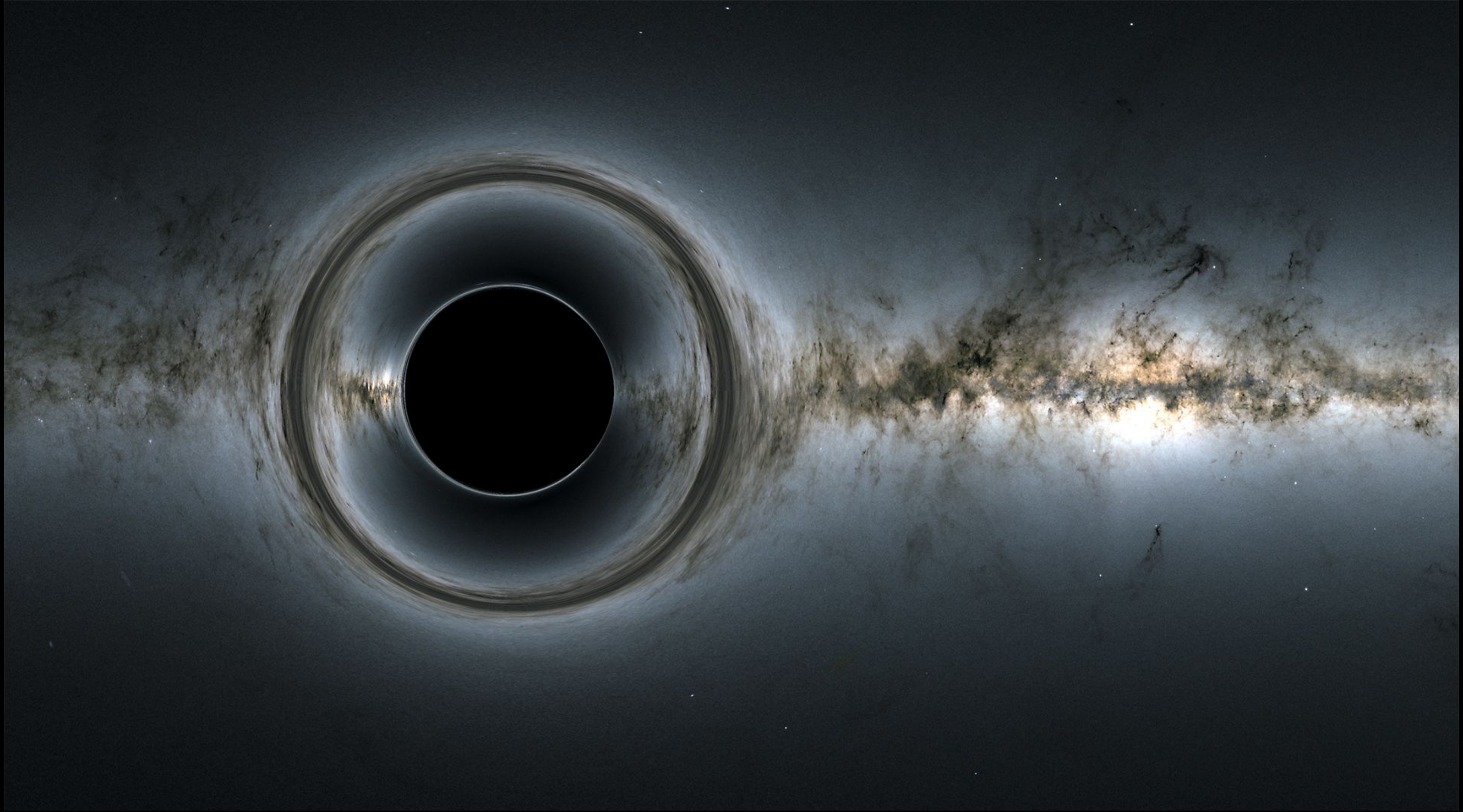
differentiable computer vision

an introduction to kornia

Edgar Riba

Open Source Vision Foundation - OpenCV.org
Computer Vision Center (CVC-UAB) - Institut de Robotica Industrial (CSIC-UPC)



Q: > Where is Classical Computer Vision in Deep Learning ?

A: > ...

Q: > Where is Classical Computer Vision in Deep Learning ?

