

# Deep Learning Practitioner's Toolbox



# Logistics

- Project proposal submission is due **today**.
- Reviews will be published in the following days.



# The Toolbox

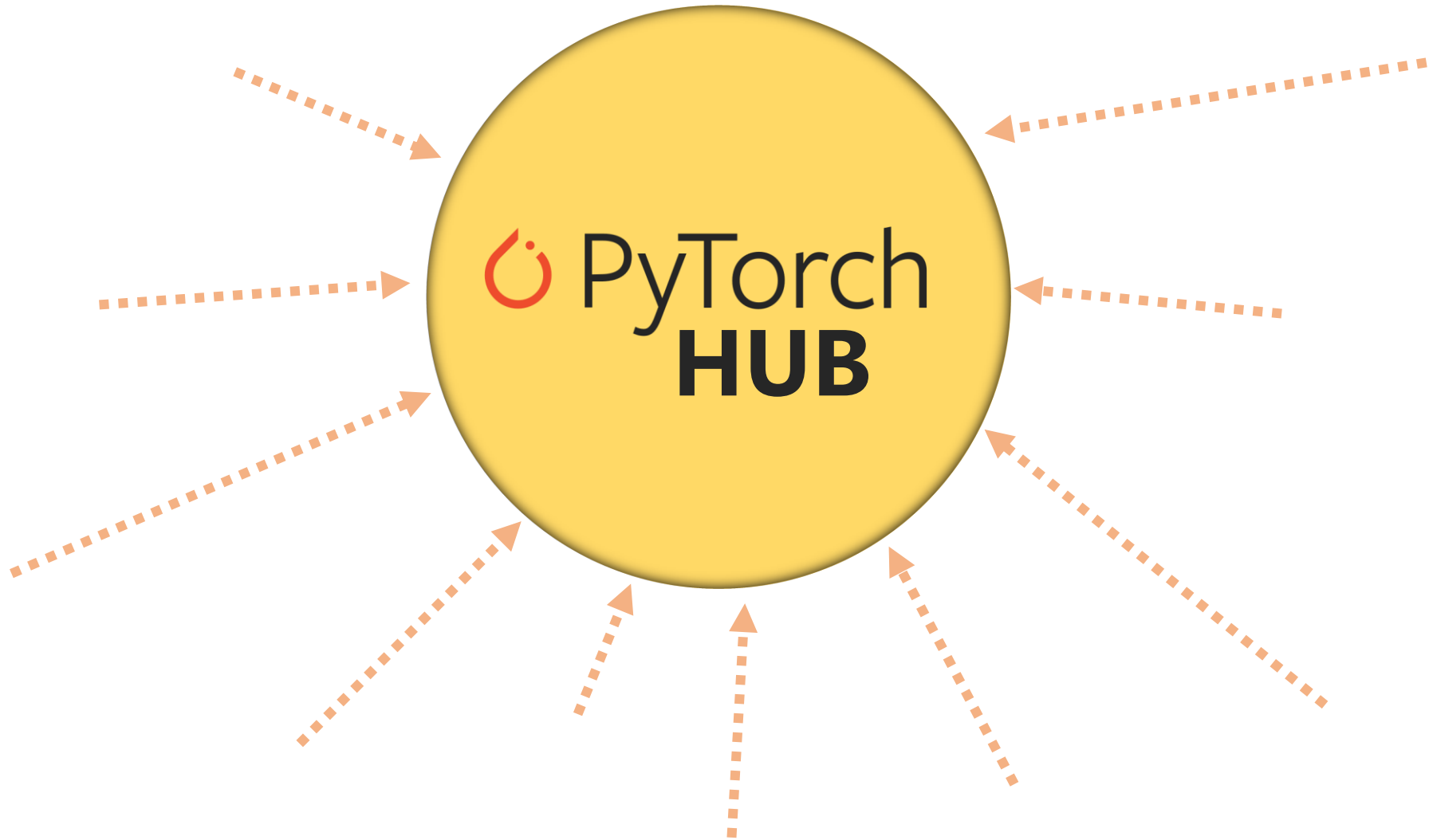
- Torch hub
- Hooks
- Tensorboard
- Transforms
- Reproducibility
- Saving & Loading Models

# Don't Reinvent the Wheel!



Image Credits: <https://postandparcel.info/80130/news/parcel/reinventing-the-wheel/>

## Use Existing Tools!



# Torch Hub

```
# list of available models in pyTorch's repo.
torch.hub.list('pytorch/vision')

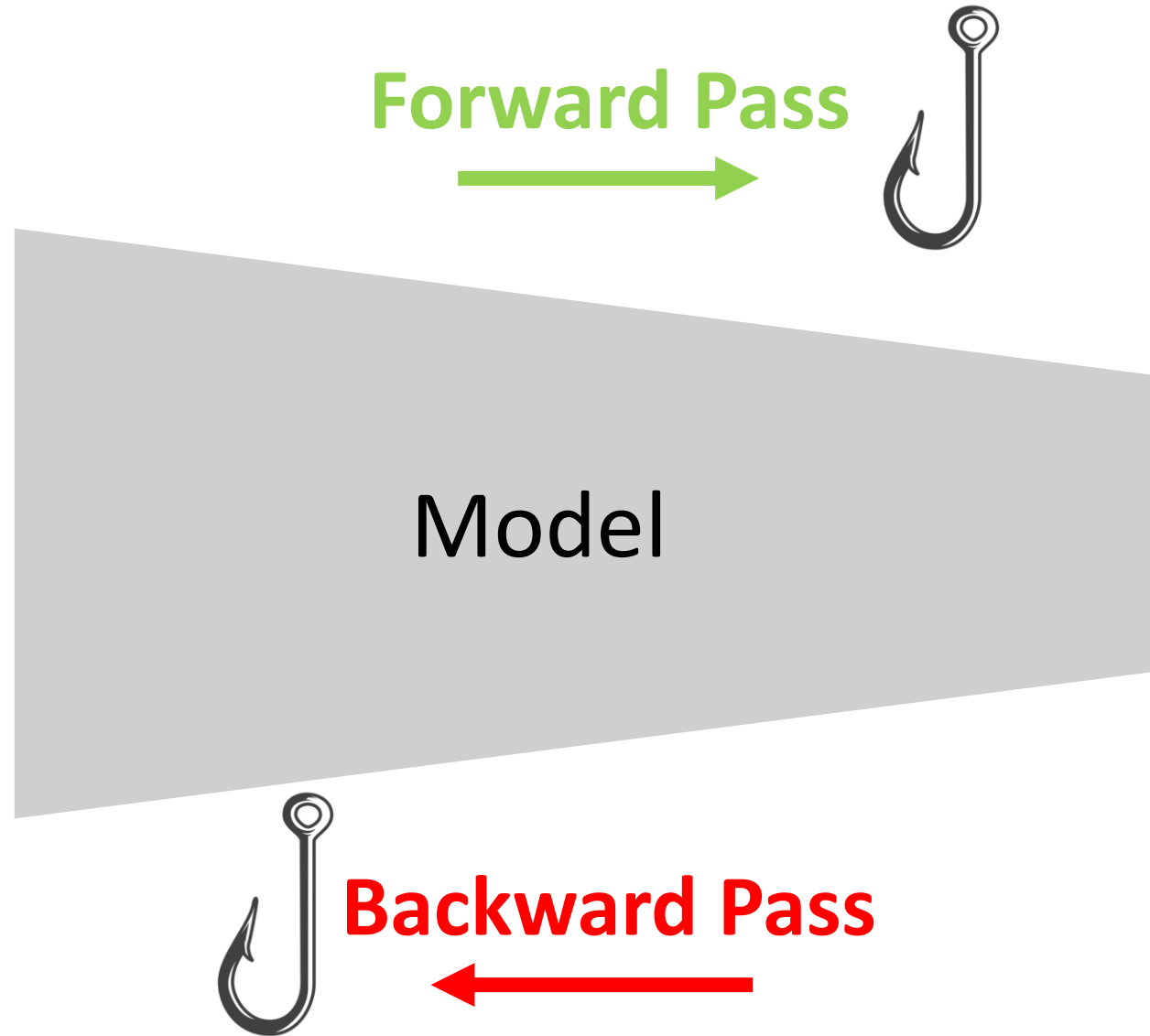
# printed: ['alexnet', 'deeplabv3_mobilenet_v3_large',
'deeplabv3_resnet101', ...]

# load model
model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)
model.eval()

input = torch.randn(1, 3, 224, 224)
output = model(input)

# output.shape is 1x1000
```

