diffusion

```
from fastai.basics import *
from fastai.vision.models.unet import *
from fastai.vision.all import *
from fastai.torch_basics import *
from denoising_diffusion_pytorch import Unet
from fastai.callback.wandb import _make_plt
from torch import autocast
from fastxtend.callback.ema import EMACallback
```

```
import wandb
wandb.init(reinit=True)
from fastai.callback.wandb import *
```

```
Failed to detect the name of this notebook, you can set it manually with the WANDB_NOTEBOOK_NAME environment variable to enable
wandb: Currently logged in as: marii. Use `wandb login --relogin` to force relogin
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
class DenoiseDiffusion:
    """
    ## Denoise Diffusion
    """

    def __init__(self, noise_schedule, device: torch.device):
        super().__init__()
        self.ns=noise_schedule

    def q_xt_x0(self, x0: torch.Tensor, t: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
        mean = self.gather(self.ns.alpha_bar, t) ** 0.5 * tensor(x0)
        var = 1 - self.gather(self.ns.alpha_bar, t)
        return mean, var
    def gather(self, consts: torch.Tensor, t: torch.Tensor):
        """Gather consts for $t$ and reshape to feature map shape"""
        c = consts.gather(-1, t)
        return c.reshape(-1, 1, 1, 1)

    def __call__(self, x0: torch.Tensor, t: torch.Tensor, eps: Optional[torch.Tensor] = None):
        if eps is None:
            eps = torch.randn_like(x0)
        mean, var = self.q_xt_x0(x0, t)
        return mean + (var ** 0.5) * eps

def p_sample_old(self, xt: torch.Tensor, t: torch.Tensor):
    """
    #### Sample from $\textcolor{lightgreen}{p_\theta}(x_{t-1}|x_t)$
    \begin{align}
    \textcolor{lightgreen}{p_\theta}(x_{t-1} | x_t) &= \mathcal{N}\big(x_{t-1};
    \textcolor{lightgreen}{\mu_\theta}(x_t, t), \sigma_t^2 \mathbf{I} \big) \\
    \textcolor{lightgreen}{\mu_\theta}(x_t, t)
      &= \frac{1}{\sqrt{\alpha_t}} \Big(x_t -
        \frac{\beta_t}{\sqrt{1-\bar\alpha_t}}\textcolor{lightgreen}{\epsilon_\theta}(x_t, t) \Big)
    \end{align}
    """
```

```
    # $\textcolor{lightgreen}{\epsilon_\theta}(x_t, t)$

    # NOTEDDDD REMOVED t

    eps_theta = self.eps_model(xt,t)
    # [gather](utils.html) $\bar\alpha_t$
    alpha_bar = gather(self.alpha_bar, t)
    # $\alpha_t$
    alpha = gather(self.alpha, t)
    # $\frac{\beta}{\sqrt{1-\bar\alpha_t}}$
    eps_coef = (1 - alpha) / (1 - alpha_bar) ** .5
    # $$\frac{1}{\sqrt{\alpha_t}} \Big(x_t -
    #      \frac{\beta_t}{\sqrt{1-\bar\alpha_t}}\textcolor{lightgreen}{\epsilon_\theta}(x_t, t) \Big)$$
    mean = 1 / (alpha ** 0.5) * (xt - eps_coef * eps_theta)
    # $\sigma^2$
    var = gather(self.sigma2, t)

    # $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    eps = torch.randn(xt.shape, device=xt.device)
    # Sample
    return mean + (var ** .5) * eps
```

```
class DiffusionSamplingTransform(ItemTransform):
    order=101