A: > Black hole area

kornia

Open Source Differentiable Computer Vision Library for PyTorch

kornia

planet Kornia

wiki

+2400 stars

+260 forks

+50 contributors

Apache 2 Licence

THE APACHE SOFTWARE FOUNDATION

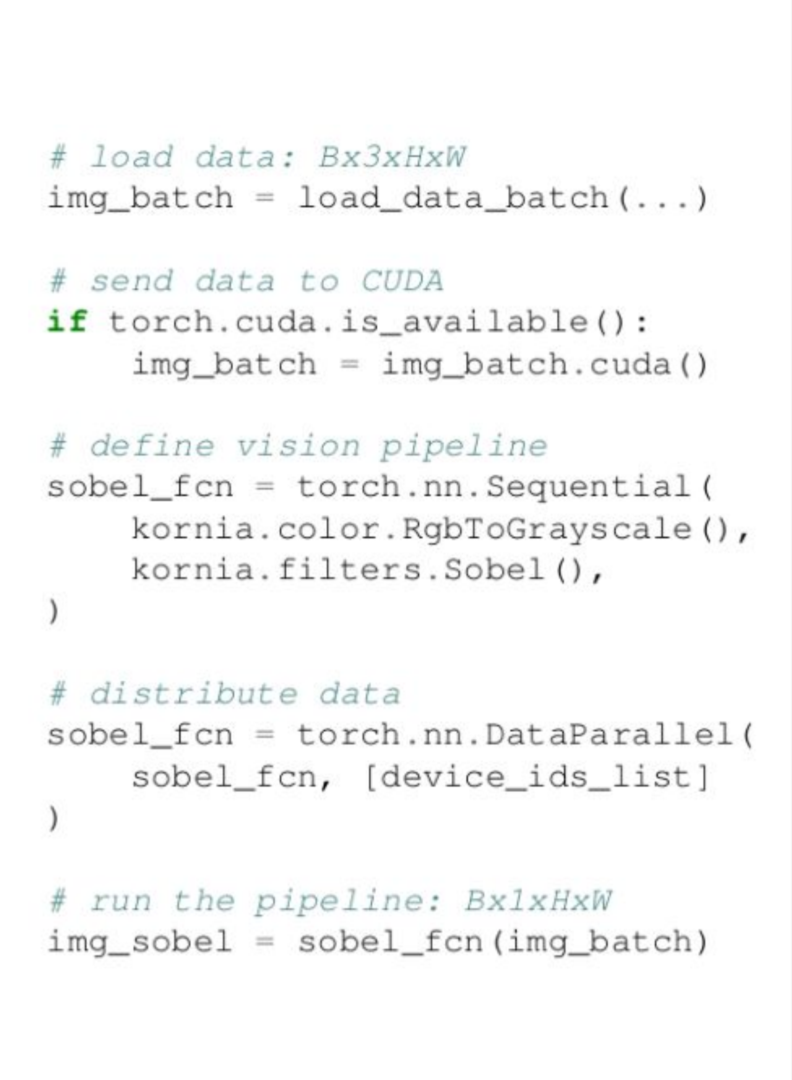
kornia

Core features

1. Differentiable
2. Transparent API
3. Parallel Programming
4. Distributed
5. Production → JIT

kornia

Core features



```
# load data: Bx3xHxW
img_batch = load_data_batch(...)

# send data to CUDA
if torch.cuda.is_available():
    img_batch = img_batch.cuda()

# define vision pipeline
sobel_fcn = torch.nn.Sequential(
    kornia.color.RgbToGrayscale(),
    kornia.filters.Sobel(),
)

# distribute data
sobel_fcn = torch.nn.DataParallel(
    sobel_fcn, [device_ids_list]
)

# run the pipeline: Bx1xHxW
img_sobel = sobel_fcn(img_batch)
```

kornia

Basic functionality

Data augmentation

Image enhancement

Color space conversions

2D feature detection

Image filtering

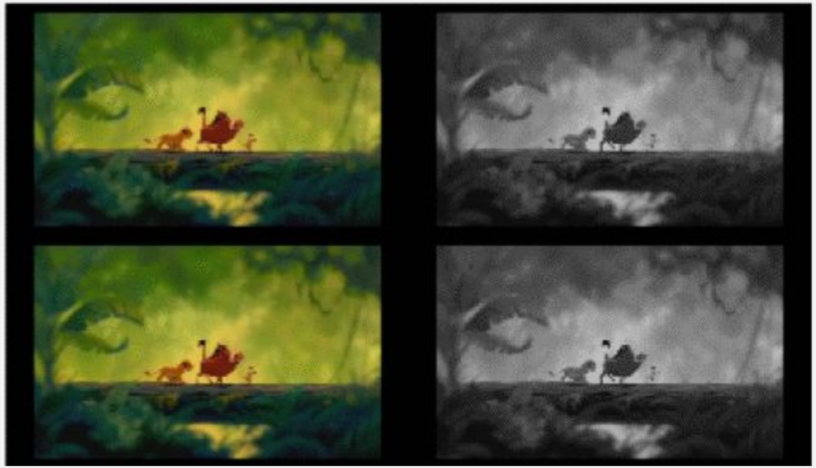
Edge detection

Geometric transformations

3D geometry

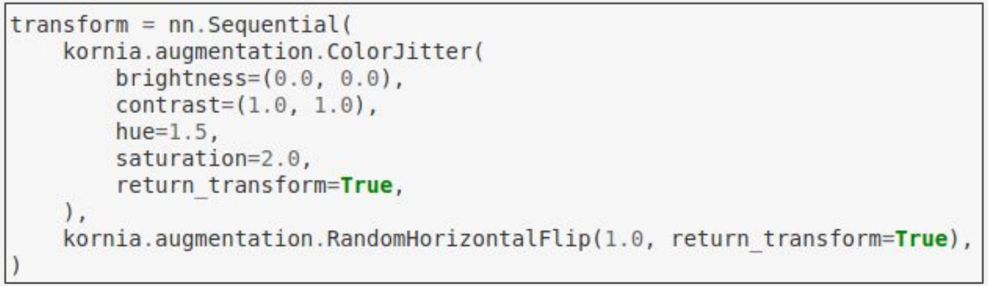
Vision loss functions

kornia



Data augmentation

- Random sampling using *torch.distributions*
- Compatible with torchvision
- Batched, GPU
- Return and chain spatial transforms



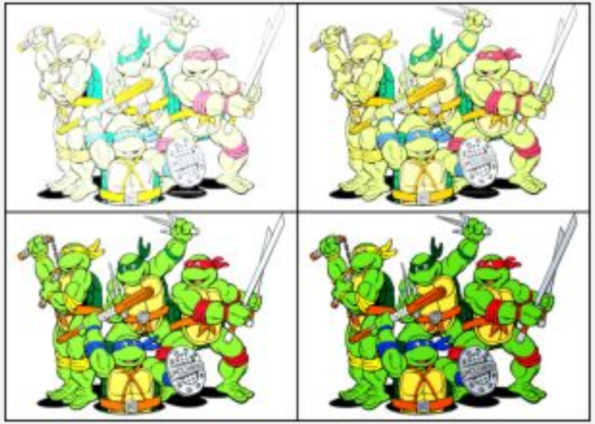
```
transform = nn.Sequential(
    kornia.augmentation.ColorJitter(
        brightness=(0.0, 0.0),
        contrast=(1.0, 1.0),
        hue=1.5,
        saturation=2.0,
        return_transform=True,
    ),
    kornia.augmentation.RandomHorizontalFlip(1.0, return_transform=True),
)
```

kornia

Image Enhancement

- Image tensors normalization
- ZCA mean/whiten
- Image Histogram 1d/2d
- contrast, brightness, gamma, hue, saturation

Gamma



kornia

Color space conversions

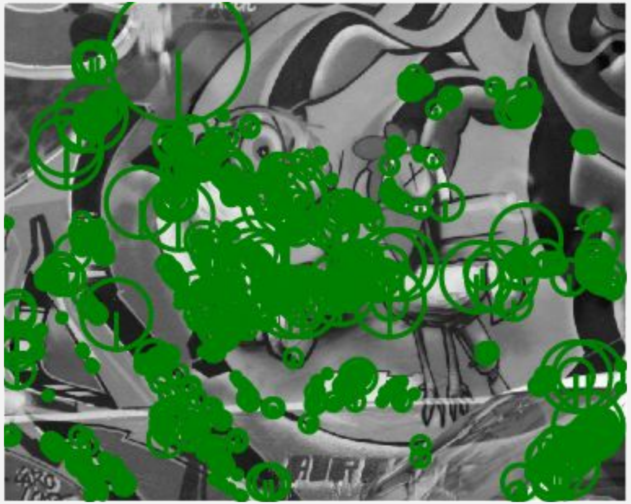
- RGB, RGBA, Grayscale
- HSV, HLS
- Luv, Lab
- XYZ, YCbCr, Yuv

