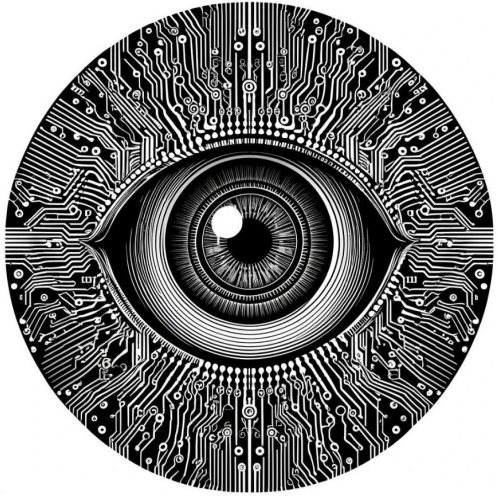
Workshop 1 - Foundations of Computer Vision



Antonio Rueda-Toicen

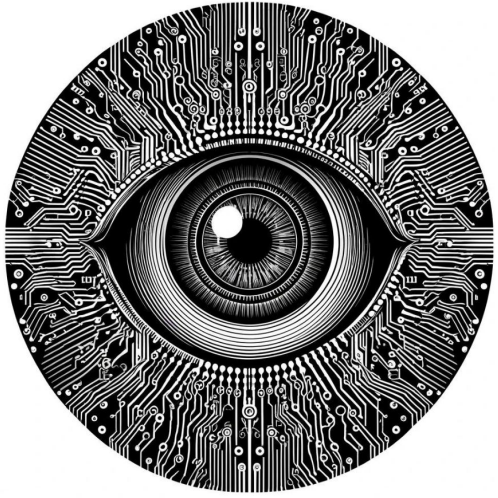
Agenda

- Common tasks in computer vision
- Digital image representation in PIL and NumPy
- Representing images as PyTorch tensors

Notebook

- **Digital Image Representation (Google Colab notebook)**

Common Tasks in Computer Vision



VOXEL51

Learning goals

- Understand what computer vision is
- Differentiate between discriminative and generative tasks

What is computer vision?

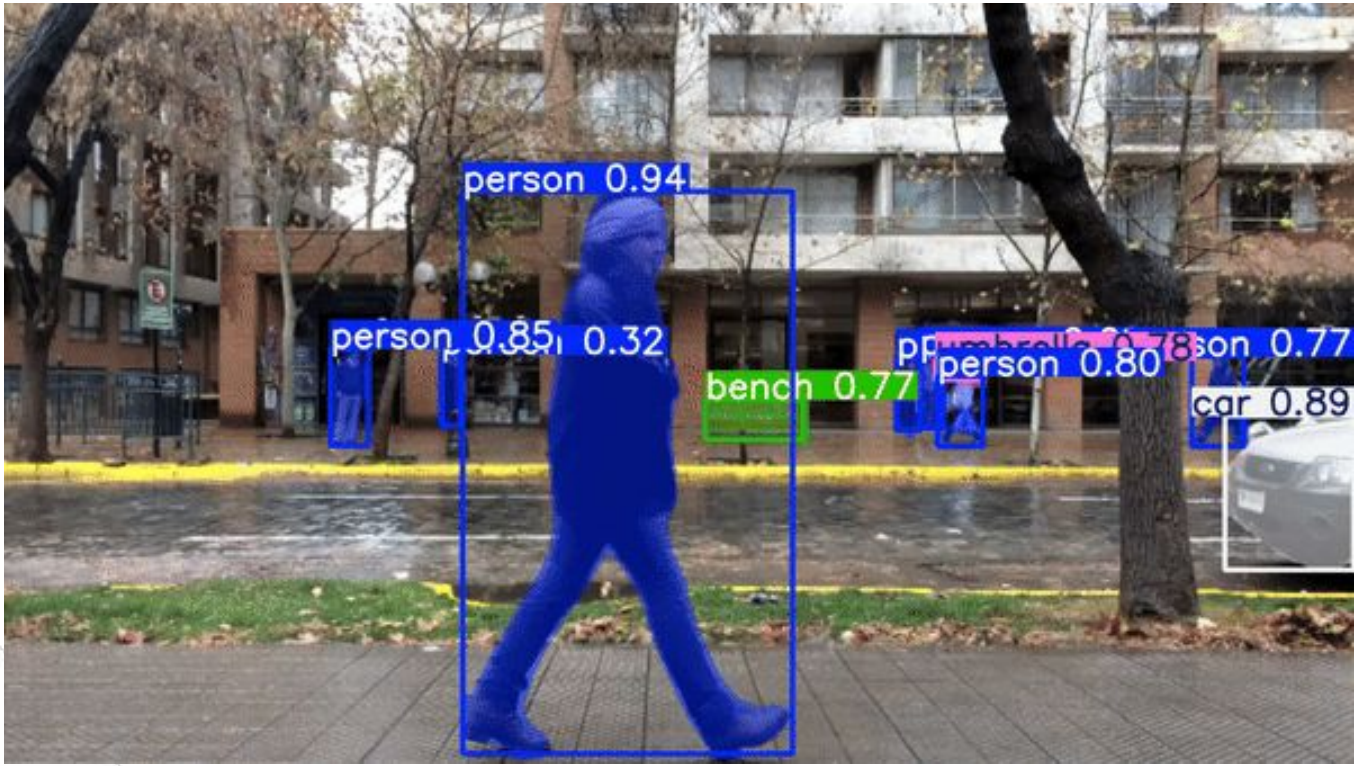


Image from https://learnopencv.com/yolo11/

Human vs machine perspective

Image source

What a human sees

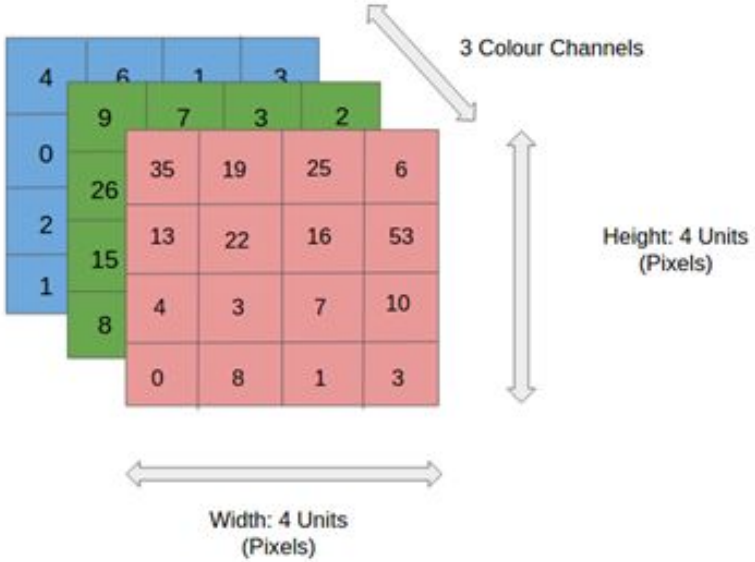


Image source

What the computer ‘sees’