    "noise_sampler, or q_sampler, goes to noise at t=T. image sampler, p_sampler, goes to image at t=0"
    def __init__(self,noise_sampler,image_sampler):
        self.q_sample=noise_sampler
        self.p_sample=image_sampler
    def encodes(self,xy):
        x=xy[0]
        y=xy[-1]
        ts = xy[2][:,0]
        x_type=type(x)
        x=self.q_sample(x, x_type(ts), eps=y)
        return (x,*xy[1:-1],y)
    def decodes(self,xy):
        x=xy[0]
        ts = type(x)(xy[2])
        return (x,*xy[1:-1],self.p_sample(x.clone().detach().cuda(),ts.clone().detach().cuda()))
```

```
class LabelToNoise(ItemTransform):
    order=100
    def encodes(self,xy):
        y=xy[-1]
        return (*xy[:-1],retain_type(torch.randn(y.shape,device=y.device),old=y))
```

```
n_steps=1000
```

```
#path = untar_data(URLs.MNIST)
path = untar_data(URLs.CIFAR)
```

```
m=Unet(dim=192+192//8,channels=3,).cuda()
```

```
@typedispatch
def show_batch(x:tuple, y:TensorImage, samples, ctxs=None, max_n=10, nrows=None, ncols=None, figsize=None, **kwargs):
    if ctxs is None: ctxs = get_grid(3*min(len(samples), max_n), nrows=nrows, ncols=3, figsize=figsize, title='Input/Original
    ctxs[0::3] = [b.show(ctx=c, **kwargs)
                  for b,c,_ in zip(samples.itemgot(0),ctxs[0::3],range(max_n))]
    ctxs[0::3] = [b.show(ctx=c, **kwargs)
```

```
                for b,c,_ in zip(samples.itemgot(2),ctxs[0::3],range(max_n))]
    ctxs[1::3] = [b.show(ctx=c, **kwargs)
                for b,c,_ in zip(samples.itemgot(1),ctxs[1::3],range(max_n))]
    ctxs[2::3] = [b.show(ctx=c, **kwargs)
                for b,c,_ in zip(samples.itemgot(3),ctxs[2::3],range(max_n))]
    return ctxs
```

```
class SamplerDDPM():
    def __init__(self,model,betas=None):
        self.model=model
        if betas is None: self.betas = torch.linspace(0.0001, 0.02, 1000).cuda()
        self.alphas = 1. - self.betas
        self.a_bars = torch.cumprod(self.alphas, dim=0)

    def __call__(self,x_t,ts=None):
        T=self.betas.shape[0]
        if ts is None: ts=T*torch.ones([x_t.shape[0],1],device=x_t.device)
        ts=ts[:,0]
        noise = torch.randn_like(x_t)
        while((ts>0).any()):
            x,t,n=x_t[ts>0],ts[ts>0],noise[ts>0]
            with torch.no_grad():
                e = self.model(x,t)
            x_t[ts>0]=self._sample_t(x,e,t,n)
            ts[ts>0]-=1
        return x_t
    def _sample_t(self,x,es,t,n=None):
        t=t[...,None,None,None]
        #if(n is None): maybe try passing n so no regeneration?
        n=torch.randn_like(x)
        e,a,b=self._noise_at_t(es,t),self.alphas[t],self.betas[t]
        signal = (x - e) / (a ** 0.5)
        noise = b**.5 * n
        return signal + noise
    def _noise_at_t(self,es,t):
        eps_coef = (1 - self.alphas[t]) / (1 - self.a_bars[t]) ** .5
        return eps_coef* es
```

```
#This is for ddpm paper impl
#sampler=SamplerDDPM(m)
```