RGB to Grayscale

kornia

2D feature detection

- Harris, Hessian, DoG
- Scale Space framework
- NMS , ConvSoftMax2d/3d
- Local Affine Frames (LAF)
- Differentiable SIFT, Deep descriptors

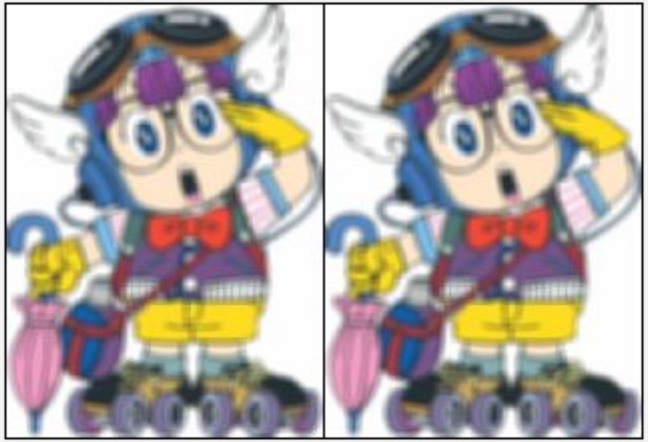


kornia

Image filtering

- Filter 2D / 3D
- Kernels API: gaussian, laplacian, sobel
- Blurring: median, box, gaussian, motion

Gaussian Blur

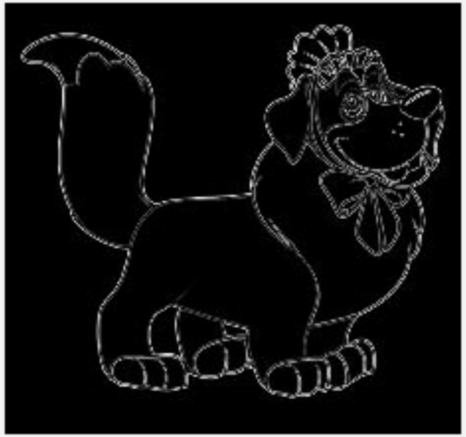


kornia

Edge detection

- Laplacian
- Sobel
- Spatial gradient 2d/3d

Sobel



kornia

Geometric transformations

- Rotate, translate, scale, shear, resize
- Gaussian pyramid, PyUp, PyrDown
- Crop: center crop, crop and resize
- Flip: horizontal/vertical, rot180

kornia

Geometric transformations

- Rotate, translate, scale, shear, resize
- Gaussian pyramid, PyUp, PyrDown
- Crop: center crop, crop and resize
- Flip: horizontal/vertical, rot180

kornia

Geometric transformations

- Rotate, translate, scale, shear, resize
- Gaussian pyramid, PyUp, PyrDown
- Crop: center crop, crop and resize
- Flip: horizontal/vertical, rot180

kornia

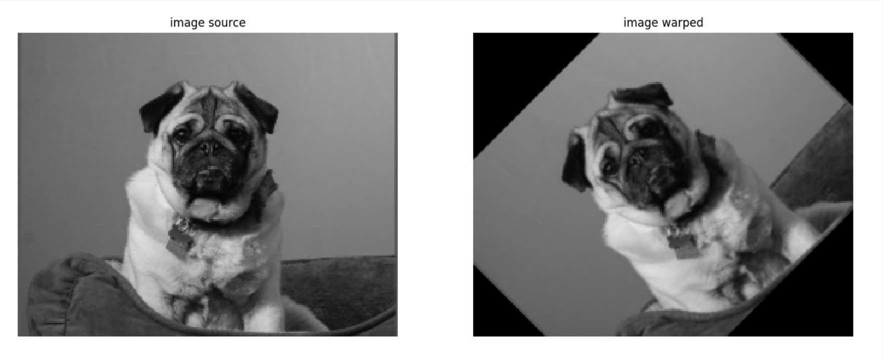
Geometric transformations

- Rotate, translate, scale, shear, resize
- Gaussian pyramid, PyUp, PyrDown
- Crop: center crop, crop and resize
- Flip: horizontal/vertical, rot180

kornia

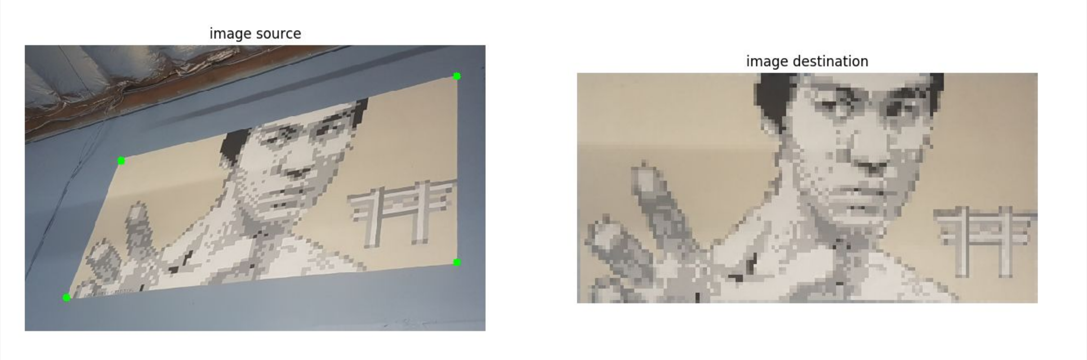
Geometric transformations

- warp_affine
- warp_perspective
- get_perspective_transform
- get_rotation_matrix2d



