

Kornia



Open Source and Computer Vision

Powered by  PyTorch

Зачем? Почему? Для чего?

Аугментации картинок

- Mix

- [RandomCutMixV2](#)
- [RandomJigsaw](#)
- [RandomMixUpV2](#)
- [RandomMosaic](#)
- [RandomTransplantation](#)

```
import kornia as K
img1 = K.io.load_image("panda.jpg", K.io.ImageLoadType.RGB32)
img2 = K.augmentation.RandomEqualize(p=1.0, keepdim=True)(img1)
img3 = K.augmentation.RandomInvert(p=1.0, keepdim=True)(img1)
img4 = K.augmentation.RandomChannelShuffle(p=1.0, keepdim=True)(img1)

plt.figure(figsize=(21, 9))
plt.imshow(K.tensor_to_image(torch.cat([img1, img2, img3, img4], dim=-1)))
plt.show()
```

- Синтаксис, как в torhcivision, переходить будет легко и приятно
- Есть всякие уникальные, типа RandomMosaic и т.д., [гуглите](#), их куча



Аугментации картинок 2

```
dilation()  
erosion()  
opening()  
closing()  
gradient()  
top_hat()  
bottom_hat()
```



Dilation



Erosion



kornia.enhance

kornia.feature

kornia.filters

kornia.nerf

kornia.tracking

kornia.geometry

kornia.sensors

- Синтаксис, [как в opencv](#), если есть опыт работы с ним, то пригодится
- В библиотеке есть ещё куча всего, типа homography, есть смысл поизучать доки

Аугментации видео

```
import kornia.augmentation as K

transform = K.VideoSequential(
    K.RandomAffine(360),
    K.RandomGrayscale(p=0.5),
    K.RandomAffine(p=0.5)
    data_format="BCTHW",
    same_on_frame=True
)
```



```
import torch
import kornia

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

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



- Помимо kornia есть в volumentations
- Синтаксис такой же, как в аугментациях картинок

GPU

- Все аугментации можно засунуть в `nn.Sequential`
- Это очень удобно, потому что можно засовывать их прямо в модельку, необязательно в ладер
- Ещё можно подбирать оптимальные аугментации через [AutoAugment](#) (я не пробовал, но звучит хайпово)

```
transforms = nn.Sequential(  
    K.augmentation.RandomGrayscale(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomHorizontalFlip(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomErasing(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33, scale=(0.02, 0.33), ratio=(0.3, 3.3)  
    )  
)
```

GPU

- Все аугментации можно засунуть в `nn.Sequential`
- Это очень удобно, потому что `Sequential` можно перенести на другой девайс абсолютно естественным образом
- `Albumentations` правда тоже умеет работать на гпухе, и быстрее

```
transforms = nn.Sequential(  
    K.augmentation.RandomGrayscale(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomHorizontalFlip(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomErasing(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33, scale=(0.02, 0.33), ratio=(0.3, 3.3)  
    )  
)
```

Libraries	TorchVision	Albumentations	Kornia (GPU)
Batch Size	1	1	1
RandomPerspective	4.88±1.82	4.68±3.60	4.74±2.84
ColorJiggle	4.40±2.88	3.58±3.66	4.14±3.85
RandomAffine	3.12±5.80	2.43±7.11	3.01±7.80
RandomVerticalFlip	0.32±0.08	0.34±0.16	0.35±0.82
RandomHorizontalFlip	0.32±0.08	0.34±0.18	0.31±0.59

GPU

- Все аугментации можно засунуть в `nn.Sequential`
- Это очень удобно, потому что `Sequential` можно перенести на другой девайс абсолютно естественным образом
- `Albumentations` правда тоже умеет работать на гпухе, и быстрее, **если она у вас одна...**

```
transforms = nn.Sequential(  
    K.augmentation.RandomGrayscale(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomHorizontalFlip(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33  
    ),  
    K.augmentation.RandomErasing(  
        same_on_batch=False,  
        keepdim=True,  
        p=0.33, scale=(0.02, 0.33), ratio=(0.3, 3.3)  
    )  
)
```

Libraries	TorchVision	Albumentations	Kornia (GPU)		
Batch Size	1	1	1	32	128
RandomPerspective	4.88±1.82	4.68±3.60	4.74±2.84	0.37±2.67	0.20±27.00
ColorJiggle	4.40±2.88	3.58±3.66	4.14±3.85	0.90±24.68	0.83±12.96
RandomAffine	3.12±5.80	2.43±7.11	3.01±7.80	0.30±4.39	0.18±6.30
RandomVerticalFlip	0.32±0.08	0.34±0.16	0.35±0.82	0.02±0.13	0.01±0.35
RandomHorizontalFlip	0.32±0.08	0.34±0.18	0.31±0.59	0.01±0.26	0.01±0.37

DiffAug

- Все аугментации дифференцируемы
- В теории должно давать более быструю сходимость

```
def forward(self, x):  
  
    # apply transforms if present  
    if self.transforms is not None:  
        conv_out = self.transforms(x)
```

Ну и на бекварде тоже всё будет ништяк, базарю. Обычные аугментации так не умеют

Differentiable Augmentation for Data-Efficient GAN Training