This is the ddpm dataloader

bs=128 diffusion = DenoiseDiffusion(m,n_steps,torch.device(0)) dls=DataBlock((ImageBlock(), ImageBlock(), TransformBlock(type_tfms=[DisplayedTransform(enc=lambda o: TensorCategory(o),dec=Category)]), ImageBlock()), n_inp=3, item_tfms=[Resize(32)], batch_tfms=(Normalize.from_stats(*cifar_stats),LabelToNoise,DiffusionSamplingTransform(diffusion,sampler)), get_items=get_image_files, get_x=[lambda x:x,lambda x:x, lambda x: torch.randint(0, n_steps, (1,), dtype=torch.long)], splitter=IndexSplitter(range(bs)),).dataloaders(path,bs=bs,val_bs=2*bs) #dls.show_batch()

```
class FlattenCallback(Callback):
    order=1 #after GatherPredsCallback
    #Maybe done for specific model?
    def before_batch(self):
        self.xbo=self.xb
        self.learn.xb=(self.xb[0],self.xb[-1].view(self.xb[-1].shape[::2]),)
    def after_batch(self):
        self.learn.xb=self.xbo
```

```
#[<class 'tuple'>, <class 'fastai.torch_core.TensorImage'>, <class 'fastcore.foundation.L'>, <class 'fastcore.foundation.L'>,
#This is a general wandb_process that just shows all samples and outs
@typedispatch
def wandb_process(x:tuple, y:TensorImage, samples, outs, preds):
    "Process `sample` and `out` depending on the type of `x/y`"
```

```
    res_sample = []
    for s,o in zip(samples, outs):
        #import pdb; pdb.set_trace()
        data = itertools.chain(s,o)
        caption = itertools.starmap('{}{}'.format,zip(
                                itertools.chain(len(s)*['Sample'],len(o)*['Out']),
                                itertools.chain(range(len(s)),range(len(o)))))
        ress = len(s)*len(o)*[res_sample]
        for t, capt, res in zip(data,caption,ress):
            if hasattr(t,'ndim') and t.ndim==3:
                t = t.permute(1,2,0)
                fig, ax = _make_plt(t)
                t.show(ctx=ax)
                res.append(wandb.Image(fig, caption=capt))
                plt.close(fig)

    return {"Samples":res_sample}
```

```
@typedispatch
def show_results(x:tuple, y:TensorImage, samples, outs, ctxs=None, max_n=10, figsize=None,**kwargs):
    if ctxs is None: ctxs = get_grid(3*min(len(samples), max_n), ncols=3, figsize=figsize, title='Input/Original/DenoisedImag
    ctxs[0::3] = [b.show(ctx=c, **kwargs) for b,c,_ in zip(samples.itemgot(0),ctxs[0::3],range(max_n))]
    ctxs[1::3] = [b.show(ctx=c, **kwargs) for b,c,_ in zip(samples.itemgot(1),ctxs[1::3],range(max_n))]
    ctxs[0::3] = [b.show(ctx=c, **kwargs) for b,c,_ in zip(samples.itemgot(2),ctxs[0::3],range(max_n))]
    ctxs[2::3] = [o.show(ctx=c, **kwargs) for o,c,_ in zip(outs.itemgot(0),ctxs[2::3],range(max_n))]
    return ctxs
```

```
# DDPM training
#learn = Learner(dls,m,MSELossFlat(),opt_func=Lamb,cbs=[WandbCallback(log_preds_every_epoch=True),FlattenCallback])
#learn = learn.to_fp16()
#learn.fit_flat_cos(1714//4+1,lr=2e-4,wd=0.)
```

```
#first ddpm train
#learn.save('2day_train')
```

```
class LinearNoiseSchedule:
    "Schedule like used in DDPM"
    def __init__(self,betas=None,n_steps=None):
        if betas is not None: self.n_steps=betas.shape[0]
        if n_steps is None: self.n_steps=1000
        if betas is None: self.betas = torch.linspace(0.0001, 0.02, self.n_steps).cuda()
        self.alphas = 1. - self.betas
        self.alpha_bar = torch.cumprod(self.alphas, dim=0)
```

```
class SamplerDDIM():
    def __init__(self,model,noise_schedule,betas=None,n_steps=50,predicts_x=False):
        self.n_steps=n_steps
        self.model=model
        self.ns=noise_schedule
        self.predicts_x=predicts_x

    def __call__(self,x_t,ts=None,n_steps=None,sample_n_steps=None):
        if(n_steps is None): n_steps=self.n_steps
        T=self.ns.betas.shape[0]
        fin_t=torch.zeros([x_t.shape[0]],device=x_t.device) if(sample_n_steps is None) else (ts-T/self.n_steps*sample_n_steps
        if ts is None: ts=(T-1)*torch.ones([x_t.shape[0],1],dtype=torch.long,device=x_t.device)
        ts=ts[:,0]
        t_sched=torch.linspace(T,0,n_steps+1,dtype=ts.dtype,device=x_t.device)
        while((ts>fin_t).any()):
            x,tm1,t=x_t[ts>fin_t],*self._next_ts(ts[ts>fin_t],t_sched)
            with autocast(device_type='cuda', dtype=x.dtype):
```

```
                with torch.no_grad():
                    e = self.model(x,t)
            if self.predicts_x: x_t[ts>fin_t]=self._sample_t(x,None,t,tm1,xs=e)
            else: x_t[ts>fin_t]=self._sample_t(x,e,t,tm1)
            ts[ts>fin_t]=tm1.squeeze()
        return x_t
    def _next_ts(self, ts, t_sched):
        return t_sched[(ts[...,None]>t_sched).max(dim=1).indices],ts
    def _sample_t(self,z,es,t,tm1,xs=None):
        a,a_tm1=self.ns.alpha_bar[t][...,None,None,None],self.ns.alpha_bar[tm1][...,None,None,None]
        if xs is None: xs=(z - (1-a)**.5 * es)/ (a ** .5)
        else: es=(z - (a)**.5 * xs)/(1-a)**.5
        signal = a_tm1**.5*(xs)
        noise = (1-a_tm1)**.5*es
        return signal + noise
#sampler=SamplerDDIM(m,LinearNoiseSchedule())
```

```
@patch
def decodes(self:DiffusionSamplingTransform,xy):
    x=xy[0]
    ts = type(x)(xy[2])
    return (x,*xy[1:-1],sampler(x.clone().detach().cuda(),ts.clone().detach().cuda()))
```

```
#learn.show_results()
```