Warp Affine

Warp Perspective



kornia

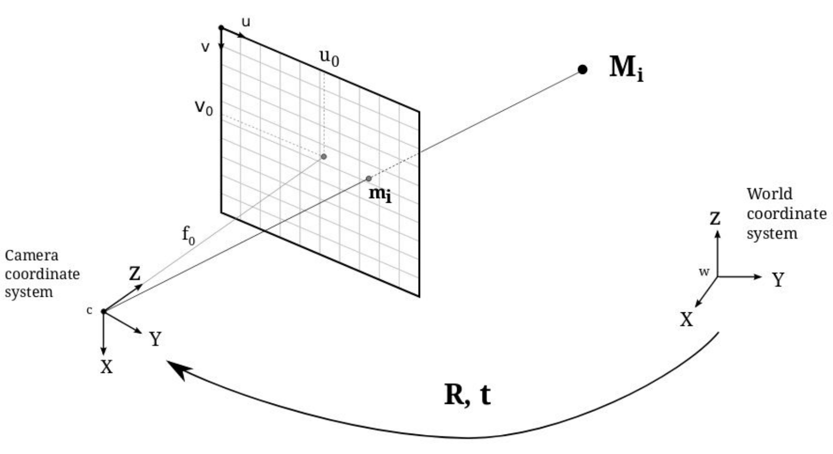
3D Geometry

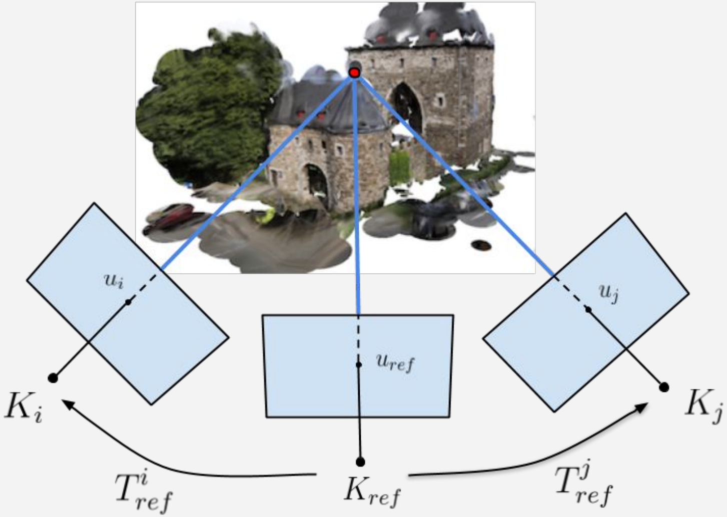
- Pinhole and perspective camera API

- Conversions: homogeneous, euclidean, rotation matrix, quaternion, axis-angle, normalize_coordinates

- Subpixel: conv/soft_softargmax2d/3d, conv_quad_interp3d

- Epipolar, Lie algebra and SfM utilities



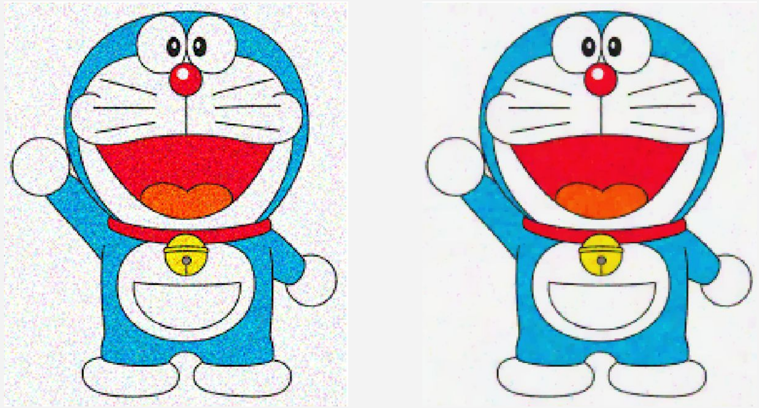


kornia

Vision loss functions

Specific Loss functions

- Image reconstruction
- Semantic segmentation
- Heatmaps



Total Variation

SSIM, PSNR

Focal Loss

...

kornia

development

1. **Easy to install**
2. Easy to use
3. OpenCV syntax
4. Ecosystem Integration

1. Easy to install

- Install from pip or source

From pip:

```
pip install kornia
```

From source:

```
python setup.py install
```

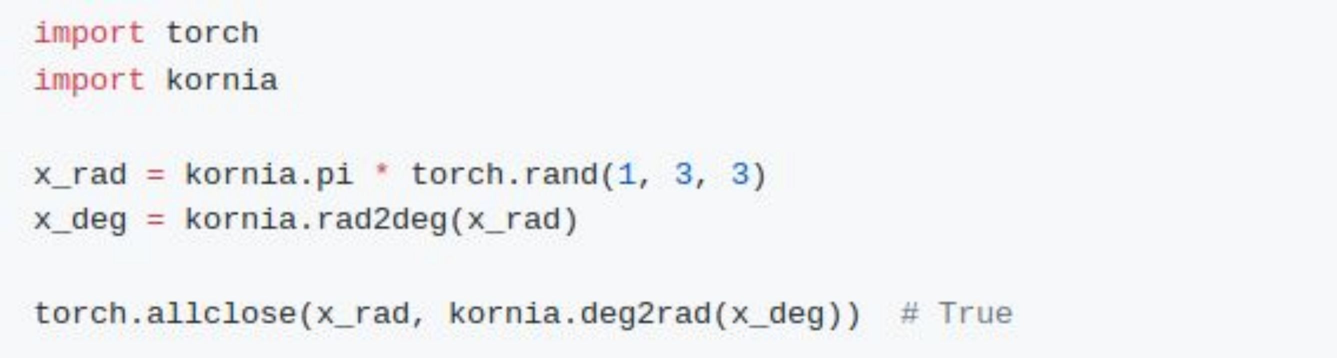
kornia is dependency-free - **ONLY PyTorch**

kornia
development

1. Easy to install
2. **Easy to use**
3. OpenCV syntax
4. Ecosystem Integration

2. Easy to import and use

- Import from any *Python >= 3.6* script
- Compatible with any *torch.Tensor* operator



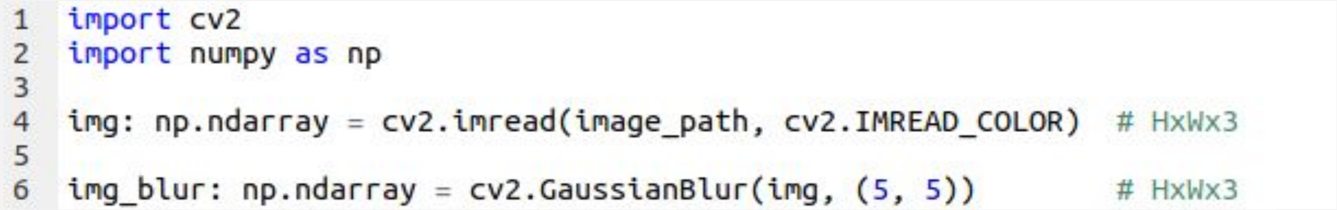
```python
import torch
import kornia

x_rad = kornia.pi * torch.rand(1, 3, 3)
x_deg = kornia.rad2deg(x_rad)

torch.allclose(x_rad, kornia.deg2rad(x_deg))  # True
```