# Hooks





# Register a Hook

```
def my_hook(module, input, output):  
    # module: the module being hooked  
    # input: a tuple of inputs  
    # output: a tuple of outputs  
    print("hook!", input[0].shape, output[0].shape)  
  
# register the hook  
net.conv1.register_forward_hook(my_hook)  
  
# use the hook  
y = net(x)  
# printed: "hook! torch.Size([...]) torch.Size([...])"
```

# Remove a Hook

- A remove handle is returned during the registration.

```
# register the hook
handle = net.conv1.register_forward_hook(my_hook)

# remove the hook
handle.remove()

# use the hook
y = net(x)
# nothing is printed
```

# Feature Extraction

```
class FeatureExtractor:
    def __init__(self, model):
        self._feature = None
        self.model = model
        self._handle = self.model.layer1.register_forward_hook(self._get_feature_hook())

    def __del__(self):
        self._handle.remove()

    def _get_feature_hook(self):
        def _get_feature(module, input, output):
            self._feature = output
        return _get_feature

    def __call__(self, input):
        output = self.model(input)
        return self._feature
```

# Feature Extraction

```
# create a model
model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)

# create a feature extractor
feature_extractor = FeatureExtractor(model)

# create input
input = torch.randn(1, 3, 224, 224)

# get features
feat = feature_extractor(input)

# feat.shape is 1x64x56x56
```

# TensorBoard

live loss plot

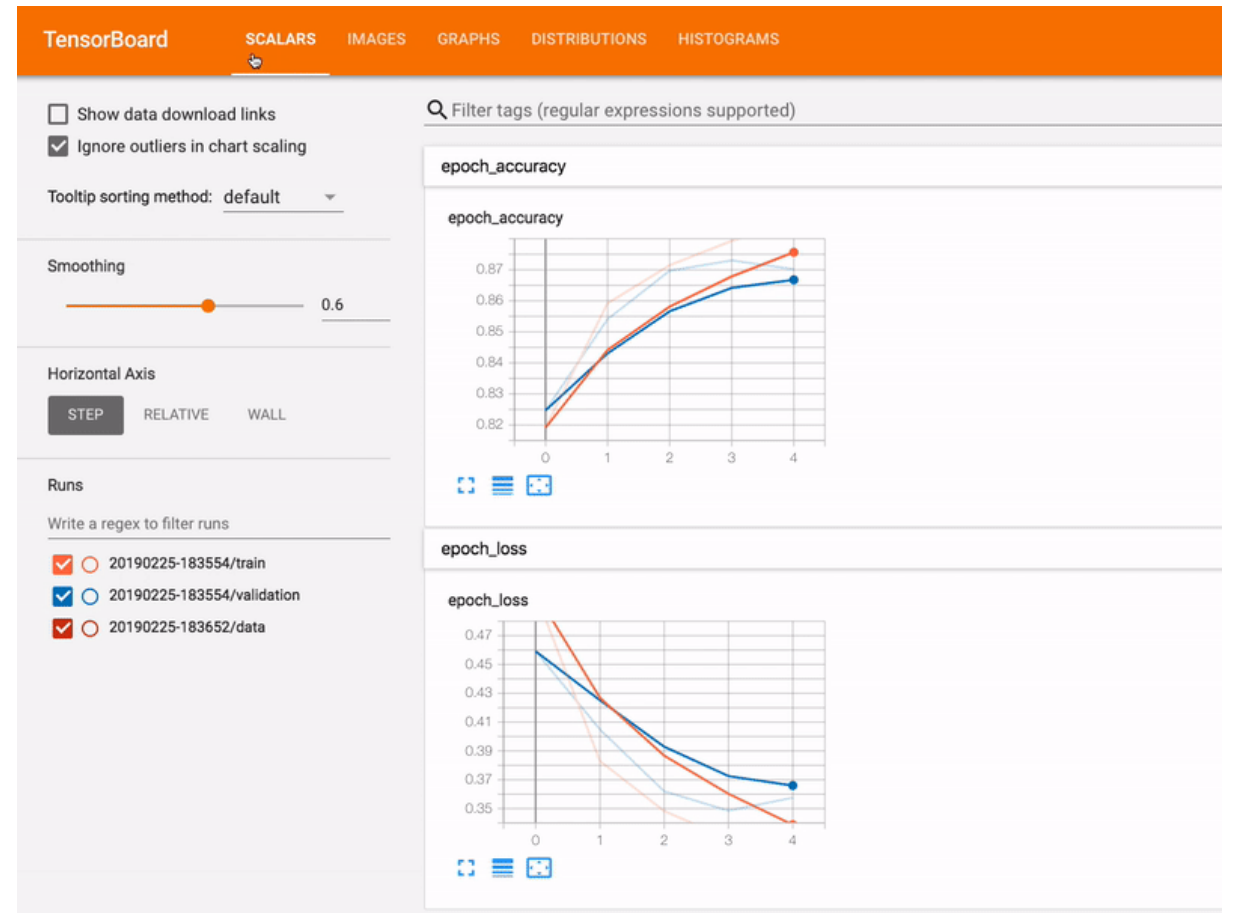
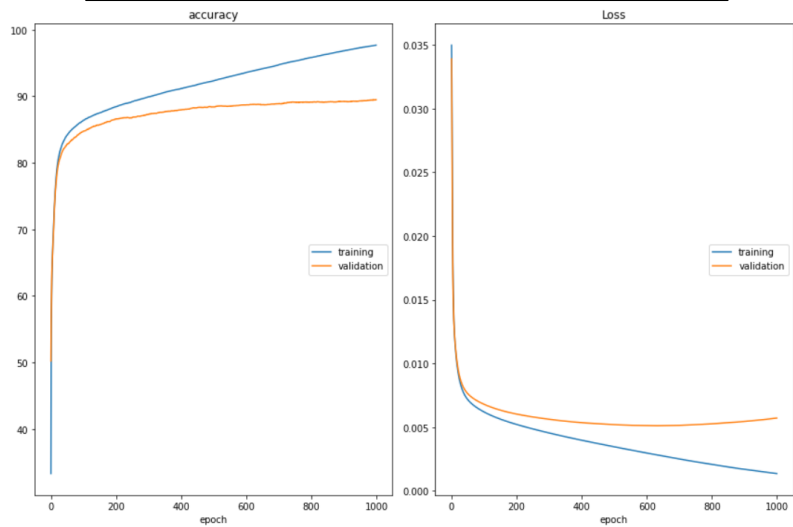


Image credit to Tensorflow

# TensorBoard Preparations

```
# install tensorboard
!pip install tensorboard

%load_ext tensorboard

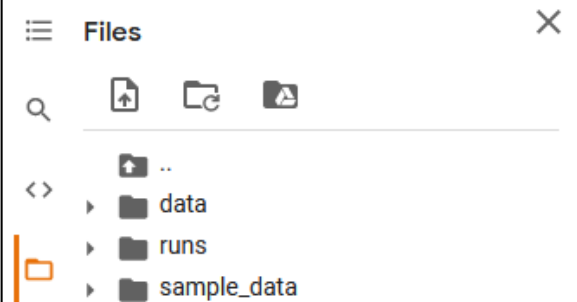
...

# run the user interface
%tensorboard --logdir .
```

# TensorBoard

# Summary Writer

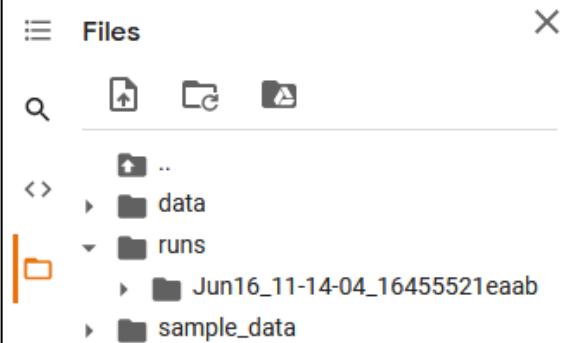
```
from torch.utils.tensorboard import SummaryWriter  
  
writer = SummaryWriter()
```