Train X model

```
class LabelToX(ItemTransform):
    order=106
    def encodes(self,xy):
        return (*xy[:-1],xy[1])
```

```
s=0.008 #from Improved Denoising Diffusion Probabilistic Models
def cos_sched(at):
    return torch.cos((at+s)/(1+s)*torch.pi/2)**2
```

```
class CosineNoiseSchedule:
    "Schedule like used in DDPM"
    def __init__(self,betas=None,n_steps=None):
        if betas is not None: self.n_steps=betas.shape[0]
        if n_steps is None: self.n_steps=1000
        if betas is None: self.betas = torch.linspace(1e-4, 1, self.n_steps).cuda()
        self.alpha_bar = cos_sched(self.betas)
```

```
bs=128//2
diffusion = DenoiseDiffusion(CosineNoiseSchedule(),torch.device(0))
sampler=SamplerDDIM(m,CosineNoiseSchedule(),predicts_x=True)
dls=DataBlock((ImageBlock(),
               ImageBlock(),
               TransformBlock(type_tfms=[DisplayedTransform(enc=lambda o: TensorCategory(o),dec=Category)]),
               ImageBlock()),
          n_inp=3,
          item_tfms=[Resize(32)],
          batch_tfms=(Normalize.from_stats(*cifar_stats),LabelToNoise,DiffusionSamplingTransform(diffusion,sampler),LabelToX),
          get_items=get_image_files,
          get_x=[lambda x:x,lambda x:x,
                 lambda x: torch.randint(0, n_steps, (1,), dtype=torch.long)],
          splitter=IndexSplitter(range(bs)),
).dataloaders(path,bs=bs,val_bs=2*bs)
#dls.show_batch()
```

```
dls.show_batch()
```

Input/Original/Target

```
def mse_loss_weighted(ys,targ):
    return torch.mean(targ.w_sched[...,None] * ((ys - targ).flatten(start_dim=1) ** 2))
```

```
def snr(at): return at/(1-at)
```

```
def continuous_weights(at):
    weights = -snr(at[1:])/(snr(at[1:])-snr(at[:-1]))
    return torch.cat((weights[0:1],weights))
```

```
class WeightedCosSched(Callback):
    def after_pred(self):
        if(not hasattr(self,'ws')):
            self.ws = continuous_weights(CosineNoiseSchedule().alpha_bar).clip(min=1)
        ts=self.learn.xb[1].flatten()
        self.learn.yb[0].w_sched=self.ws[ts]
```

Run one epoch to see if decodes works, in wandb

```
learn = Learner(dls,m,mse_loss_weighted,opt_func=Adam,cbs=[WandbCallback(log_preds_every_epoch=True),FlattenCallback,Weighted
#learn = learn.to_fp16()
learn.fit_flat_cos(1714//16+1,lr=2e-4,wd=0.001,pct_start=0.5)
```

```
<IPython.core.display.HTML object>
```

epoch	train_loss	valid_loss	time
0	10.374235	9.685102	08:27
1	9.874603	9.149677	08:28
2	9.721998	8.351470	08:28
3	9.581279	8.406927	08:28
4	9.428663	8.331518	08:28
5	9.459892	7.865558	08:28
6	9.598307	8.643097	08:28
7	9.557909	8.566092	08:28
8	9.494190	7.545596	08:28
9	9.355718	8.486873	08:28
10	9.237833	7.707746	08:28
11	9.246161	8.340809	08:28
12	9.110999	8.409769	08:28
13	9.247369	8.166319	08:28
14	9.247720	9.650352	08:28
15	9.069467	9.489103	08:28
16	9.052581	8.766850	08:28
17	9.066344	7.911361	08:28
18	9.204024	8.637014	08:28
19	9.188643	9.829635	08:28
20	9.320750	9.271799	08:28
21	9.217371	8.455935	08:29
22	9.162722	8.511451	08:28
23	9.008821	7.789998	08:28
24	9.147062	8.096663	08:29
25	8.918532	8.440753	08:29
26	8.971040	10.096203	08:28
27	8.886042	8.815859	08:28
28	9.132473	7.929306	08:28
29	8.896766	8.834135	08:28
30	9.048318	7.950178	08:29
31	9.103962	8.043434	08:28
32	8.992520	7.101463	08:29
33	9.026378	9.409785	08:29
34	9.077930	7.918878	08:29
35	9.120553	9.848381	08:29