kornia

development

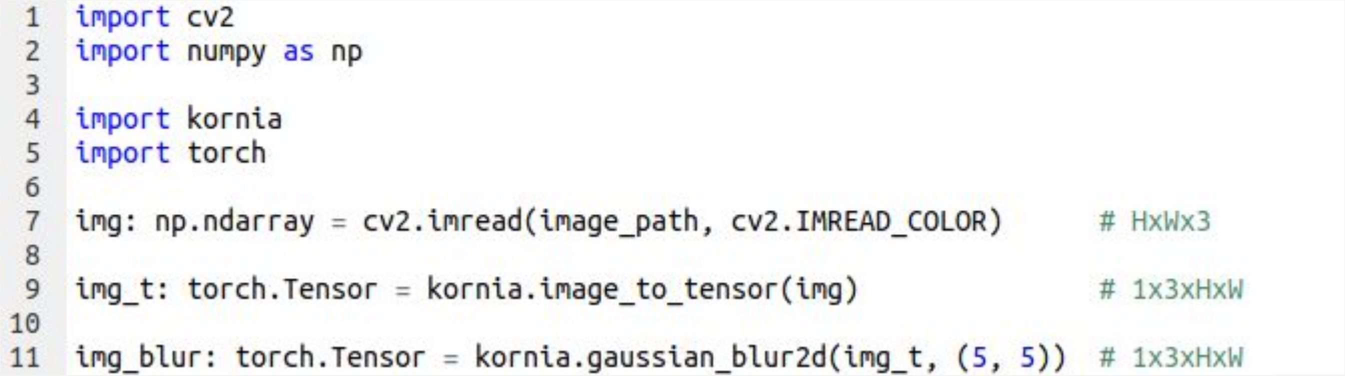
1. Easy to install
2. Easy to use
3. **OpenCV syntax**
4. Ecosystem Integration

3. OpenCV Syntax (I)



```python
import cv2
import numpy as np

img: np.ndarray = cv2.imread(image_path, cv2.IMREAD_COLOR)  # HxWx3

img_blur: np.ndarray = cv2.GaussianBlur(img, (5, 5))        # HxWx3
```

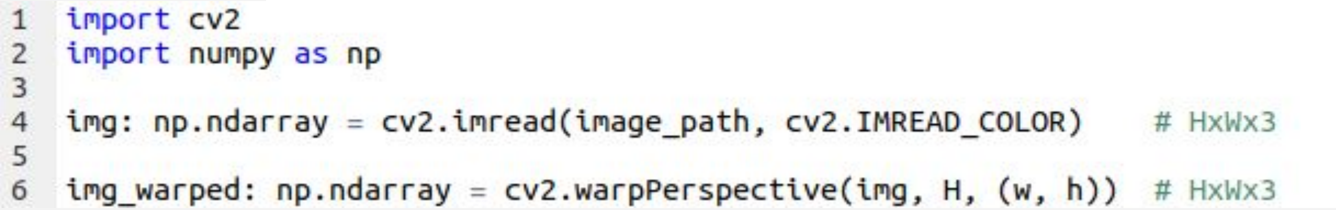
OpenCV



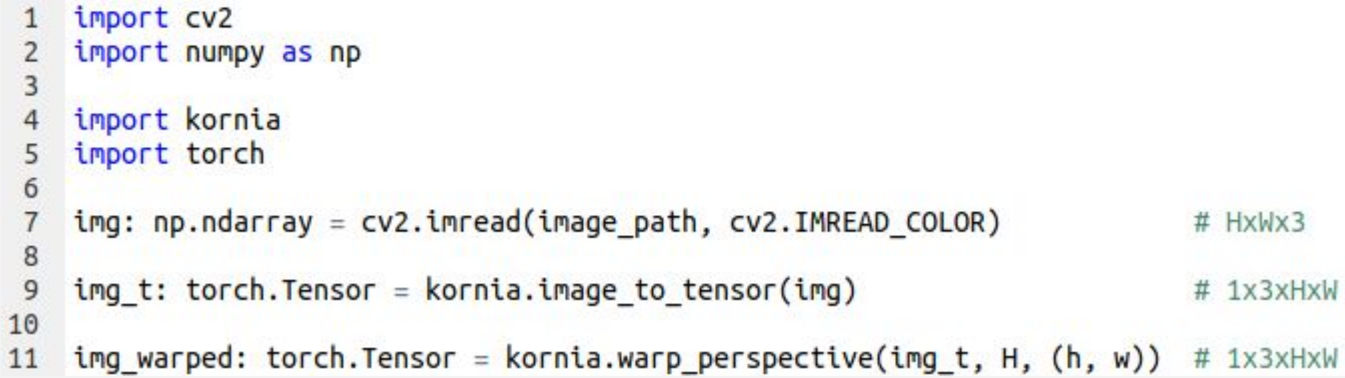
```python
import cv2
import numpy as np

import kornia
import torch

img: np.ndarray = cv2.imread(image_path, cv2.IMREAD_COLOR)      # HxWx3

img_t: torch.Tensor = kornia.image_to_tensor(img)               # 1x3xHxW

img_blur: torch.Tensor = kornia.gaussian_blur2d(img_t, (5, 5))  # 1x3xHxW
```

kornia

3. OpenCV Syntax (II)



```python
import cv2
import numpy as np

img: np.ndarray = cv2.imread(image_path, cv2.IMREAD_COLOR)    # HxWx3

img_warped: np.ndarray = cv2.warpPerspective(img, H, (w, h))   # HxWx3
```



