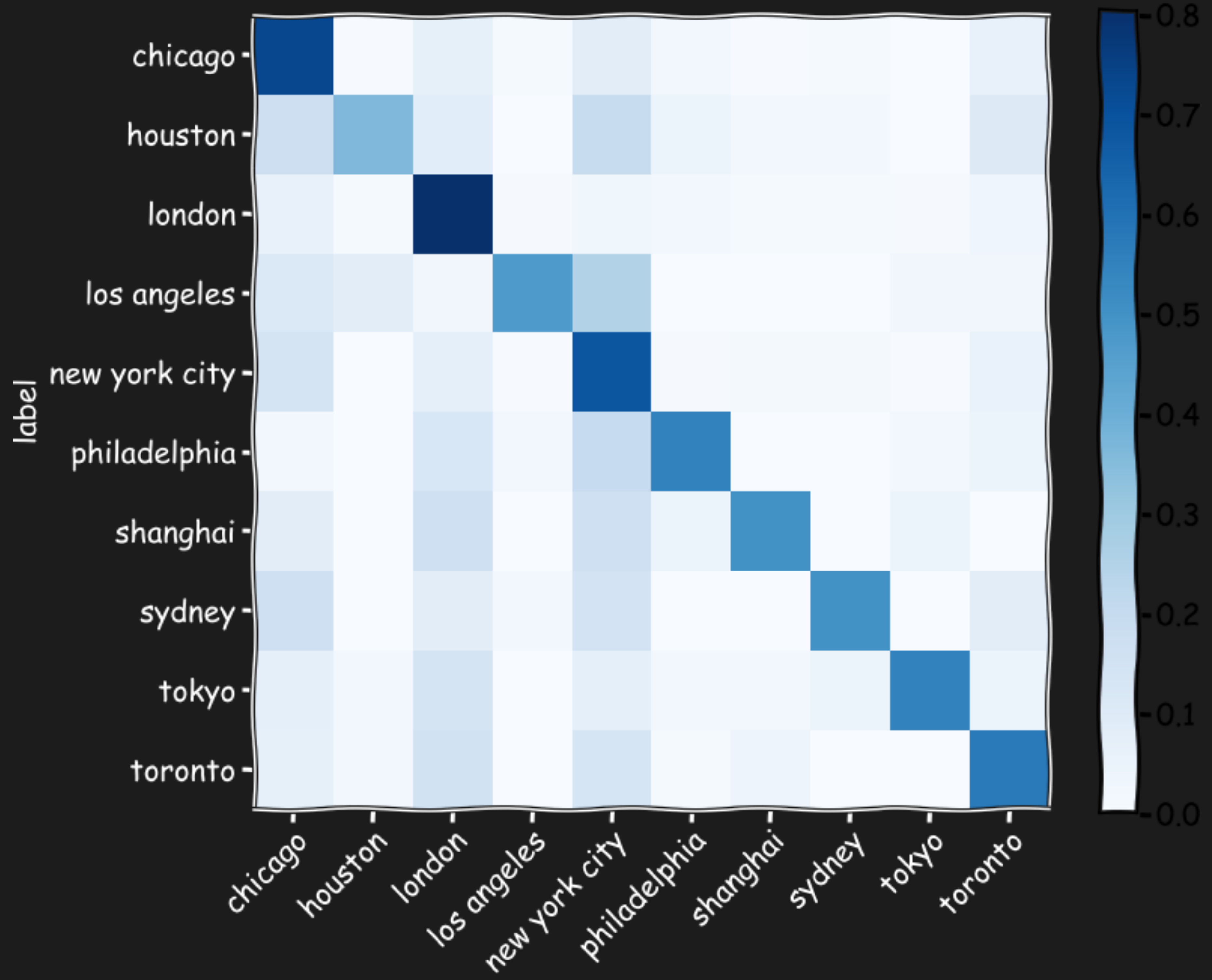
- Training an image classifier is not that difficult
- Pytorch is fun!
- Clean data is more important than a better model

Thank you!

https://gitlab.com/rogiervandergeer/skylines

https://blog.godatadriven.com

Appendix


chicago
houston
london
los angeles
new york city
philadelphia
shanghai
sydney
tokyo
toronto
label
chicago
houston
london
los angeles
new york city
philadelphia
shanghai
sydney
tokyo
toronto
prediction
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0