Algorithms for computer vision

- Giving computers the ability to ‘understand’ the content of images

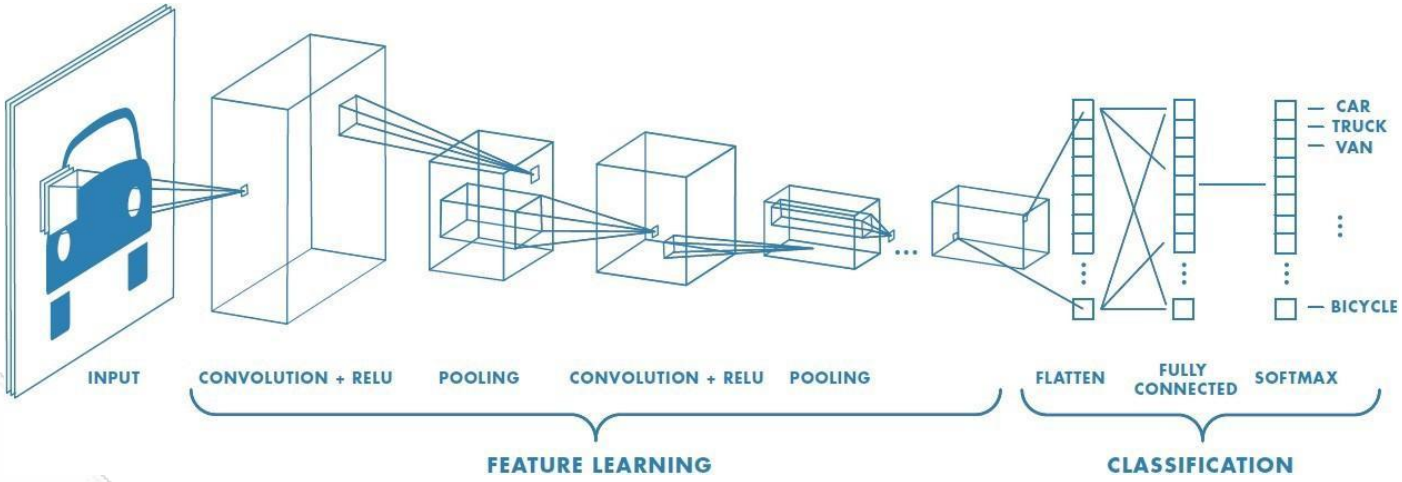
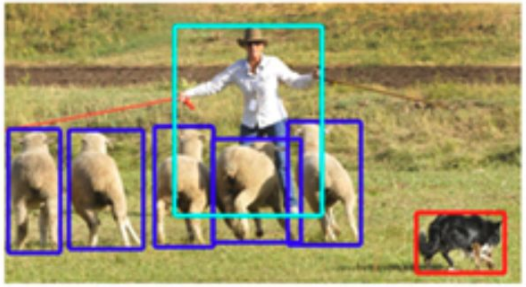


Image source

Discriminative computer vision

- “Which label or value can I assign to this image?”

Classification



Object Detection

Instance Segmentation

Image from “Learning to Segment” by Piotr Dollar, 2017

Discriminative computer vision

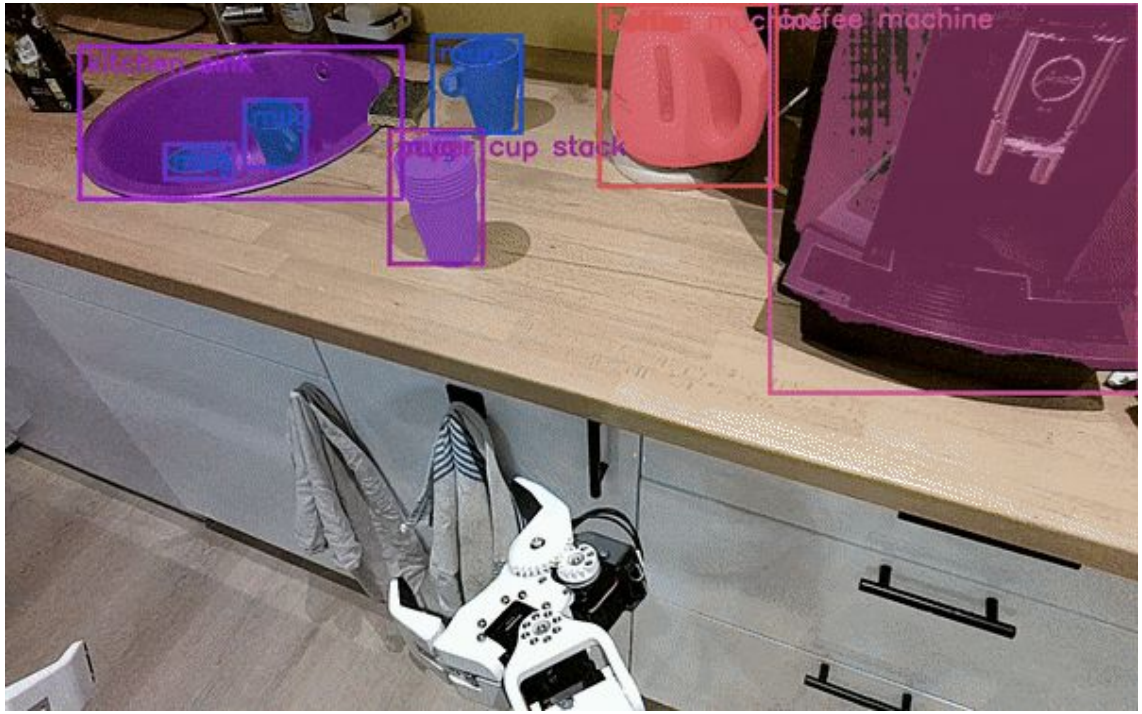


Image from https://github.com/pollen-robotics/pollen-vision

Generative computer vision

Prompt:

"Create an image of an astronaut riding a horse in pencil drawing style"

Image by OpenAI's Dall-E 3

Generative computer vision

Image from thispersondoesnotexist.com

Image from thiscitydoesnotexist.com

Integrating generative and discriminative computer vision

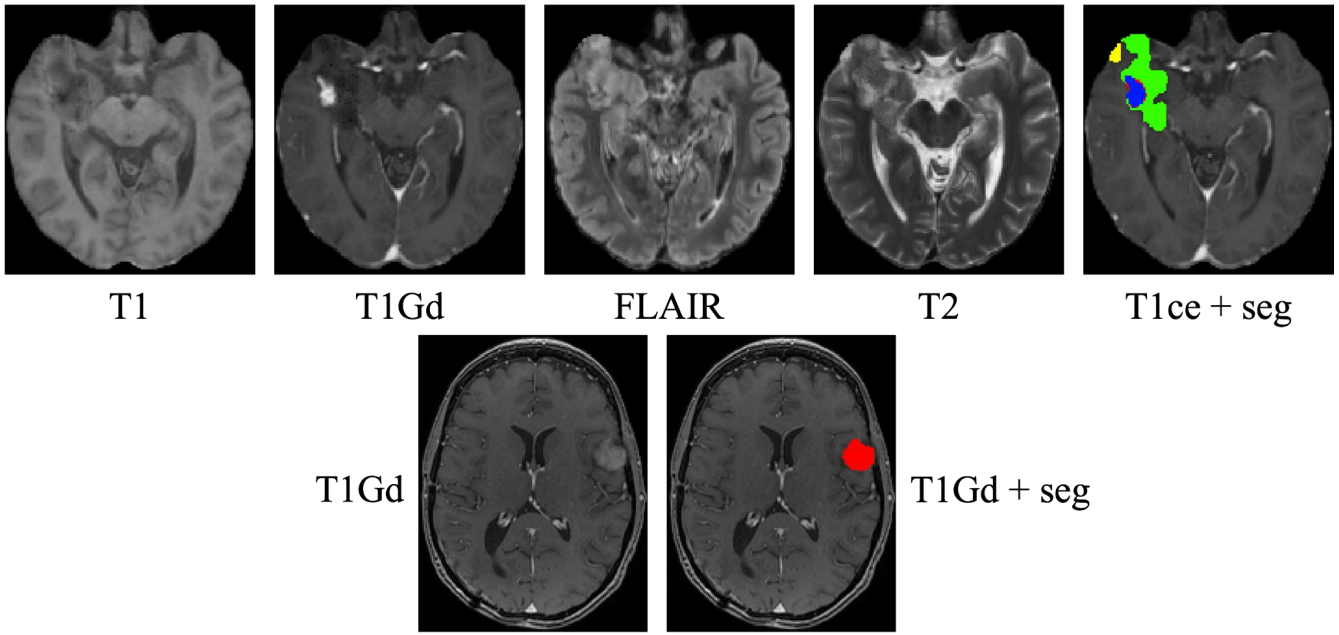


Image from Improved Multi-Task Brain Tumour Segmentation with Synthetic Data Augmentation

Summary

What is Computer Vision?