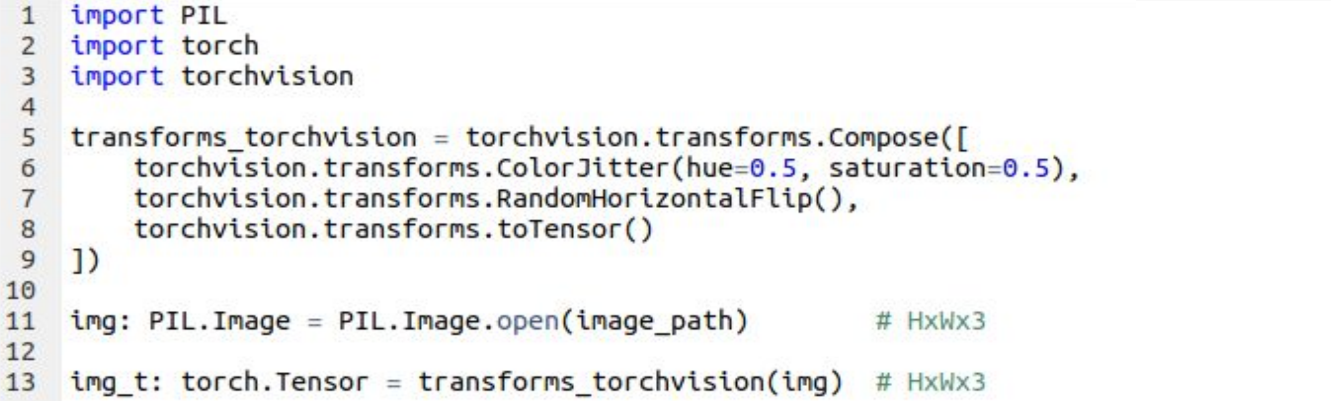
```python
import cv2
import numpy as np

import kornia
import torch

img: np.ndarray = cv2.imread(image_path, cv2.IMREAD_COLOR)            # HxWx3

img_t: torch.Tensor = kornia.image_to_tensor(img)                      # 1x3xHxW

img_warped: torch.Tensor = kornia.warp_perspective(img_t, H, (h, w))   # 1x3xHxW
```

kornia

development

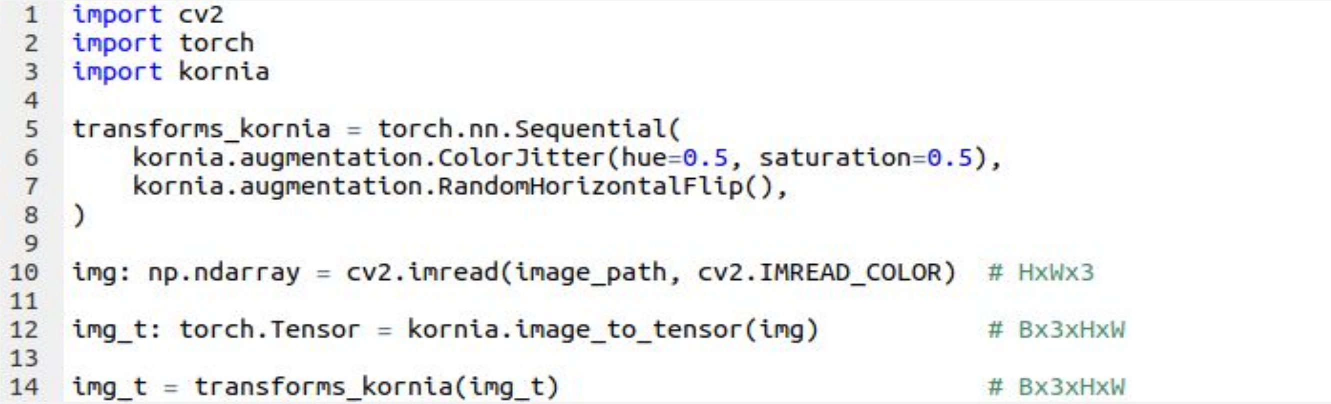
1. Easy to install
2. Easy to use
3. OpenCV syntax
4. **Ecosystem Integration**

3. Compatibility ***torchvision***



```python
import PIL
import torch
import torchvision

transforms_torchvision = torchvision.transforms.Compose([
    torchvision.transforms.ColorJitter(hue=0.5, saturation=0.5),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.toTensor()
])

img: PIL.Image = PIL.Image.open(image_path)          # HxWx3

img_t: torch.Tensor = transforms_torchvision(img)  # HxWx3
```

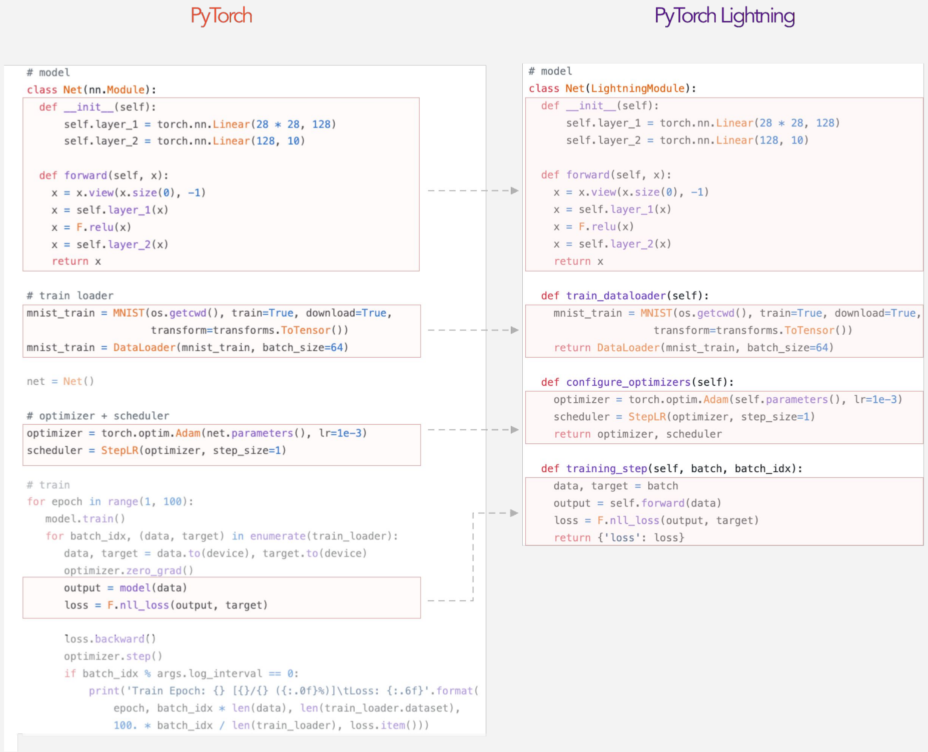
PyTorch



```python
import cv2
import torch
import kornia

transforms_kornia = torch.nn.Sequential(
    kornia.augmentation.ColorJitter(hue=0.5, saturation=0.5),
    kornia.augmentation.RandomHorizontalFlip(),
)

img: np.ndarray = cv2.imread(image_path, cv2.IMREAD_COLOR)  # HxWx3

img_t: torch.Tensor = kornia.image_to_tensor(img)           # Bx3xHxW

img_t = transforms_kornia(img_t)                            # Bx3xHxW
```