epoch	train_loss	valid_loss	time
36	9.123972	9.073471	08:33
37	8.919987	8.169960	08:29
38	8.812346	8.303776	08:29
39	9.013608	8.765829	08:29
40	9.036390	7.368114	08:29
41	8.951081	7.878605	08:29
42	8.738570	8.899674	08:29
43	8.914920	7.557009	08:29
44	8.891470	9.241924	08:29
45	8.964988	9.432287	08:29
46	8.850675	10.052511	08:29
47	8.934253	8.647129	08:29
48	8.798724	7.620637	08:29
49	8.727059	8.927790	08:29
50	8.989923	8.152671	08:29
51	9.231657	7.920894	08:29
52	8.929136	7.140281	08:29
53	8.773444	8.683802	08:30
54	8.818029	6.494678	08:35
55	8.904306	7.563613	08:29
56	8.929408	8.295150	08:30
57	8.717105	8.095920	08:30
58	8.942621	9.338118	08:30
59	8.859373	9.328197	08:30
60	8.941410	8.545383	08:30
61	9.004502	7.545311	08:30
62	8.918034	8.264477	08:36
63	8.941227	8.811317	08:30
64	8.794825	8.648201	08:30
65	8.738758	8.680571	08:30
66	8.947905	7.961948	08:30
67	8.847048	9.482048	08:30
68	8.903880	7.645896	08:30
69	8.919033	9.465273	08:30
70	8.896313	7.686443	08:30
71	8.919330	7.309587	08:38
72	8.816264	8.041633	08:30
73	8.741049	8.467232	08:30
74	8.766084	7.963510	08:30
75	8.989345	6.926301	08:30
76	8.761443	8.339436	08:30
77	8.824704	6.635482	08:31
78	8.737982	7.252566	08:31
79	8.784654	8.766117	08:30
80	8.700643	7.942421	08:31
81	8.629526	10.179396	08:39
82	8.793741	6.461315	08:31
83	8.686447	7.733202	08:31

```
wandb: Network error resolved after 0:00:11.192235, resuming normal operation.
wandb: Network error (ReadTimeout), entering retry loop.
```

```
#learn.save('diffusion2img_cos_sched')
```

```
Path('models/diffusion2img_cos_sched.pth')
```

Distillation

- Not using small steps, noise and x relationship breaks down
- So we predict x directly(or other options)

- Use DDIM sampler(deterministic)
- Student becomes the next teacher

```
m=Unet(dim=192+192//8,channels=3,)
load_model('models/diffusion2img_cos_sched.pth', m,opt=None,with_opt=False)
m=m.cuda()
```

```
diffusion = DenoiseDiffusion(CosineNoiseSchedule(),torch.device(0))
```