# TensorBoard

# Summary Writer

```
from torch.utils.tensorboard import SummaryWriter  
  
writer = SummaryWriter()
```

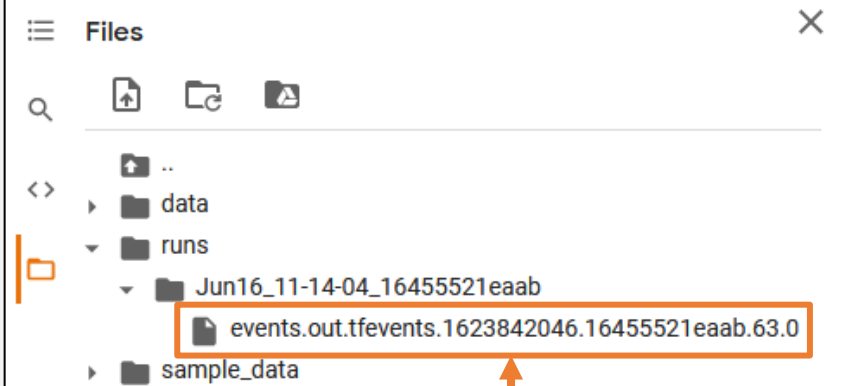




# TensorBoard

# Summary Writer

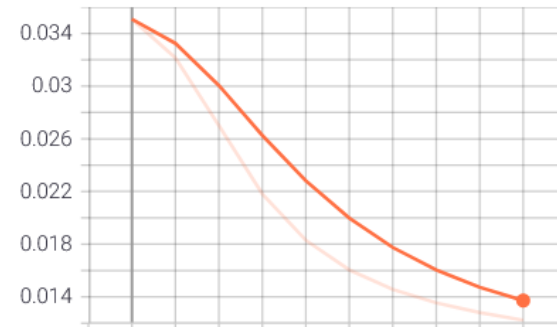
```
from torch.utils.tensorboard import SummaryWriter  
  
writer = SummaryWriter()
```



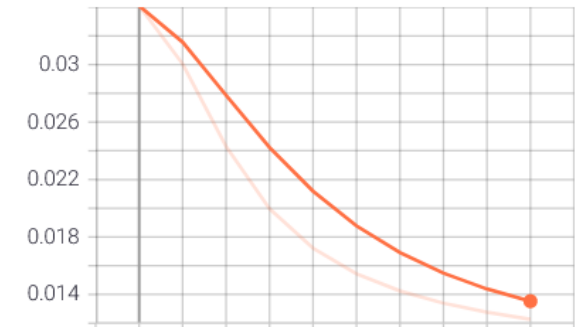
**Everything is here!**

```
# train
for epoch in range(EPOCHS):
    ...
    writer.add_scalar("Loss/train", loss, epoch)
    writer.add_scalar("Loss/val", val_loss, epoch)
    ...
```

Loss/train  
tag: Loss/train



Loss/val  
tag: Loss/val



# TensorBoard

# Summary Writer

```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()

# train
for epoch in range(EPOCHS):
    ...
    writer.add_scalar("Loss/train", loss, epoch)
    writer.add_scalar("Loss/val", val_loss, epoch)
    writer.add_scalar("Accuracy/train", acc, epoch)
    writer.add_scalar("Accuracy/val", val_acc, epoch)
    ...
```



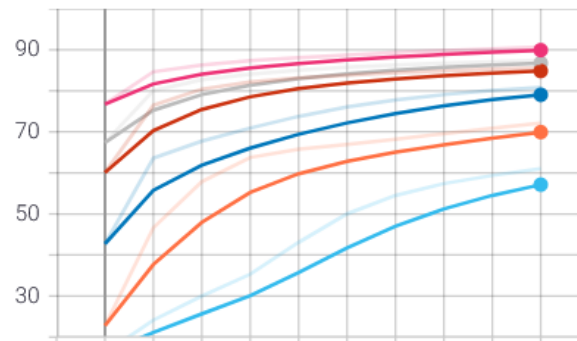
## Runs

Write a regex to filter runs

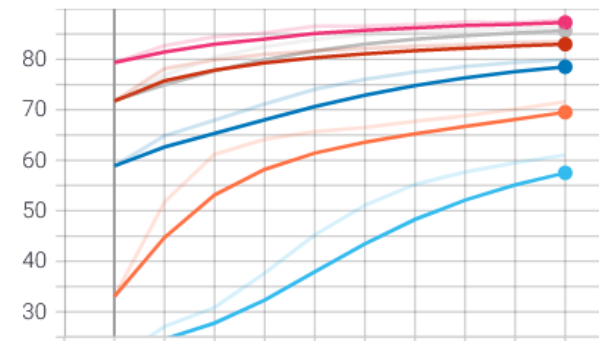
- runs/Jun13\_16-48-43\_16da605f563c
- runs/Jun13\_16-48-43\_16da605f563c/1623602980.9918222
- runs/Jun13\_17-26-20\_16da605f563c
- runs/Jun13\_17-26-20\_16da605f563c/1623605237.2393396
- runs/Jun13\_17-28-20\_16da605f563c
- runs/Jun13\_17-28-20\_16da605f563c/1623605356.016401
- runs/Jun13\_17-32-57\_16da605f563c
- runs/Jun13\_17-32-57\_16da605f563c/1

TOGGLE ALL RUNS

Accuracy/train  
tag: Accuracy/train

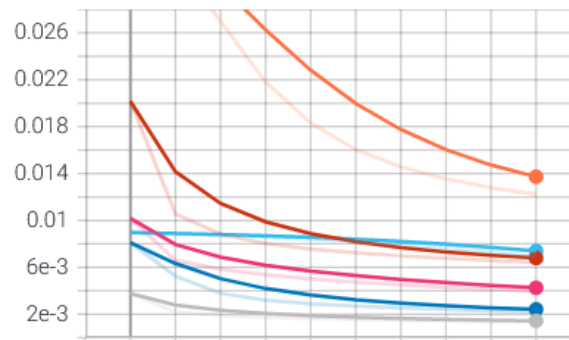


Accuracy/val  
tag: Accuracy/val

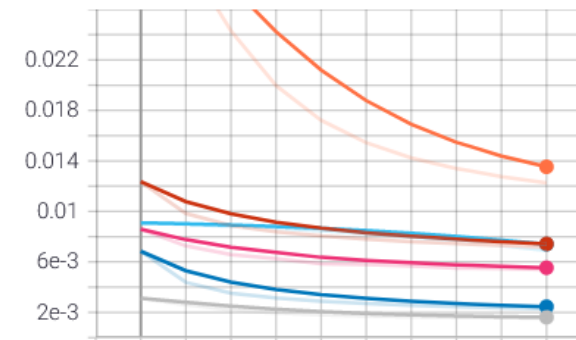


## Loss

Loss/train  
tag: Loss/train



Loss/val  
tag: Loss/val

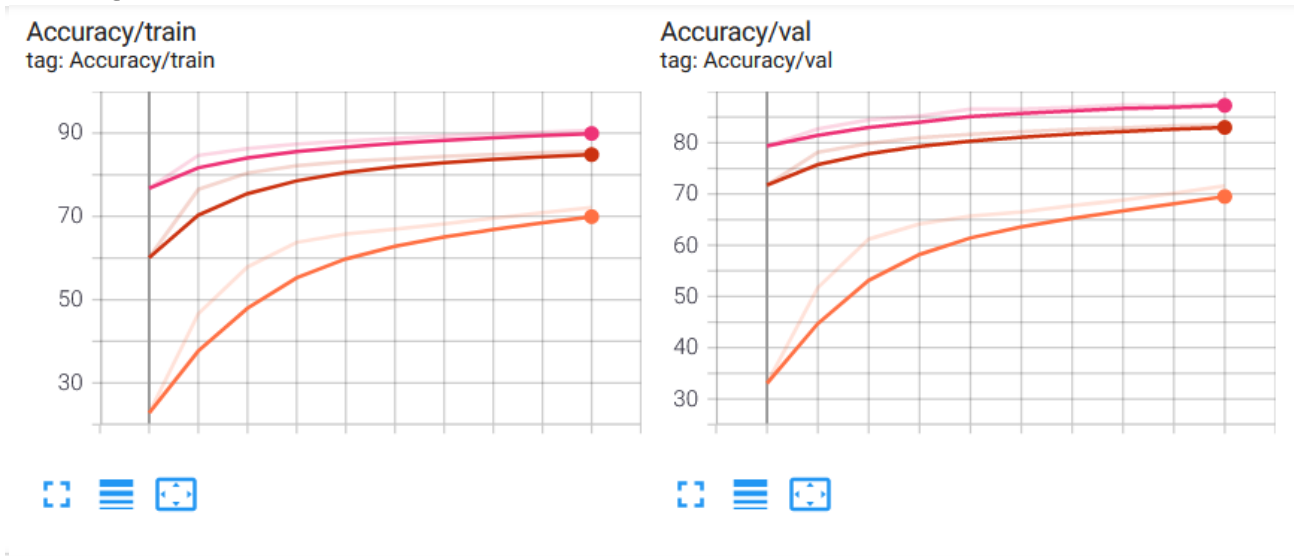


Runs

Write a regex to filter runs

- runs/Jun13\_16-48-43\_16da605f563c
- runs/Jun13\_16-48-43\_16da605f563c/1623602980.9918222
- runs/Jun13\_17-26-20\_16da605f563c
- runs/Jun13\_17-26-20\_16da605f563c/1623605237.2393396
- runs/Jun13\_17-28-20\_16da605f563c
- runs/Jun13\_17-28-20\_16da605f563c/1623605356.016401
- runs/Jun13\_17-32-57\_16da605f563c
- runs/Jun13\_17-32-57\_16da605f563c/1