- Enabling machines to interpret visual data

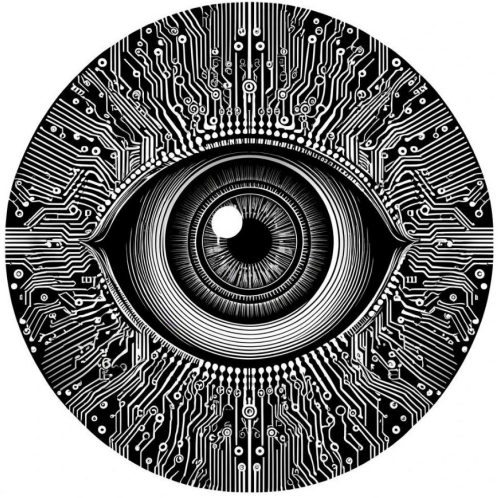
Discriminative Tasks

- Assigning scores or labels to images

Generative Tasks

- Creating images from prompts (text or other images)

Digital Images in PIL and NumPy

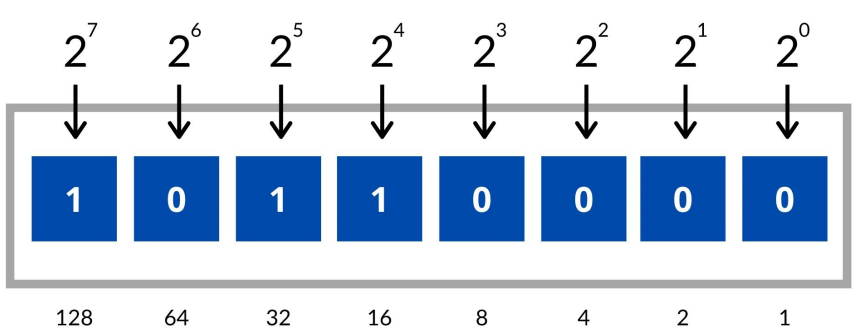


VOXEL51

Learning goals

- Understand image tensors as multidimensional arrays
- Gain familiarity with PIL images and NumPy arrays

The unsigned integer (`uint8`) datatype



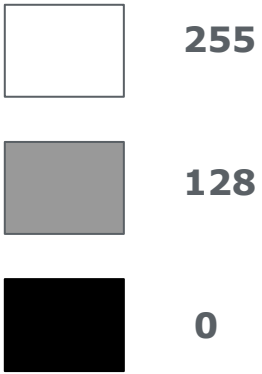
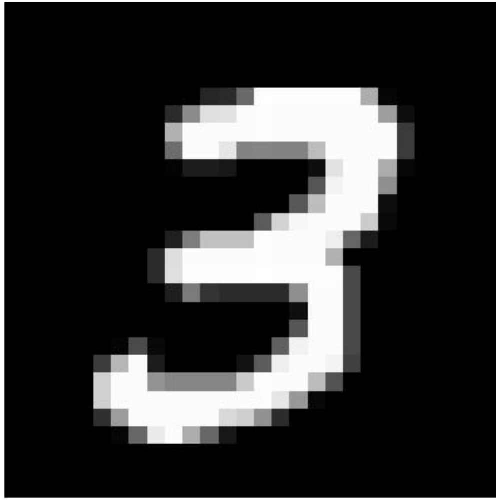
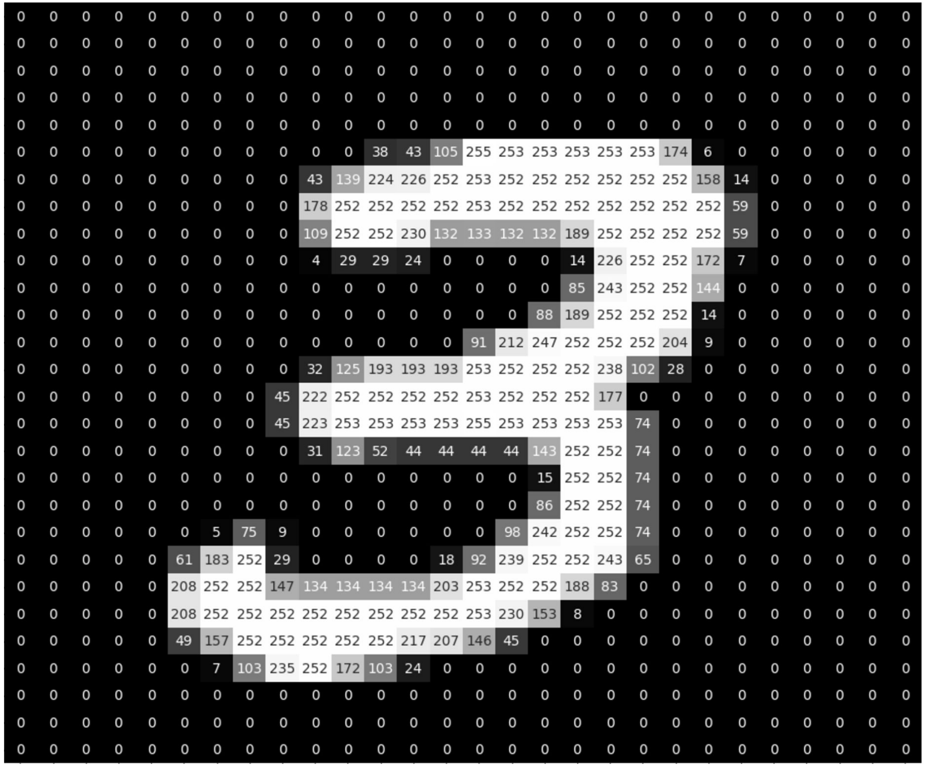


Image from [Why Are There 8 Bits in a Byte?]

Digital image representation for grayscale images



what we see



what the computer “sees”

RGB images as “tensors” (stacks of matrices)