kornia

3. Compatibility PyTorch Lightning

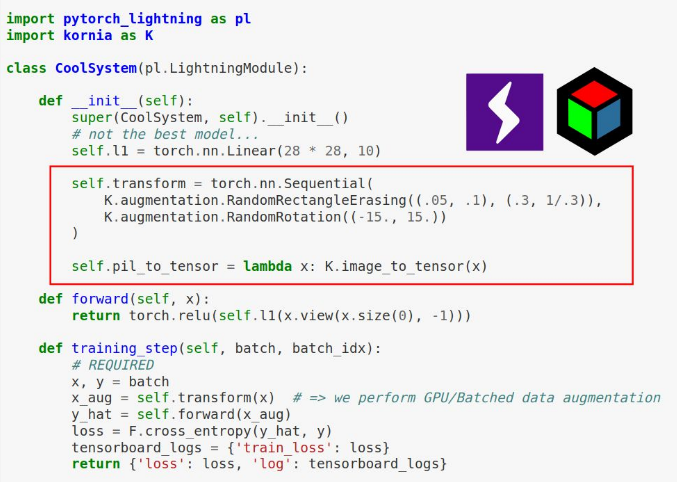


PyTorch

```
# model
class Net(nn.Module):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
   model.train()
   for batch_idx, (data, target) in enumerate(train_loader):
      data, target = data.to(device), target.to(device)
      optimizer.zero_grad()
      output = model(data)
      loss = F.nll_loss(output, target)

      loss.backward()
      optimizer.step()
      if batch_idx % args.log_interval == 0:
          print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

PyTorch Lightning

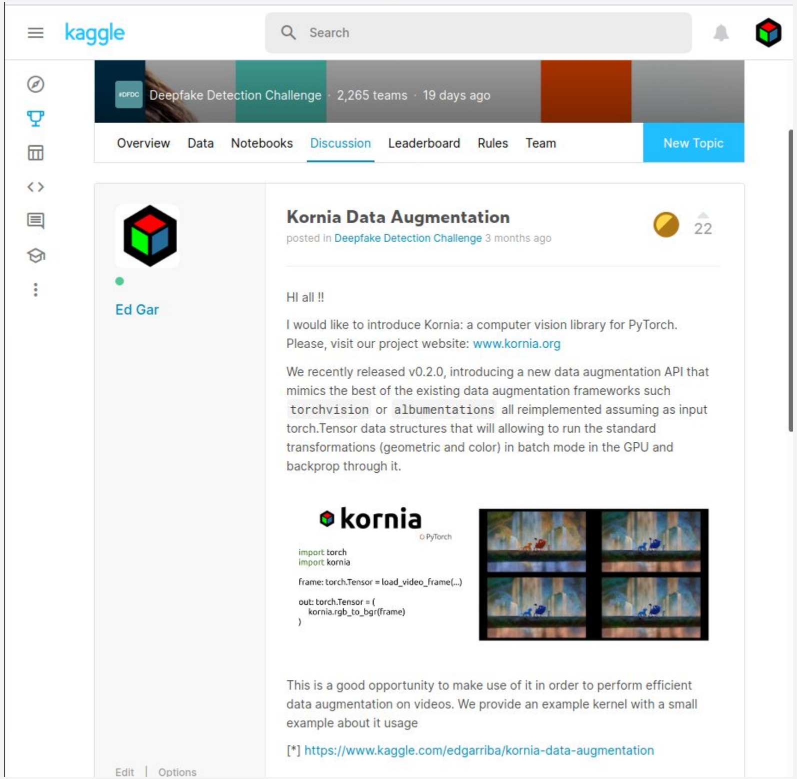
```
# model
class Net(LightningModule):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

  def train_dataloader(self):
    mnist_train = MNIST(os.getcwd(), train=True, download=True,
                        transform=transforms.ToTensor())
    return DataLoader(mnist_train, batch_size=64)

  def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
    scheduler = StepLR(optimizer, step_size=1)
    return optimizer, scheduler

  def training_step(self, batch, batch_idx):
    data, target = batch
    output = self.forward(data)
    loss = F.nll_loss(output, target)
    return {'loss': loss}
```



```
import pytorch_lightning as pl
import kornia as K

class CoolSystem(pl.LightningModule):

    def __init__(self):
        super(CoolSystem, self).__init__()
        # not the best model...
        self.l1 = torch.nn.Linear(28 * 28, 10)

        self.transform = torch.nn.Sequential(
            K.augmentation.RandomRectangleErasing((.05, .1), (.3, 1/.3)),
            K.augmentation.RandomRotation((-15., 15.))
        )

        self.pil_to_tensor = lambda x: K.image_to_tensor(x)

    def forward(self, x):
        return torch.relu(self.l1(x.view(x.size(0), -1)))

    def training_step(self, batch, batch_idx):
        # REQUIRED
        x, y = batch
        x_aug = self.transform(x)  # => we perform GPU/Batched data augmentation
        y_hat = self.forward(x_aug)
        loss = F.cross_entropy(y_hat, y)
        tensorboard_logs = {'train_loss': loss}
        return {'loss': loss, 'log': tensorboard_logs}
```

3. Compatibility

kaggle fast.ai



Kornia Data Augmentation

posted in Deepfake Detection Challenge 3 months ago

HI all !!

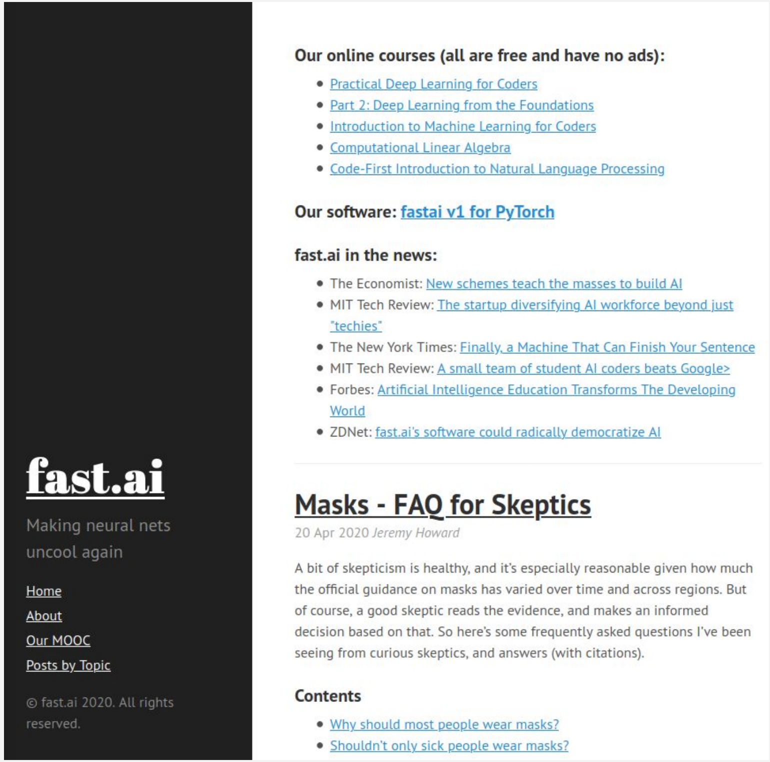
I would like to introduce Kornia: a computer vision library for PyTorch. Please, visit our project website: www.kornia.org

We recently released v0.2.0, introducing a new data augmentation API that mimics the best of the existing data augmentation frameworks such `torchvision` or `albumentations` all reimplemented assuming as input torch.Tensor data structures that will allowing to run the standard transformations (geometric and color) in batch mode in the GPU and backprop through it.