TOGGLE ALL RUNS



**Prediction-time Batch Norm ImageNet experiments** Created on Jun 9, 2020, 8:07:05 AM  
 Colab notebook with paper plots: <https://colab.research.google.com/drive/11N0wDznMQOuLrRwRoumDCrhSalhkgjof>

Show data download links  
 Ignore outliers in chart scaling

Tooltip sorting method: **default**

Smoothing

Horizontal Axis  
 STEP  RELATIVE  WALL

Runs

Write a regex to filter runs

- ensemble\_prediction\_bn
- ensemble\_train\_bn
- temp\_scaling\_prediction\_bn
- temp\_scaling\_train\_bn
- vanilla\_prediction\_bn
- vanilla\_train\_bn

TOGGLE ALL RUNS

experiment FRbuxfG5SkaFPQOH4OcpYw

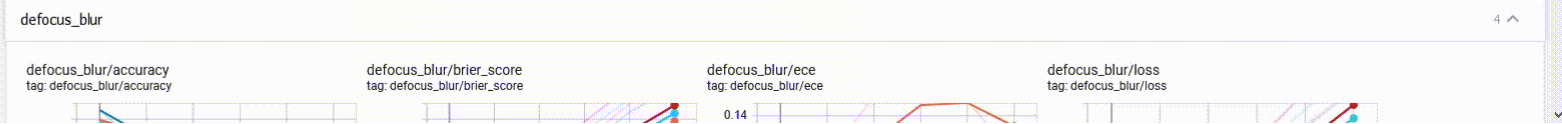
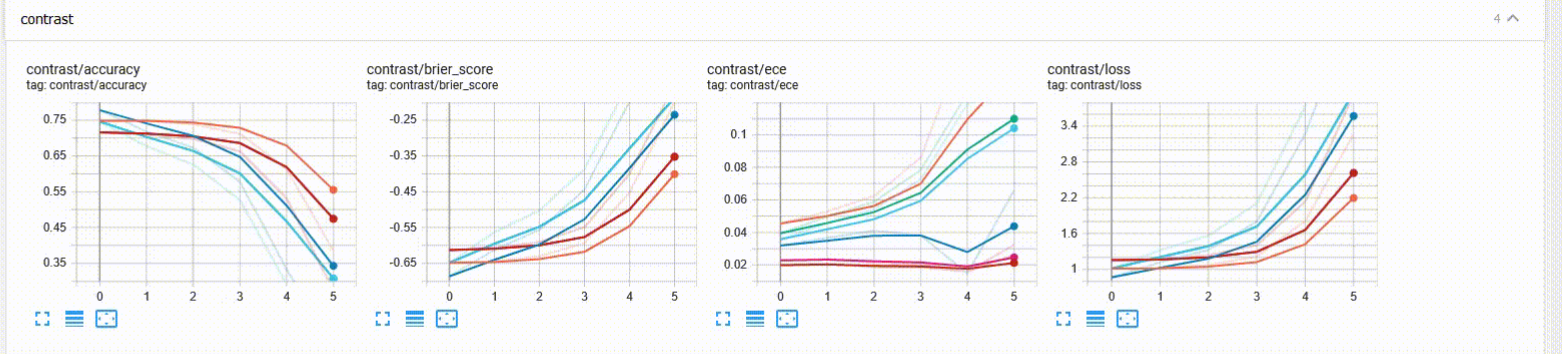
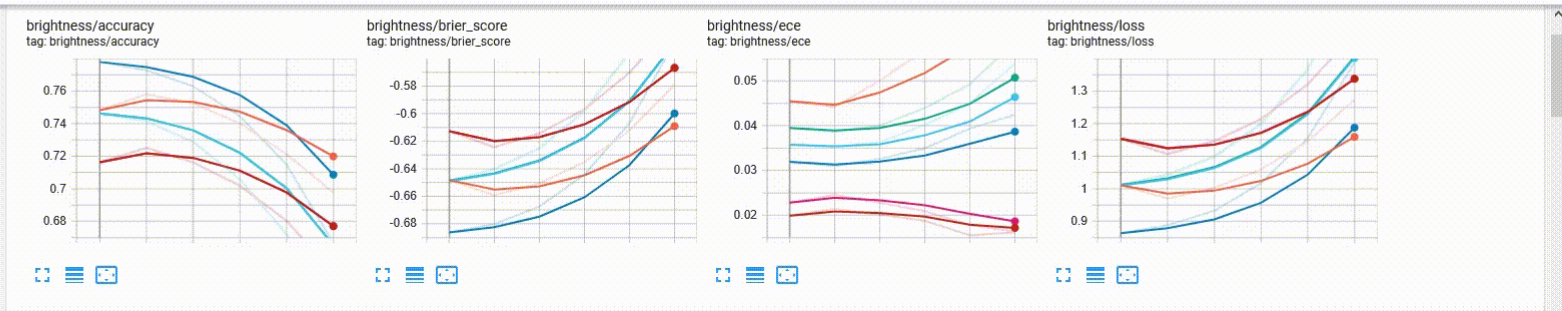


Image credit to Tensorboard.dev



# TensorBoard

# Log Images

```
# fetch a batch of data
X, y = next(iter(train_dataloader))

# create an image from all the images
grid = torchvision.utils.make_grid(X)

# add to summary
writer.add_image("images", grid)
```



```
# create matplotlib figure
figure = ...

# add to summary
writer.add_figure("batch_example", figure)
```

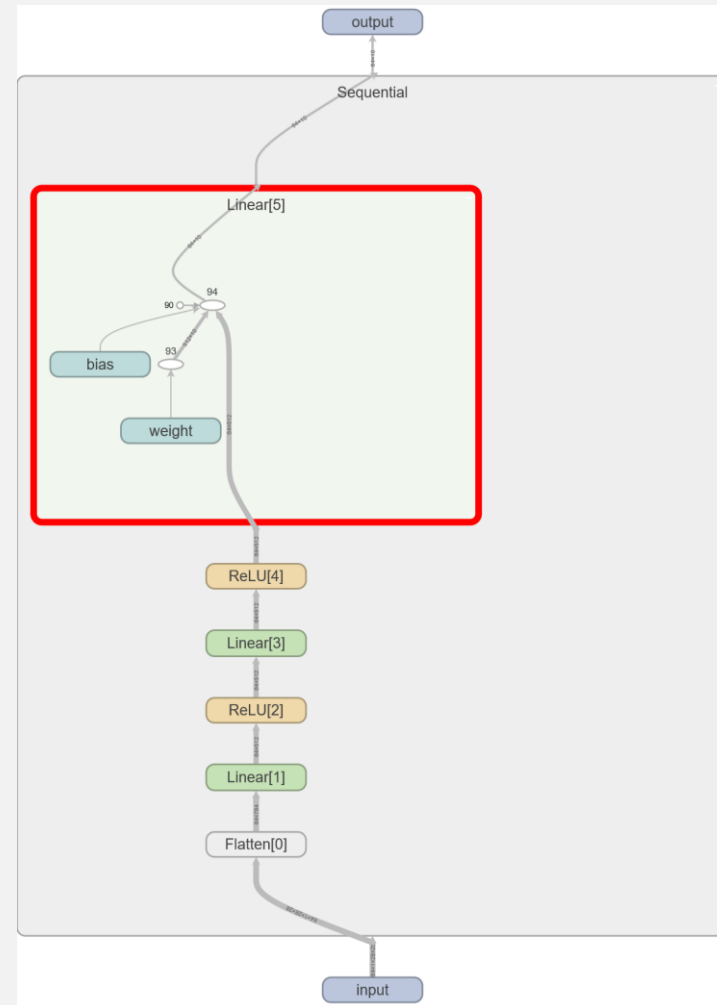
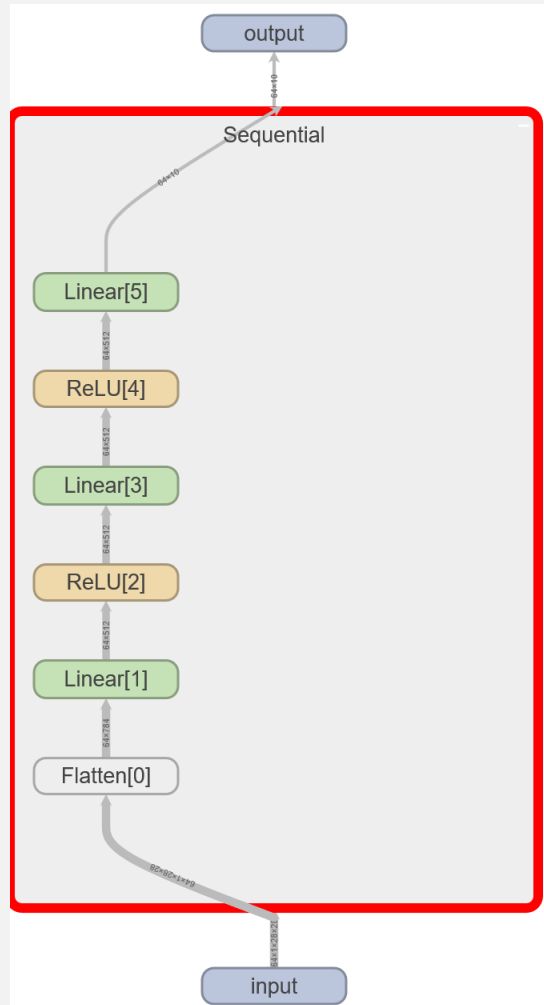
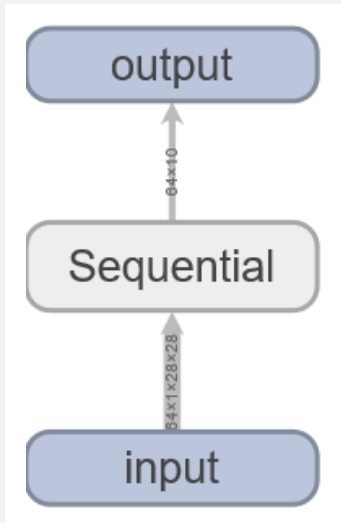