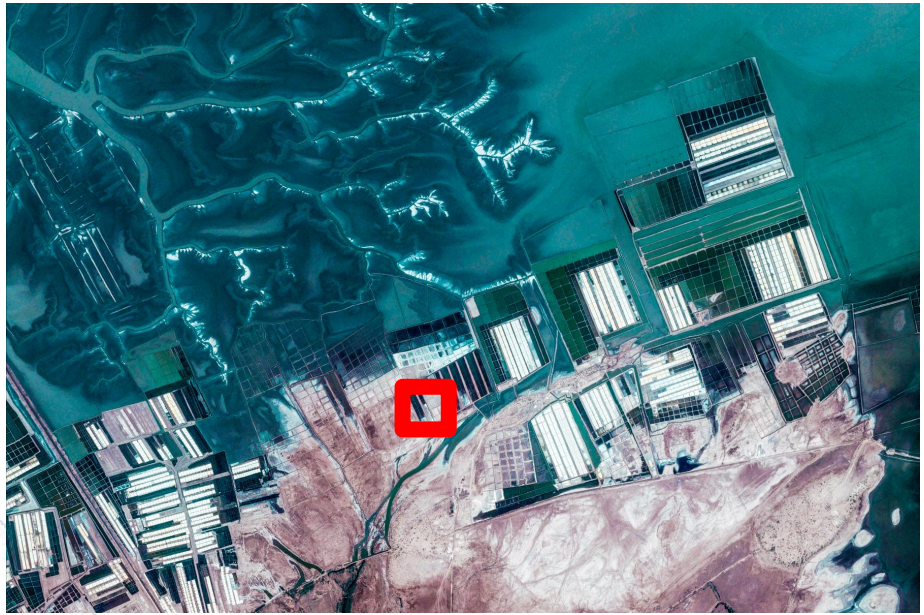


Image from Google Earth

What a human sees

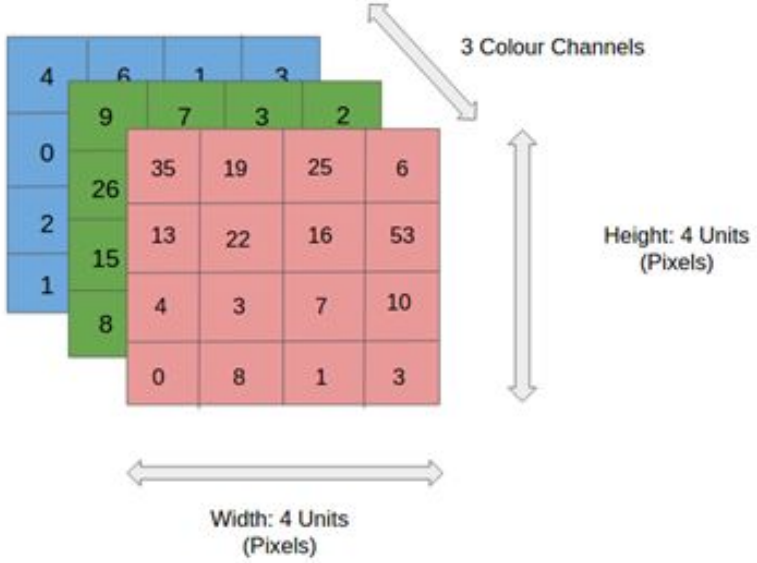
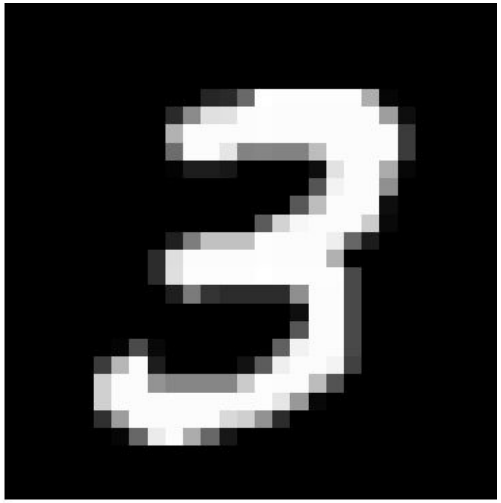


Image source

What the computer ‘sees’

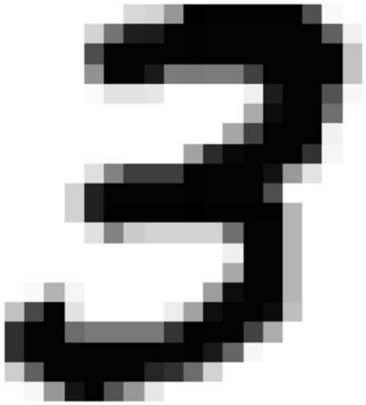
The Python Image Library (PIL)



real size is W = H = 28

```python
from PIL import Image

# Open image from file
pil_image = Image.open("path_to_image.jpg")

# Check the mode of the image, 'L' is for grayscale
# pil_image.get
print(pil_image.mode)

# Will print (28, 28), width first, height second
print(pil_image.size)

# Convert a NumPy ndarray to PIL
pil_image = Image.fromarray(np_array)

# Save image to storage as a PNG file
pil_image.save("path_to_image.png")

# Convert to numpy and upscale the image for visualization
import matplotlib.pyplot as plt
plt.imshow(pil_image)
```

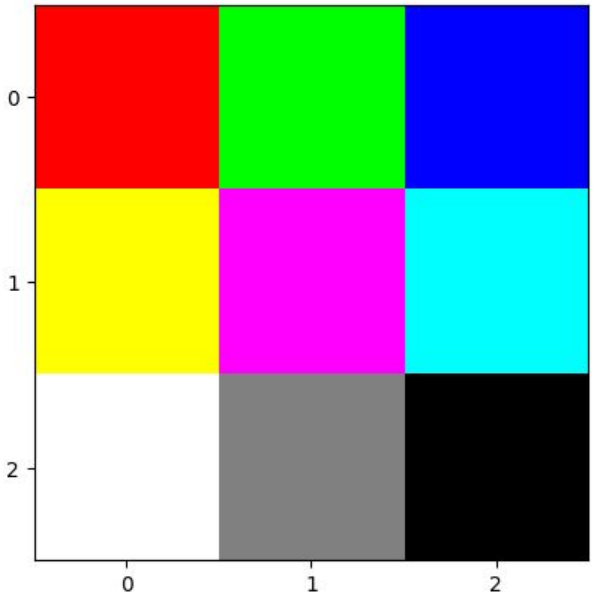
NumPy's N-dimensional array (`ndarray`)



```python
# Convert a PIL image to a Numpy array
np_array = np.array(pil_image)

# Check the shape of the array, prints (28, 28)
print(np_array.shape)

# Prints the data type, prints np.uint8
print(np_array.dtype)

# 255 minus current pixel values with NumPy's broadcasting
np_array_neg = 255 - np_array

# Show upscaled version on matplotlib
plt.imshow(np_array_neg)
```

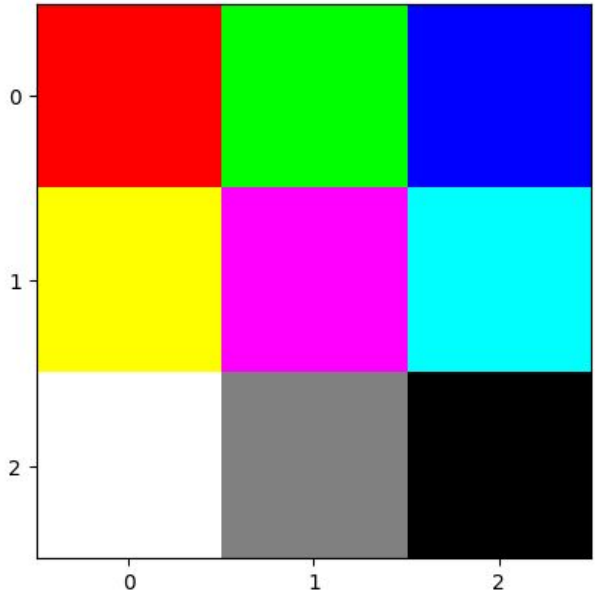
Red Green Blue (RGB) Images in NumPy

```
import numpy as np

# Create a3x3 RGB image array
rgb_image = np.array([
   # First row of pixels
   [[255, 0, 0],     # Pure red pixel
    [0, 255, 0],     # Pure green pixel
    [0, 0, 255]],    # Pure blue pixel

   # Second row of pixels
   [[255, 255, 0],  # Yellow pixel (red + green)
    [255, 0, 255],  # Magenta pixel (red + blue)
    [0, 255, 255]], # Cyan pixel (green + blue)

   # Third row of pixels
   [[255, 255, 255],# White pixel (all colors maximum)
    [128, 128, 128],# Gray pixel (all colors at half intensity)
    [0, 0, 0]]      # Black pixel (all colors minimum)
])
```