```
import torch
import kornia

frame: torch.Tensor = load_video_frame(...)

out: torch.Tensor = (
    kornia.rgb_to_bgr(frame)
)
```

This is a good opportunity to make use of it in order to perform efficient data augmentation on videos. We provide an example kernel with a small example about it usage

[*] https://www.kaggle.com/edgarriba/kornia-data-augmentation



Our online courses (all are free and have no ads):

- Practical Deep Learning for Coders
- Part 2: Deep Learning from the Foundations
- Introduction to Machine Learning for Coders
- Computational Linear Algebra
- Code-First Introduction to Natural Language Processing

Our software: fastai v1 for PyTorch

fast.ai in the news:

- The Economist: New schemes teach the masses to build AI
- MIT Tech Review: The startup diversifying AI workforce beyond just "techies"
- The New York Times: Finally, a Machine That Can Finish Your Sentence
- MIT Tech Review: A small team of student AI coders beats Google>
- Forbes: Artificial Intelligence Education Transforms The Developing World
- ZDNet: fast.ai's software could radically democratize AI

Masks - FAQ for Skeptics

20 Apr 2020 *Jeremy Howard*

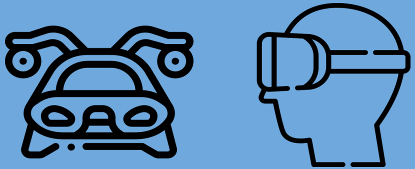
A bit of skepticism is healthy, and it's especially reasonable given how much the official guidance on masks has varied over time and across regions. But of course, a good skeptic reads the evidence, and makes an informed decision based on that. So here's some frequently asked questions I've been seeing from curious skeptics, and answers (with citations).

Contents

- Why should most people wear masks?
- Shouldn't only sick people wear masks?

fast.ai
Making neural nets uncool again
Home
About
Our MOOC
Posts by Topic
© fast.ai 2020. All rights reserved.

Future plans



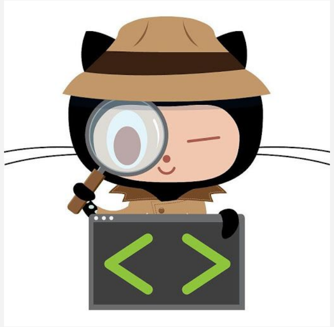
kornia

3D transformations

TorchScriptable

Structure From Motion

We need your help
and feedback !



For users

User chat for Q&A:

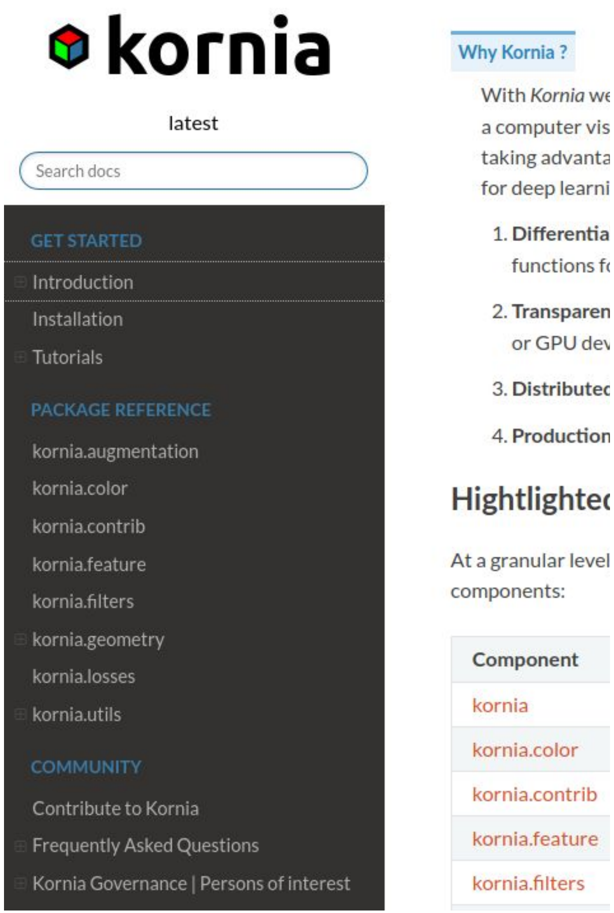
https://discuss.pytorch.org/c/vision/kornia

Official documents:

https://kornia.readthedocs.io/en/latest

SPHINX

Read the Docs



kornia

latest

Search docs

GET STARTED

Introduction

Installation

Tutorials

PACKAGE REFERENCE

kornia.augmentation

kornia.color

kornia.contrib

kornia.feature

kornia.filters

kornia.geometry

kornia.losses

kornia.utils

COMMUNITY

Contribute to Kornia

Frequently Asked Questions

Kornia Governance | Persons of interest

Why Kornia ?

With *Kornia* we
a computer vis
taking advanta
for deep learni

1. **Differentia**
functions fo
2. **Transparen**
or GPU dev
3. **Distributec**
4. **Production**

Hightlightec

At a granular level
components:

Component
kornia
kornia.color
kornia.contrib
kornia.feature
kornia.filters

For developers

Check out our contributions call
https://github.com/kornia/kornia/issues/53

or

Check out docs, and our issues marked as “contributions welcome”:
https://github.com/kornia/kornia/blob/master/CONTRIBUTING.rst

Extra material

Examples:

https://github.com/kornia/kornia-examples

Tutorials:

https://kornia.readthedocs.io/en/latest/tutorials/index.html

www.kornia.org

twitter: @kornia_foss

code: https://github.com/kornia/kornia