# TensorBoard

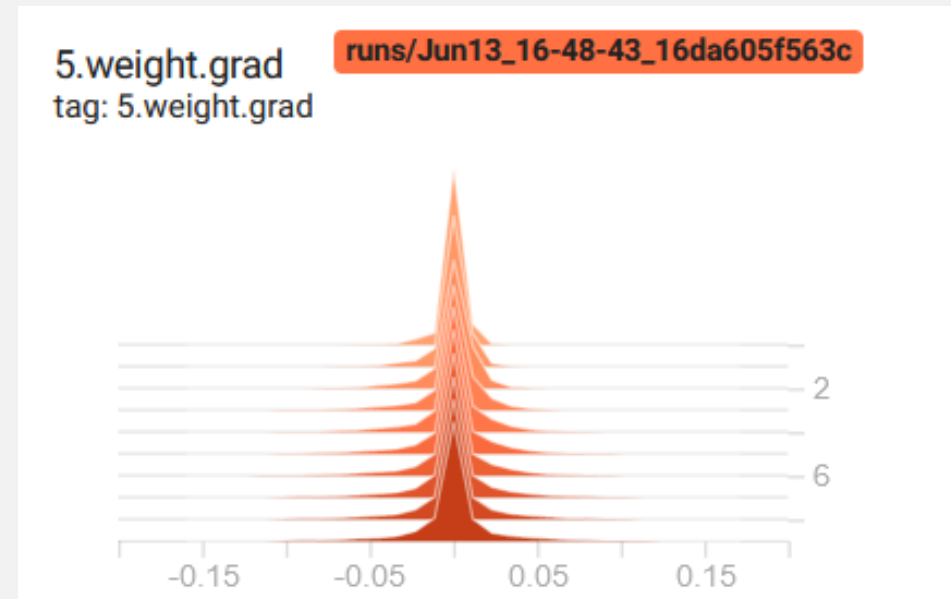
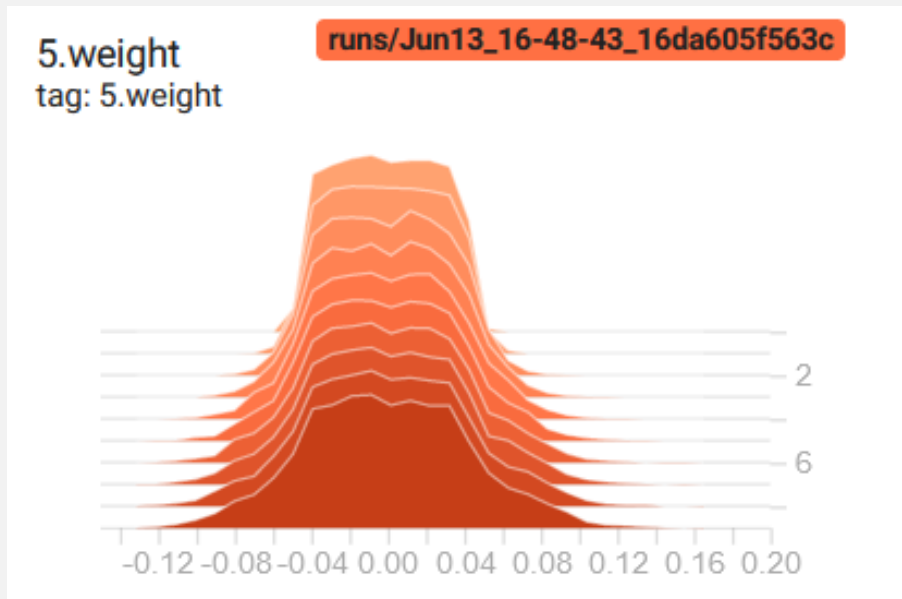
# Graphs

```
writer.add_graph(model, X)
```



# TensorBoard Histograms

```
for name, weight in model.named_parameters():  
    writer.add_histogram(name, weight, epoch)  
    writer.add_histogram(f'{name}.grad', weight.grad, epoch)
```



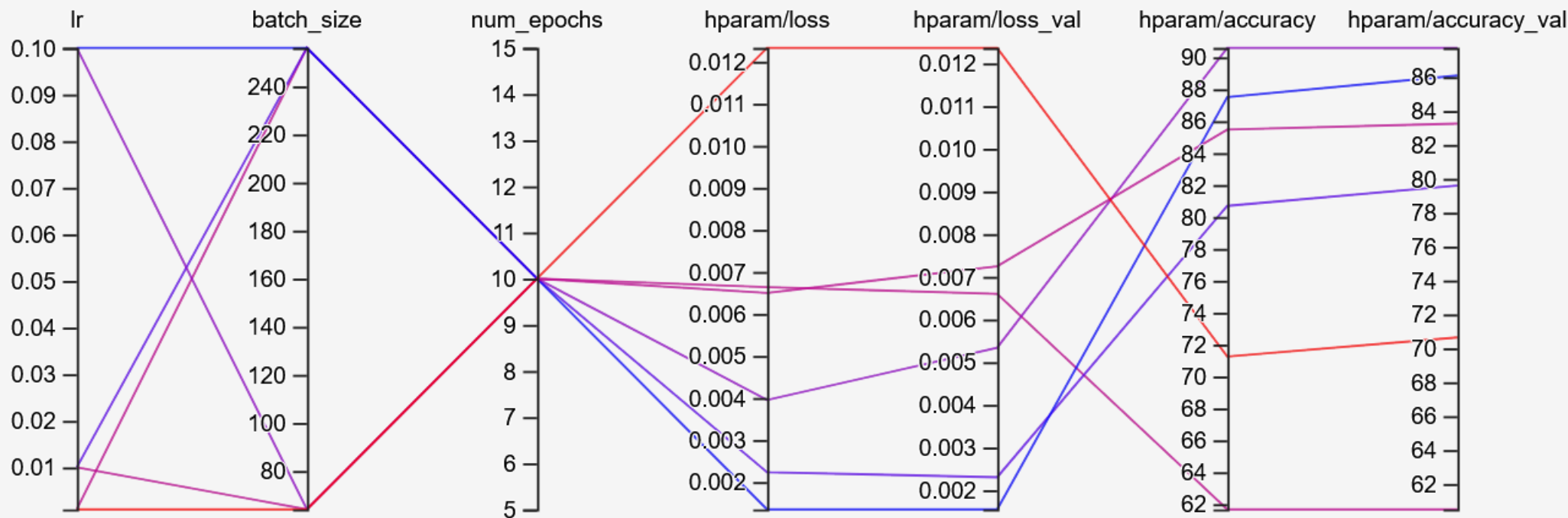
```
writer.add_hparams({'lr': LR, 'batch_size': B, 'num_epochs': EPOCHS},  
                  {'hparam/loss': train_loss, 'hparam/loss_val': val_loss,  
                  'hparam/accuracy': train_acc, 'hparam/accuracy_val': val_acc})
```