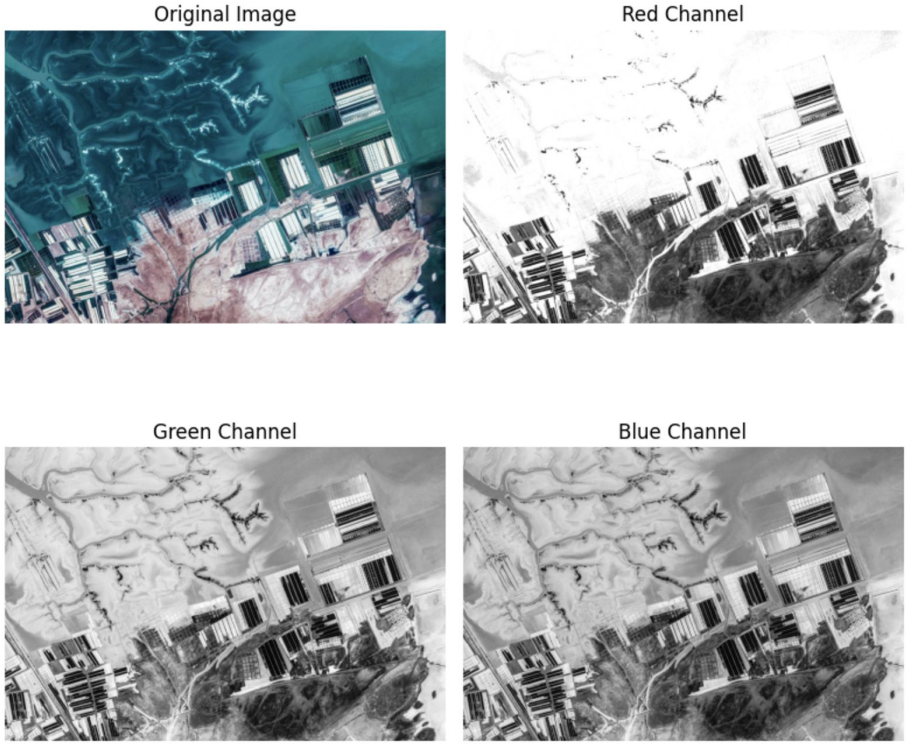


Indexing values in NumPy

```
# We select a row and a column
# Indexing starts at 0, as in standard Python
# We index first with height, then width
pixel = rgb_image[2, 0]

# Will show [255, 255, 255]
# Which index should we use to print magenta's values?
print(f"RGB values: {pixel}")
```

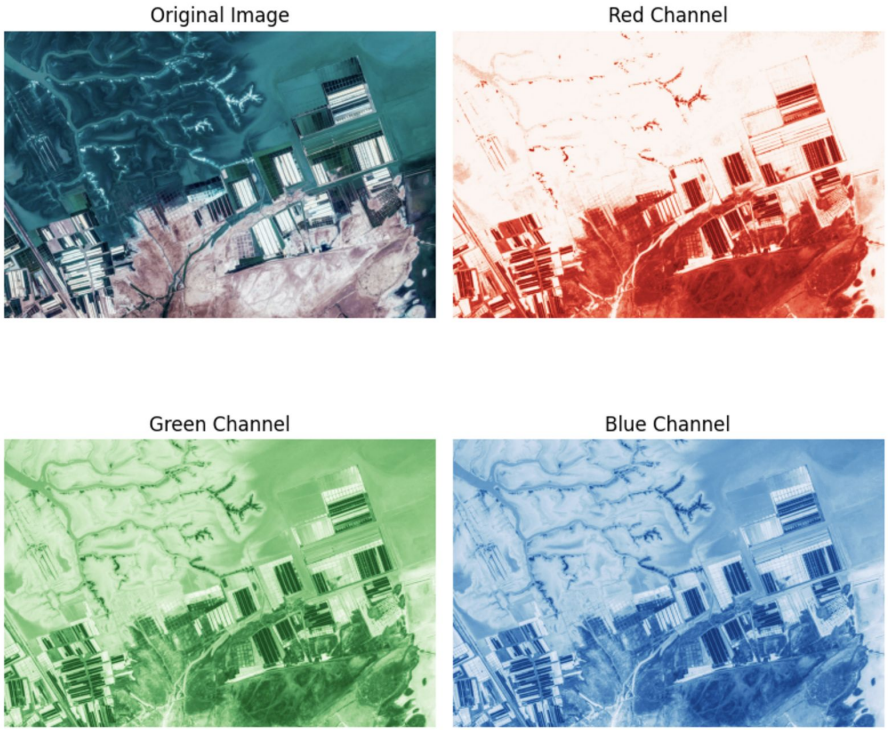


Intensities for RGB images in NumPy




```python
import maplotlib.pyplot as plt

# Intensity image from the R channel
plt.imshow(rgb_np_array[:, :, 0])

# Intensity image from the G channel
plt.imshow(rgb_np_array[:, :, 1])

# Intensity image from the B channel
plt.imshow(rgb_np_array[:, :, 2])
```

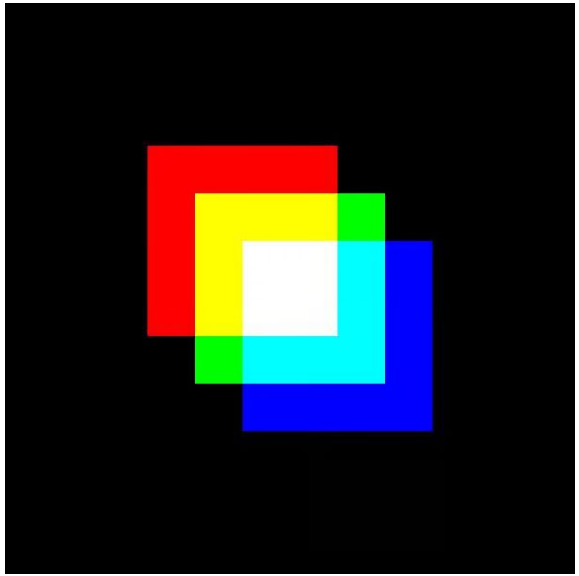
False colors for intensity images




```
import skimage.io as io

# Converts from JPEG to NumPy array
rgb_np_array =
io.imread("gujarat_image_url.jpeg")

# Images in NumPy are saved as
# Height (H), Width(W), Channels (C)
# Prints (1200, 1800, 3)
print(rgb_np_array.shape)

# Prints uint8
print(np_array.dtype)

# Intensity image with Red colormap
plt.imshow(rgb_np_array[:, :, 0], cmap='Reds')
plt.imshow(rgb_np_array[:, :, 1], cmap='Greens')
plt.imshow(rgb_np_array[:, :, 2], cmap='Blues')
```

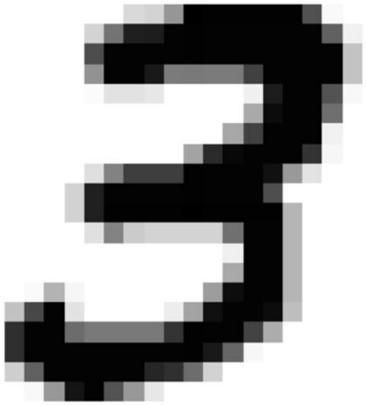
Image arithmetic in NumPy

```
import numpy as np
from PIL import Image

# Create arrays of of zeros
red = np.zeros((600, 600))
green = np.zeros((600, 600))
blue = np.zeros((600, 600))

# Set sections to maximum intensity
red[150:350, 150:350] = 255
green[200:400, 200:400] = 255
blue[250:450, 250:450] = 255

red_img = Image.fromarray(red).convert("L")
green_img = Image.fromarray(green).convert("L")
blue_img = Image.fromarray((blue)).convert("L")

# Merge channels
square_img = Image.merge("RGB", (red_img, green_img, blue_img))
square_img.show()
```



Code and image from source

The dimensions of PIL and NumPy arrays



NumPy image tensors follow the format: `[H, W, C]` where:

- `H` = height in pixels
- `W` = width in pixels
- `C` = channels (e.g., 1 for grayscale, 3 for RGB)

PIL image tensors follow the format `[W, H]` with mode

- mode is `"L"` for grayscale images
- mode is `"RGB"` for color images

Summary

Digital images are represented as numeric pixel values

- uint8 is memory-efficient and commonly used to store images

We use different Python libraries for image processing

- PIL for image loading and saving to JPEG or PNG
- NumPy for numerical operations and visualization with matplotlib