Trial ID	Show Metrics	lr	batch_size	num_epochs	hparam/loss	hparam/loss_val	hparam/accuracy	hparam/accuracy_val
runs/Jun13_19-...	<input type="checkbox"/>	0.0010000	256.00	10.000	0.0066344	0.0065976	61.670	60.500
runs/Jun13_19-...	<input type="checkbox"/>	0.010000	256.00	10.000	0.0022308	0.0023018	80.687	79.600
runs/Jun13_19-...	<input type="checkbox"/>	0.10000	256.00	10.000	0.0013479	0.0015497	87.502	86.080
runs/Jun13_19-...	<input type="checkbox"/>	0.10000	64.000	10.000	0.0039495	0.0053355	90.593	87.720
runs/Jun13_19-...	<input type="checkbox"/>	0.010000	64.000	10.000	0.0064962	0.0072452	85.473	83.250
runs/Jun13_19-...	<input type="checkbox"/>	0.0010000	64.000	10.000	0.012328	0.012364	71.242	70.640

# TensorBoard

# Hyper-parameters

```
writer.add_hparams({'lr': LR, 'batch_size': B, 'num_epochs': EPOCHS},  
                 {'hparam/loss': train_loss, 'hparam/loss_val': val_loss,  
                 'hparam/accuracy': train_acc, 'hparam/accuracy_val': val_loss})
```



# Don't Reinvent the Wheel!



Image Credits: <https://postandparcel.info/80130/news/parcel/reinventing-the-wheel/>

## Use Existing Tools!

# A Alumentations

- Data Augmentations for various tasks
  - Classification / Representation Learning



A.RandomCrop  
probability 100%



A.HorizontalFlip  
probability 50%



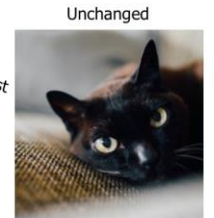
A.HorizontalFlip  
probability 50%



A.RandomBrightnessContrast  
probability 20%



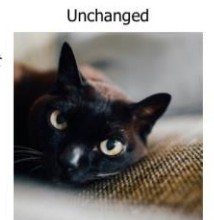
A.RandomBrightnessContrast  
probability 80%



A.RandomBrightnessContrast  
probability 20%



A.RandomBrightnessContrast  
probability 80%



```
import alumentations as A
```

```
transform = A.Compose([  
    A.RandomCrop(width=256, height=256),  
    A.HorizontalFlip(p=0.5),  
    A.RandomBrightnessContrast(p=0.2),  
])
```

Image credit to alumentations tutorial:

[https://alumentations.ai/docs/getting\\_started/image\\_augmentation/](https://alumentations.ai/docs/getting_started/image_augmentation/)

# A Alumentations

- Data Augmentations for various tasks
  - Classification / Representation Learning
  - Object Detection

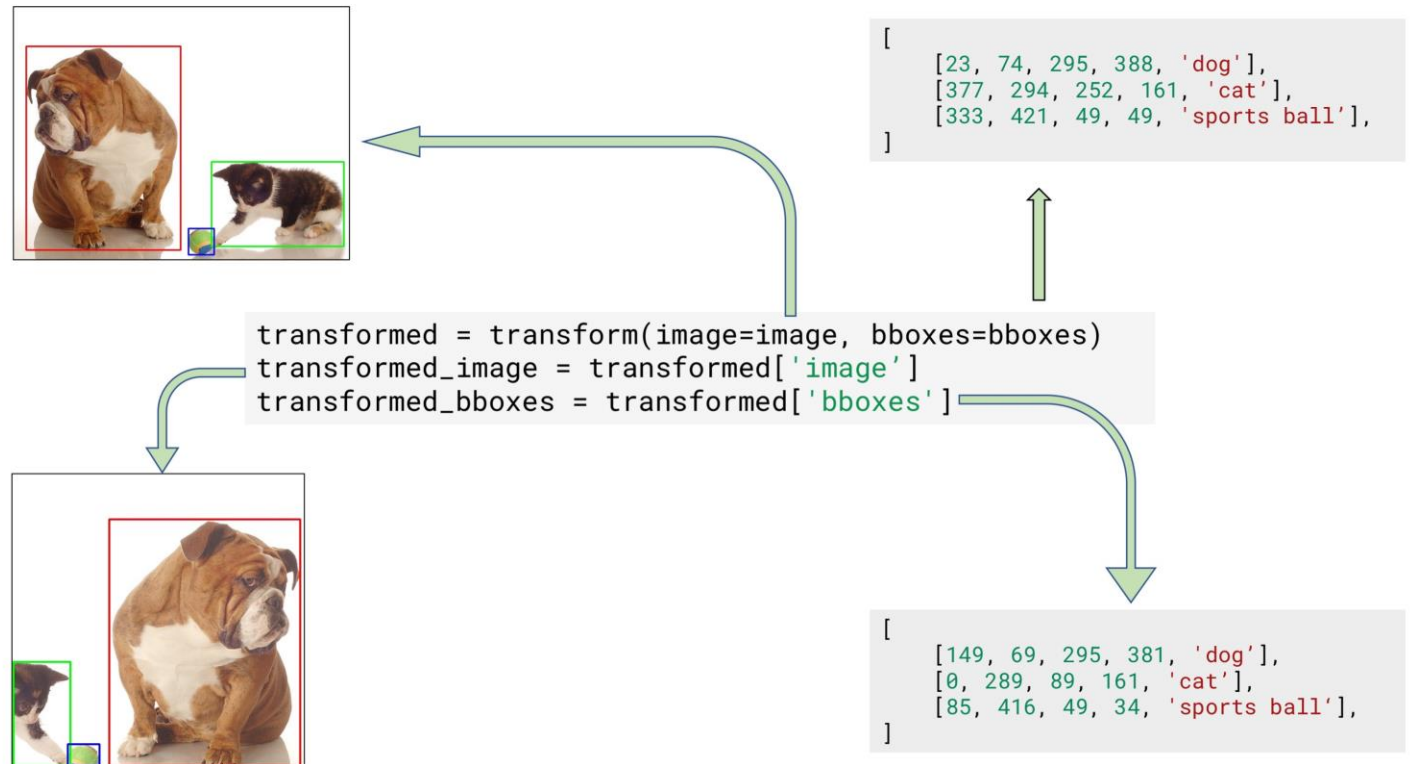


Image credit to alumentations tutorial:

[https://alumentations.ai/docs/getting\\_started/bounding\\_boxes\\_augmentation/](https://alumentations.ai/docs/getting_started/bounding_boxes_augmentation/)



# A Alumentations

- Data Augmentations for various tasks
  - Classification / Representation Learning
  - Object Detection
  - Keypoint Detection

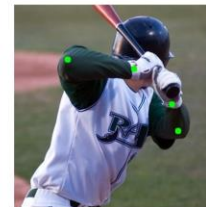


```
[  
  [414, 249],  
  [236, 134],  
  [404, 206],  
  [343, 149],  
  [215, 387],  
]
```

```
[  
  'left_elbow',  
  'right_elbow',  
  'left_wrist',  
  'right_wrist',  
  'right_hip',  
]
```

```
[  
  'left',  
  'right',  
  'left',  
  'right',  
  'right',  
  'right',  
]
```

```
transformed = transform(image=image, keypoints=keypoints, class_labels=class_labels, class_sides=class_sides)  
transformed_class_sides = transformed['class_sides']  
transformed_class_labels = transformed['class_labels']  
transformed_keypoints = transformed['keypoints']  
transformed_image = transformed['image']
```



```
[  
  [264, 203],  
  [86, 88],  
  [254, 160],  
  [193, 103],  
]
```

```
[  
  'left_elbow',  
  'right_elbow',  
  'left_wrist',  
  'right_wrist',  
]
```

```
[  
  'left',  
  'right',  
  'left',  
  'right',  
]
```

Image credit to alumentations tutorial:  
[https://alumentations.ai/docs/getting\\_started/keypoints\\_augmentation/](https://alumentations.ai/docs/getting_started/keypoints_augmentation/)



# A Alumentations

- Data Augmentations for various tasks
  - Classification / Representation Learning
  - Object Detection
  - Keypoint Detection
  - Mask Segmentation