PIL and NumPy use different formats to represent images

- Height, width, and number of channels is the standard for NumPy
- Width, height and mode is the standard for PIL

References

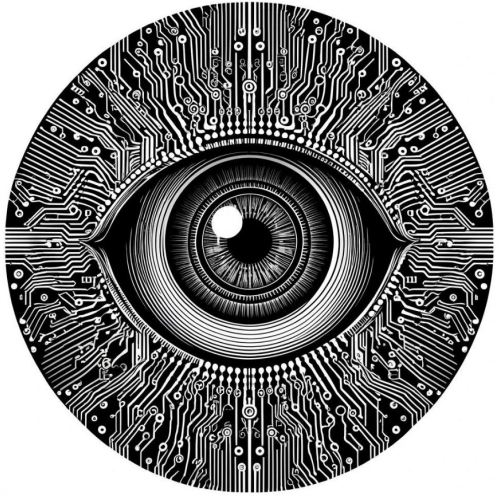
NumPy's N-dimensional array

- https://numpy.org/doc/2.1/reference/arrays.ndarray.html

Intro to PIL and NumPy for image manipulation

- https://realpython.com/image-processing-with-the-python-pillow-library/

Image Tensors in PyTorch



HPI

VOXEL51

Learning goals

- Gain familiarity with `torch` tensors
- Understand the need for data type transformation and scaling
- Understand how to move `torch` tensors between devices

Images as tensors

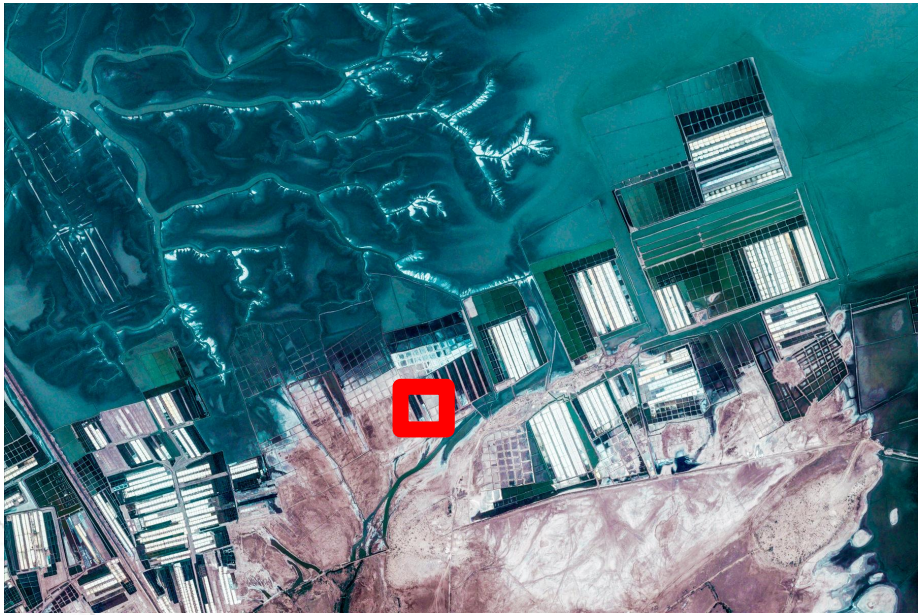


Image from Google Earth

What a human sees

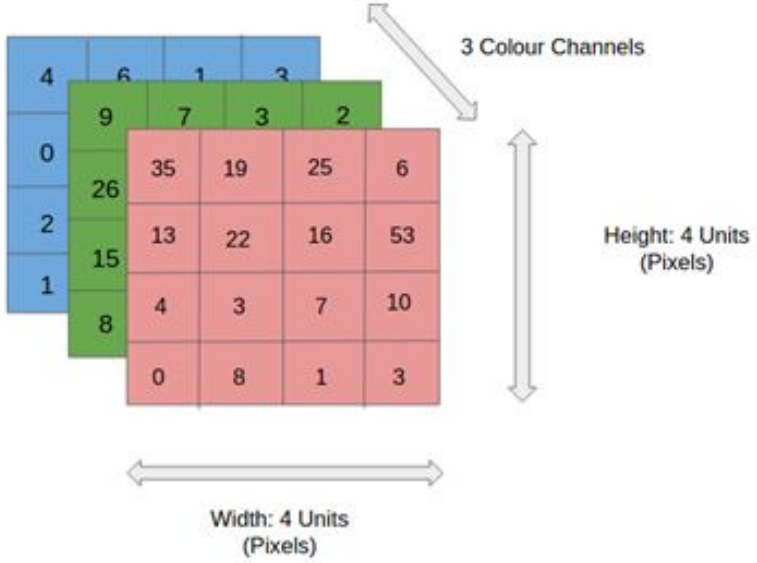
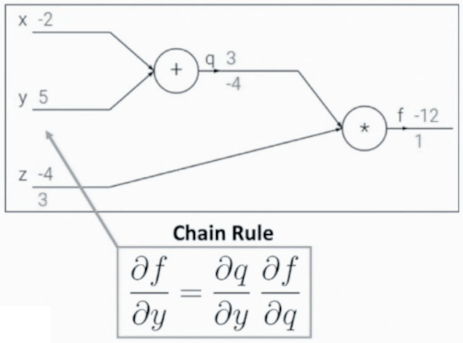


Image source

What the computer ‘sees’

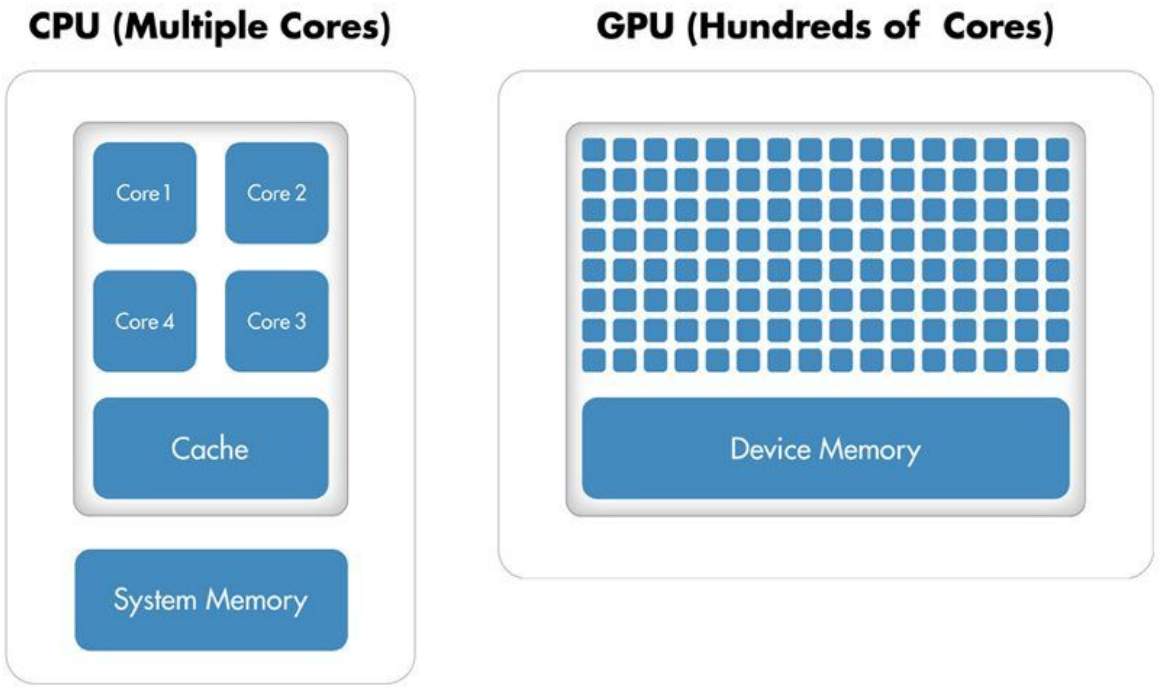
Why do we use PyTorch tensors?

1. Can run on the Graphics Processing Unit (GPU) or Tensor Processing Unit (TPU) speeding up training and inference

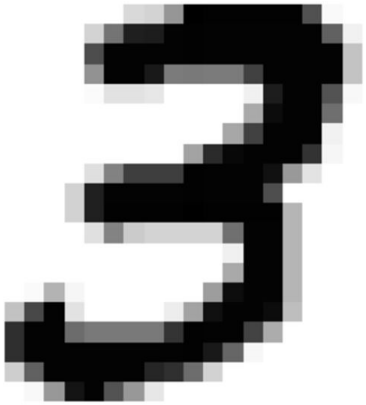


2. Do automatic gradient computation (autograd), enabling us to update the weights of models with automatically computed derivatives of the loss with respect to the weights

CPUs vs GPUs

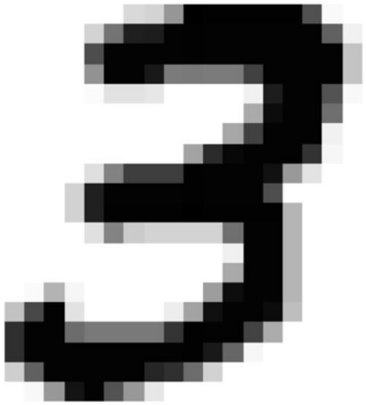

CPU (Multiple Cores)
Core 1
Core 2
Core 4
Core 3
Cache
System Memory
GPU (Hundreds of Cores)
Device Memory

Moving tensors to the GPU



```
import torch.nn as nn
# A model with a single linear layer
model = nn.Linear(in_features = 28 * 28,
                  out_features = 10)

device = 'cuda' if torch.cuda.is_available() else 'cpu'
# Move the model to the GPU
model = model.to(device)
torch_tensor_gray.to(device)

# Get prediction scores aka 'logits' (in GPU)
scores = model(torch_tensor_gray)
```

Moving tensors to the CPU for inspection/visualization



```
# Get prediction scores aka 'logits' (in GPU)

scores = model(torch_tensor_gray)


# Fails, scores are in the GPU

plt.show(scores)


# Moves a copy of the tensor the cpu
# scores.cpu() is acceptable too
plt.show(scores.to('cpu'))
```

Speedup on matrix multiplication

```
Running matrix multiplication benchmarks...

Device: Tesla P100-PCIE-16GB
+--------------+-----------------+-----------------+-----------+
| Matrix Size  | CPU Time (s)    | GPU Time (s)    | Speedup   |
+==============+=================+=================+===========+
| 32x32        | 0.0000 ± 0.0000 | 0.0000 ± 0.0000 | 0.3x      |
+--------------+-----------------+-----------------+-----------+
| 2048x2048    | 0.0588 ± 0.0024 | 0.0026 ± 0.0001 | 23.0x     |
+--------------+-----------------+-----------------+-----------+
| 8192x8192    | 3.8277 ± 0.1194 | 0.1285 ± 0.0022 | 29.8x     |
+--------------+-----------------+-----------------+-----------+
```

Quick benchmark on Kaggle

Speedup on matrix multiplication

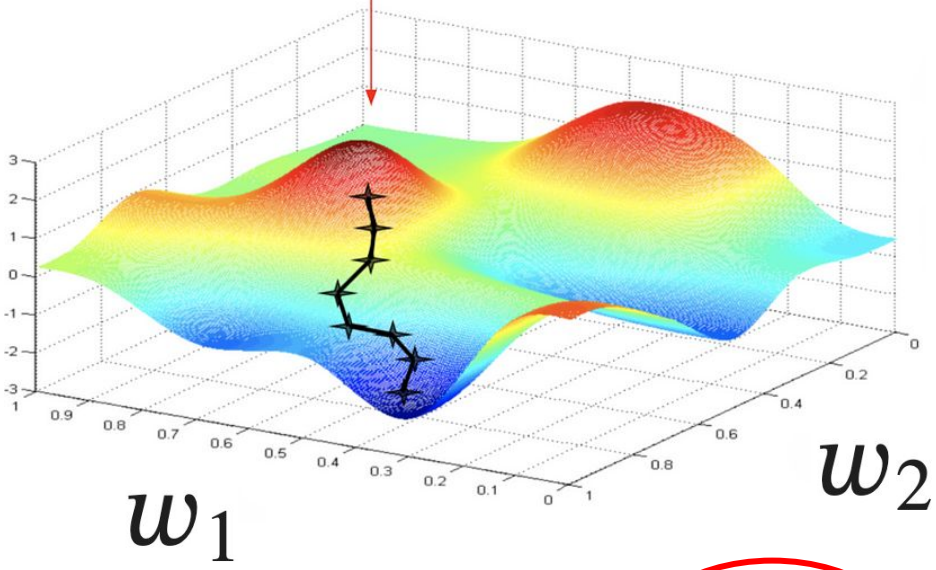
```
Running matrix multiplication benchmarks...

Device: Tesla P100-PCIE-16GB
+--------------+-----------------+-----------------+-----------+
| Matrix Size  | CPU Time (s)    | GPU Time (s)    | Speedup   |
+==============+=================+=================+===========+
| 32x32        | 0.0000 ± 0.0000 | 0.0000 ± 0.0000 | 0.3x      |
+--------------+-----------------+-----------------+-----------+
| 2048x2048    | 0.0588 ± 0.0024 | 0.0026 ± 0.0001 | 23.0x     |
+--------------+-----------------+-----------------+-----------+
| 8192x8192    | 3.8277 ± 0.1194 | 0.1285 ± 0.0022 | 29.8x     |
+--------------+-----------------+-----------------+-----------+
```

Quick benchmark on Kaggle

Gradient descent (we need derivatives to train models)

we start at a random position in the loss landscape

$L(w_1, w_2)$



w_1

w_2

$$w_{ij} = w_{ij} - (\text{learning rate} * \frac{dL}{dw_{ij}})$$

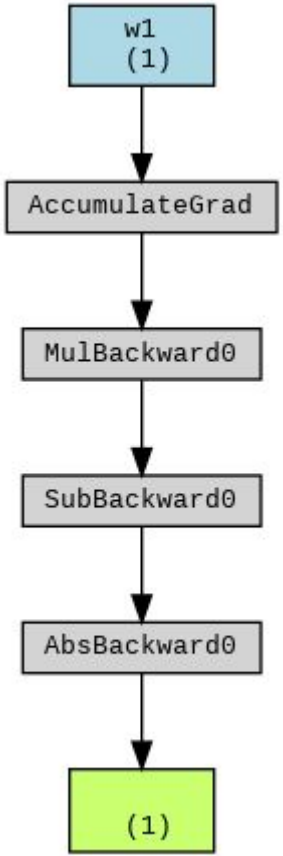
This is what gets produced "automatically" by PyTorch (autograd)

Autograd to compute derivatives on a minimal network

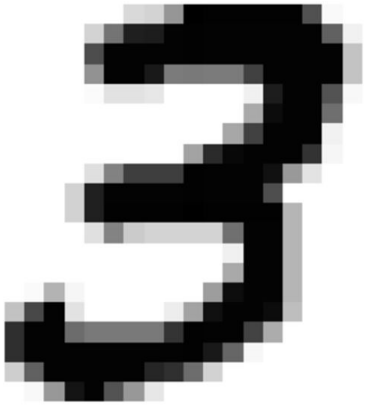
```python
import torch

# Create weight tensor with requires_grad=True
w1 = torch.tensor([0.3], requires_grad=True)

# Input features and ground truth do not require gradients (do not change)
# Suppose that this is the height of a person
x1 = torch.tensor([183.0], requires_grad=False)

# And this is the age
y = torch.tensor([35.0], requires_grad=False)

# Computing the loss creates a computation graph for w1, to calculate the
derivative of the loss with respect to the weight
L = torch.abs(w1 * x1 - y)

# Will show requires_grad = True
print(f"loss = {L.item()}, requires_grad = {L.requires_grad}")
# Compute gradients
L.backward()
```

$$L = |w_1 x_1 - y|$$

$$\frac{\partial L}{\partial w_1} = \begin{cases} x_1 & \text{if } w_1 x_1 - y \geq 0, \\ -x_1 & \text{if } w_1 x_1 - y < 0. \end{cases}$$