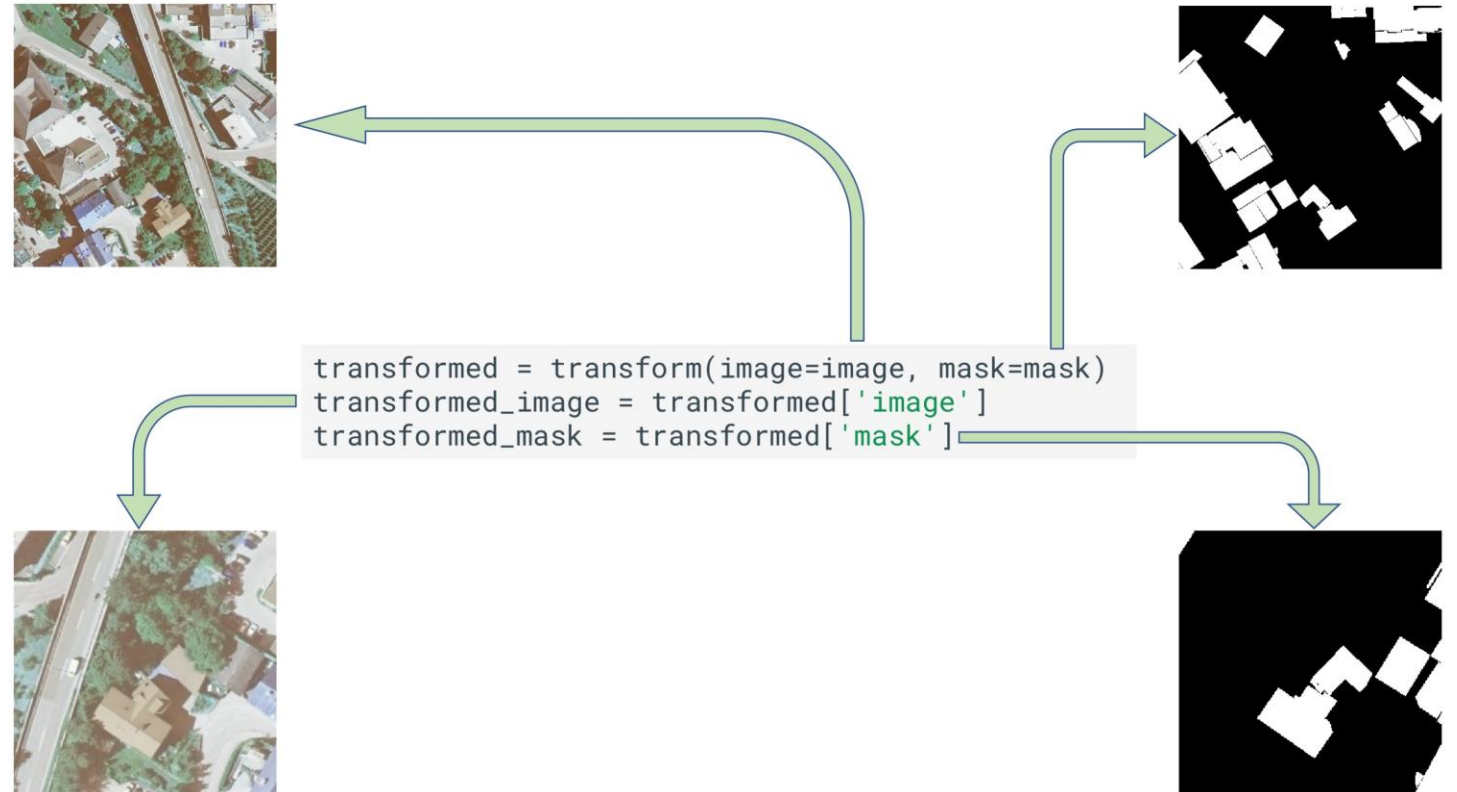


Image credit to alumentations tutorial:

[https://alumentations.ai/docs/getting\\_started/mask\\_augmentation/](https://alumentations.ai/docs/getting_started/mask_augmentation/)

# kornia



```
import torch  
import kornia
```

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

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



**FULLY DIFFERENTIABLE**

# kornia

```
# compute perspective transform
M = K.get_perspective_transform(points_src, points_dst)

# warp the original image by the found transform
img_warp = K.warp_perspective(img.float(), M, dsize=(h, w))
```

image source



image destination



Image credit to kornia tutorial:

[https://kornia-tutorials.readthedocs.io/en/latest/warp\\_perspective.html](https://kornia-tutorials.readthedocs.io/en/latest/warp_perspective.html)

# kornia

```
# create the operator
canny = K.filters.Canny()

# blur the image
x_magnitude, x_canny = canny(data.float())
```

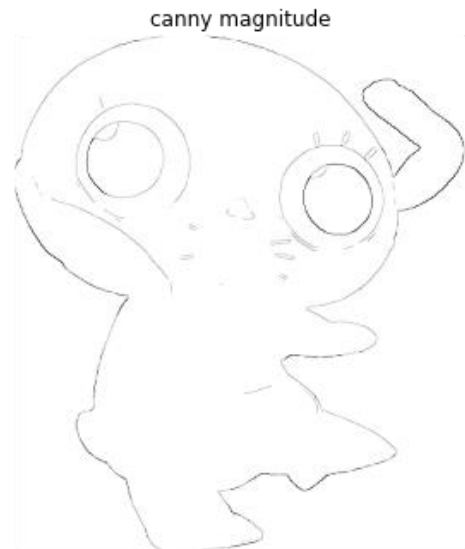


Image credit to kornia tutorial:

<https://kornia-tutorials.readthedocs.io/en/latest/canny.html>

# kornia

```
# create the operator
gauss = K.filters.GaussianBlur2d((11, 11), (10.5, 10.5))

# blur the image
x_blur = gauss(data.float())
```

image source

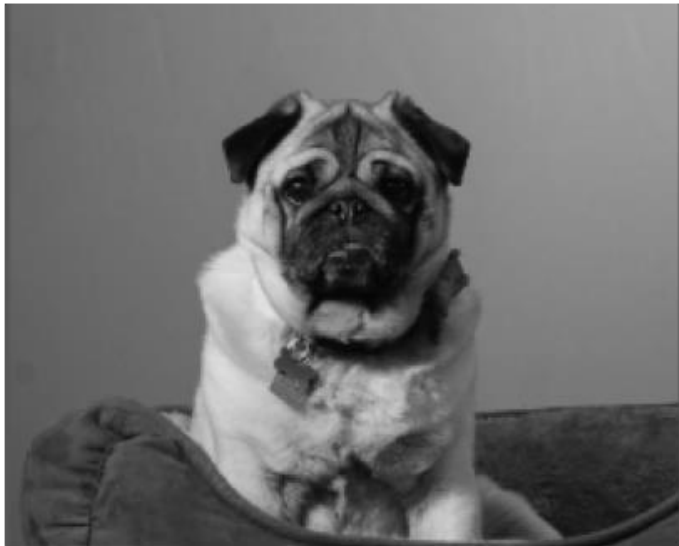


image blurred

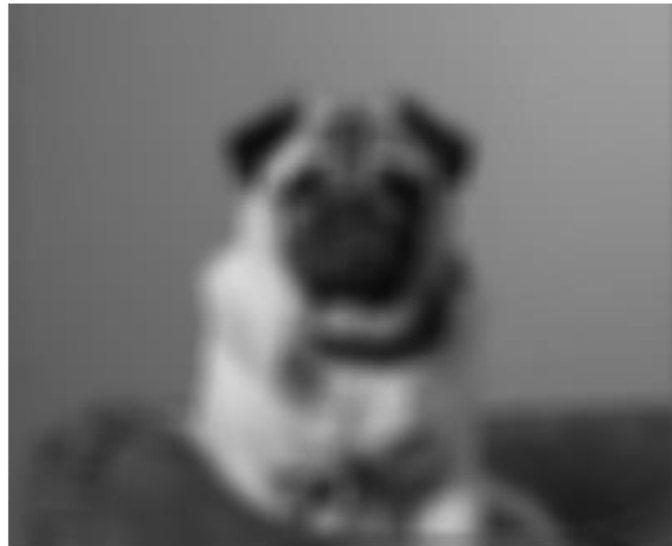


Image credit to kornia tutorial:

[https://kornia-tutorials.readthedocs.io/en/latest/gaussian\\_blur.html](https://kornia-tutorials.readthedocs.io/en/latest/gaussian_blur.html)