The dimensions of a `torch` image tensor



`torch` image tensors follow the format: `[N, C, H, W]` where:

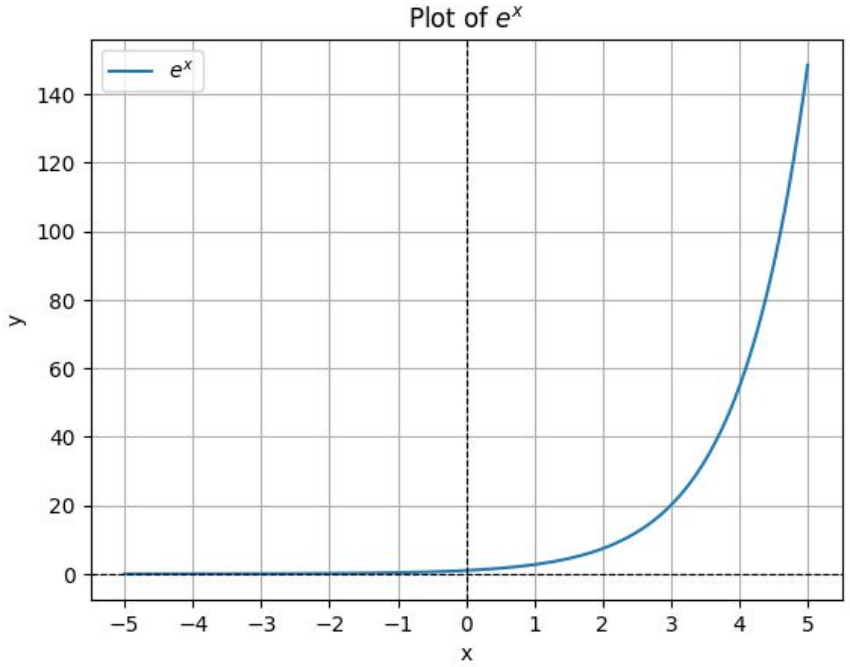
- `N` = batch size (number of images)
- `C` = channels (e.g., 1 for grayscale, 3 for RGB)
- `H` = height in pixels
- `W` = width in pixels

```
# Tensors shown to neural networks include the batch size
print(torch_tensor_gray.unsqueeze(0).shape)
# prints torch.Size([1, 1, 28, 28])
print(torch_tensor_gray.dim())
# prints 3, the “rank” (# of dimensions)
```

PyTorch (aka `torch`) tensors

```
# ToImage() converts NumPy arrays with H, W, C format to torch's C, W, H
transform = transforms.ToImage()
image_tensor = transform(np_array)

# Will print 3, 1200, 1800
print(image_tensor.shape)

# Will print 1200, 1800, 3
print(image_tensor.permute(1, 2, 0).shape)

# Error, matplotlib expects NumPy format
plt.imshow(image_tensor)

# Channel order needs to permuted to use matplotlib, as it expects NumPy's channel order
plt.imshow(image_tensor.permute(1, 2, 0))
```

Numerical stability




```
original_values = torch.tensor([255, 204, 170], dtype=torch.uint8)

torch.exp(original_values)
```

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{for } i = 1, \ldots, n$$

Numerical stability example

```
# Let's simulate a single pixel intensity in RGB
original_values = torch.tensor([255, 204, 170], dtype=torch.uint8)
print(f"Original values: {original_values}")

# Convert to float and scale
float_values = original_values.float() / 255
print(f"Scaled values: {float_values}")

# Example of multiplication stability
print(f"Original exp: {torch.exp(original_values)}")  # Very large! inf!
print(f"Scaled exp: {torch.exp(float_values)}")  # Stay small, bitte
```

Numerical stability

```
Original values: tensor([255, 204, 170], dtype=torch.uint8)
Scaled values: tensor([1.0000, 0.8000, 0.6667])
Original exp: tensor([inf, inf, inf])
Scaled exp: tensor([2.7183, 2.2255, 1.9477])
```

`torch` tensors with rescaled floating point values

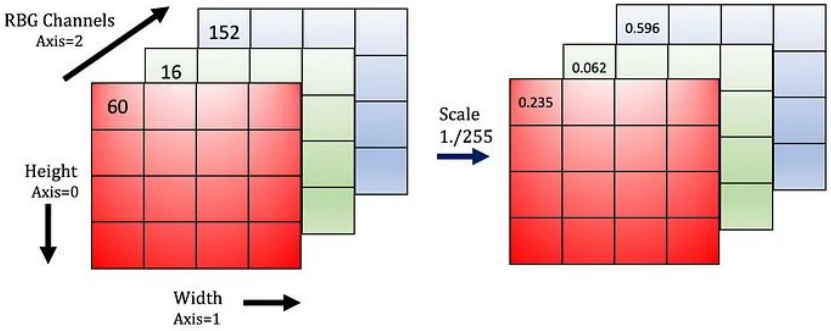


Image from source


```python
# We convert the input to float and rescale it to
# 0,1 to avoid overflow and get stable operations
transform = transforms.Compose([
       transforms.ToImage(),
      transforms.ToDType(torch.float32,  scale=True)])

# This transform would also work with a NumPy array
image_tensor = transform(pil_image)

# Prints (1.0, 0.0)
print(image_tensor.max(), image_tensor.min())
```

Summary

Tensors in PyTorch can run on the GPU and do automatic gradient computation

- We use `tensor_name.to(device)` to move tensors between CPU and GPU memory
- We use r`esult.backward()` to compute gradients on tensors with `requires_grad = True`

Common pitfall: different standards of tensor dimensions in NumPy vs PyTorch

- We use the B, C, H, W order in PyTorch; H, W, C in NumPy. The batch dimension corresponds to the number of tensors that we process at once on the device (GPU or CPU). The `toImage()` transform allows us to put the channel dimension on its proper position

Conversion to float and scaling tensors helps against numerical errors

- Operations like `torch.exp()` overflow without scaling

References

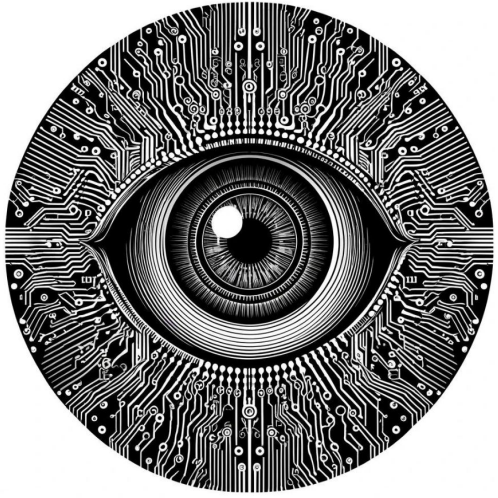
torch.Tensor.to

- https://pytorch.org/docs/stable/generated/torch.Tensor.to.html

ToImage

- https://pytorch.org/vision/0.19/generated/torchvision.transforms.v2.ToImage.html

Getting Started with FiftyOne



VOXEL51

Today we create a FiftyOne dataset

DATASET

Tags | Names | Classes | Description | ...

SAMPLE 0

ID | Filepath | Tags | Label | Field

SAMPLE 1

ID | Filepath | Tags | Label | Field

SAMPLE 2

ID | Filepath | Tags | Label | Field

GitHub Repository and YouTube Playlist

- Repository
- Playlist