# kornia

```
# define sharpening mask
sharpen = kornia.filters.UnsharpMask((9, 9), (2.5, 2.5))

# create the sharpened image
sharpened_tensor = sharpen(data)

# get difference between original and sharpened image
difference = (sharpened_tensor - data).abs()
```

image source



sharpened



difference

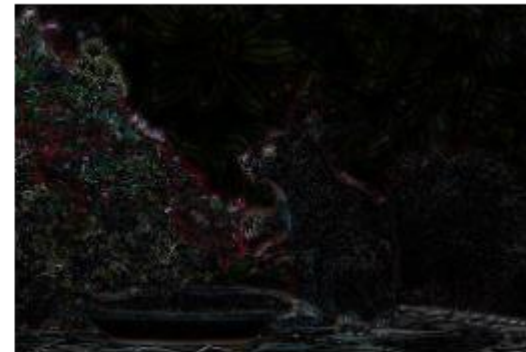


Image credit to kornia tutorial:

[https://kornia-tutorials.readthedocs.io/en/latest/unsharp\\_mask.html](https://kornia-tutorials.readthedocs.io/en/latest/unsharp_mask.html)



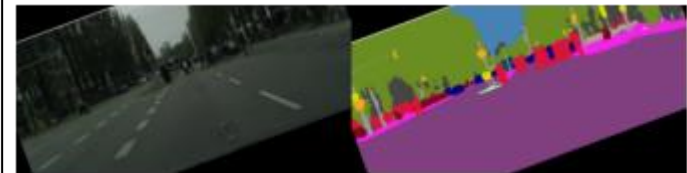
# kornia

```
import torch
import torch.nn as nn
import kornia as K

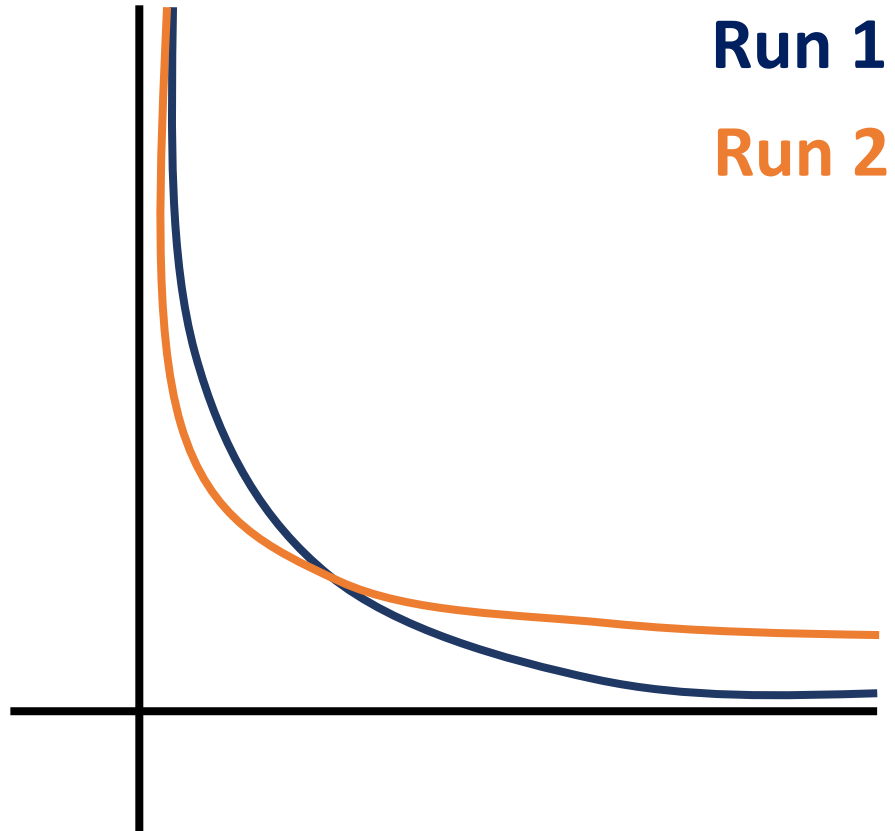
img = load_image(...) #BxCxHxW

aug = nn.Sequential(
    K.augmentations.ColorJitter(0.15, 0.25, 0.25, 0.25),
    K.augmentation.RandomAffine([-45., 45.], [0., 0.15],
                                [0.5, 1.5], [0., 0.15]),
)

out = aug(img) #BxCxHxW
```



# Reproducibility





# Reproducibility

```
# set random seeds
seed = 42
torch.manual_seed(seed)
random.seed(seed)
np.random.seed(seed)

# use deterministic algorithms only
torch.use_deterministic_algorithms(True)

# use known convolution algorithm in
cudnn
torch.backends.cudnn.benchmark = False
```

```
# fix workers randomness

def seed_worker(worker_id):
    worker_seed = torch.initial_seed() %
2**32
    numpy.random.seed(worker_seed)
    random.seed(worker_seed)

g = torch.Generator()
g.manual_seed(seed)

DataLoader(
    train_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g
)
```

Base on:

<https://pytorch.org/docs/stable/notes/randomness.html>



# Saving & Loading Models

## Serialize entire model

```
torch.save(model, "my_model.pth")  
...  
model = torch.load("my_model.pth")
```

### Pros:

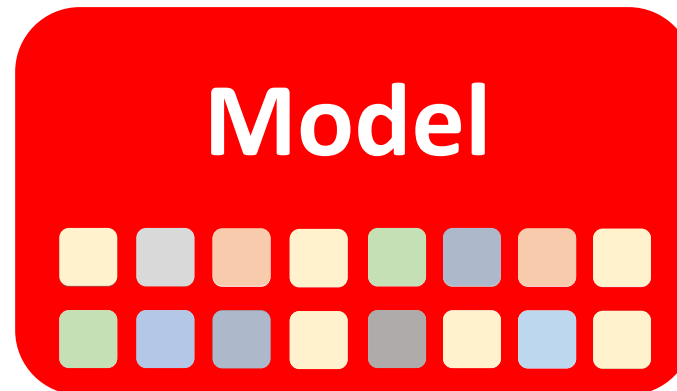
Simple

### Cons:

Can't change the model class

Can't continue training

```
# save an object to disk  
torch.save(object, path)  
  
# load an object from disk  
object = torch.load(path)
```



State dict



Base on:

[https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)

# Saving & Loading Models

```
# Define model
class TheModelClass(nn.Module):
    def __init__(self):
        super(TheModelClass, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        ...

# Initialize model
model = TheModelClass()
```

```
Model's state_dict:
conv1.weight torch.Size([6, 3, 5, 5])
conv1.bias torch.Size([6])
conv2.weight torch.Size([16, 6, 5, 5])
conv2.bias torch.Size([16])
fc1.weight torch.Size([120, 400])
fc1.bias torch.Size([120])
fc2.weight torch.Size([84, 120])
fc2.bias torch.Size([84])
fc3.weight torch.Size([10, 84])
fc3.bias torch.Size([10])
```

Base on:

[https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)



# Saving & Loading Models

## Saving the better way

```
# save the model's state dict
torch.save(model.state_dict(), "my_model.pth")

...

# create and load the model's state dict
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load("my_model.pth"))
```

```
# save an object to disk
torch.save(object, path)

# load an object from disk
torch.load(path)

# load the state dict to a
model
model.load_state_dict(sd)
```

Base on:

[https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)



# Saving & Loading Models

```
# Initialize optimizer
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

```
Optimizer's state_dict:
state      {}
param_groups  [{'lr': 0.001, 'momentum': 0.9, 'weight_decay': 0, ...}]
```

Base on:

[https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)



# Saving & Loading Models

## Saving for training

```
checkpoint = torch.save({'epoch': epoch,  
                        'model_sd': model.state_dict(),  
                        'opt_sd': optimizer.state_dict(),  
                        'loss': loss,  
                        ...}, 'checkpoint.pth')
```

```
model = TheModelClass(*args, **kwargs)  
optimizer = TheOptimizerClass(*args, **kwargs)  
  
checkpoint = torch.load('checkpoint.pth')  
model.load_state_dict(checkpoint['model_sd'])  
optimizer.load_state_dict(checkpoint['opt_sd'])  
epoch = checkpoint['epoch']  
loss = checkpoint['loss']  
# continue training
```

Base on:

[https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)



Next week:

# Implicit 3D Representations



# Sources

- Tensorboard

[https://pytorch.org/tutorials/intermediate/tensorboard\\_tutorial.html](https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html)

- <https://medium.com/@iamsdt/using-tensorboard-in-google-colab-with-pytorch-458f9bb95212>

- <https://towardsdatascience.com/a-complete-guide-to-using-tensorboard-with-pytorch-53cb2301e8c3>