Duplicate Code? class Diffusion_Sampler(ItemTransform): order=101 def **init**(self,q_sampler,p_sampler): self.q_sampler=q_sampler self.p_sampler=p_sampler def encodes(self,xy): x=xy[0] y=xy[-1] ts = xy[2][:,0]#torch.randint(0, self.diffusion.n_steps, (x.shape[0],), device=x.device, dtype=torch.long) x_type=type(x) x=self.q_sampler(x, x_type(ts), eps=y) return (x,*xy[1:-1],y)* *def decodes(self,xy): x=xy[0] ts = type(x)(xy[2]) return (x,*xy[1:-1],self.p_sampler(x.clone().detach().cuda(),ts.clone().detach().cuda()))*

```
def z_to_x(sampler,z1,z2,ts,n_steps=2):
    a1=sampler.ns.alpha_bar[ts.flatten()]
    a2=sampler.ns.alpha_bar[(ts.flatten()-sampler.ns.alpha_bar.shape[0]/sampler.n_steps*n_steps).long().clamp(min=0)]
    a1,a2=type(z1)(a1),type(z1)(a2)
    sig2o1=(1-a2)**0.5/(1-a1)**0.5
    sig2o1=sig2o1
    noise_mean=a2-sig2o1*a1+0.000001
    x=z2-sig2o1[...,None,None,None]*z1
    return x/noise_mean[...,None,None,None]
```

```
class LabelToDDIM(ItemTransform):
    order=102
    def __init__(self,sampler,n_steps=2):
        self.sampler=sampler
        self.n_steps=n_steps
    def encodes(self,xy):
        x=xy[0]
        ts = xy[2]
        with torch.no_grad():
            x_type=type(x)
            with autocast(device_type='cuda', dtype=torch.float16):
                z=sampler(x,x_type(ts.clone()),sample_n_steps=self.n_steps)
        return (*xy[:-1],retain_type(z_to_x(sampler,x,z,ts),old=x))
#z_to_x(sampler,x,z,ts)
```

```
student=Unet(dim=192+192//8,channels=3,).cuda()
load_model('models/diffusion2img_cos_sched.pth', student,opt=None,with_opt=False)
```

```
#student=Unet(dim=192+192//8,channels=3,)
#load_model('models/distil1.pth', student,opt=None,with_opt=False)
#student=student.cuda()
```

```
sampler=SamplerDDIM(m,CosineNoiseSchedule(),n_steps=50,predicts_x=True)
```

```
bs=128//4
ddim_steps=50
sampler=SamplerDDIM(m,CosineNoiseSchedule(),n_steps=50,predicts_x=True)
student_sampler=SamplerDDIM(student,CosineNoiseSchedule(),n_steps=25,predicts_x=True)
dls=DataBlock((ImageBlock(),
               ImageBlock(),
               TransformBlock(type_tfms=[DisplayedTransform(enc=lambda o: TensorCategory(o),dec=Category)]),
               ImageBlock()),
          n_inp=3,
          item_tfms=[Resize(32)],
          batch_tfms=(Normalize.from_stats(*cifar_stats),LabelToNoise,DiffusionSamplingTransform(diffusion,student_sampler),I
          get_items=get_image_files,
          get_x=[lambda x:x,lambda x:x,
```

```
                    lambda x: 2000//(ddim_steps)*torch.randint(1, ddim_steps//2+1, (1,), dtype=torch.long)-1],
            splitter=IndexSplitter(range(bs)),
    ).dataloaders(path,bs=bs,val_bs=2*bs)
    dls.show_batch()
```

```
NameError: name 'student' is not defined
```

```
    def show_norm(x):
            return show_images(Normalize.from_stats(*cifar_stats).decode(x).clamp(0,1))
```

```
    learn = Learner(dls,student,mse_loss_weighted,opt_func=Adam,cbs=[WandbCallback(log_preds_every_epoch=True),FlattenCallback,We
```

```
    learn.fit_one_cycle(30,lr_max=2e-4,wd=0.001,pct_start=0.5)
```

```
<IPython.core.display.HTML object>
```

epoch	train_loss	valid_loss	time
0	0.111837	0.098590	18:11
1	0.018400	0.017847	18:08
2	0.007993	0.013191	18:12
3	0.004196	0.003947	18:10
4	0.003806	0.003563	18:11
5	0.002123	0.002521	18:04
6	0.002634	0.001477	18:12
7	0.003403	0.003208	18:12
8	0.001422	0.002685	18:07
9	0.001268	0.001141	18:13
10	0.001714	0.001555	18:08
11	0.002447	0.001783	18:12
12	0.000998	0.000468	18:09
13	0.001143	0.001345	18:10
14	0.001982	0.002736	18:08
15	0.002896	0.000767	18:13
16	0.000567	0.000420	18:12
17	0.000592	0.000203	18:11
18	0.000928	0.000619	18:11
19	0.000440	0.000199	18:10
20	0.000198	0.000088	18:08
21	0.000140	0.000143	18:10
22	0.000102	0.000082	18:16
23	0.000091	0.000085	18:07
24	0.000061	0.000044	18:12
25	0.000044	0.000039	18:07
26	0.000036	0.000032	18:12
27	0.000030	0.000026	18:08
28	0.000029	0.000023	18:08
29	0.000027	0.000026	18:14

```
wandb: Network error (ReadTimeout), entering retry loop.
```

```
    learn.show_results()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

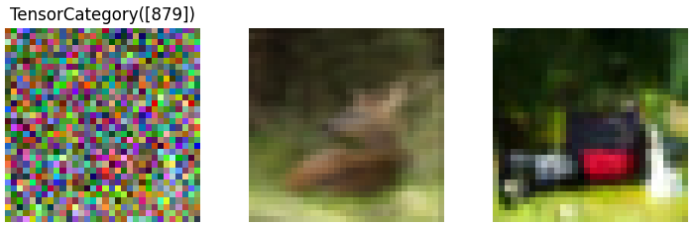
Input/Original/DenoisedImage

TensorCategory([159])

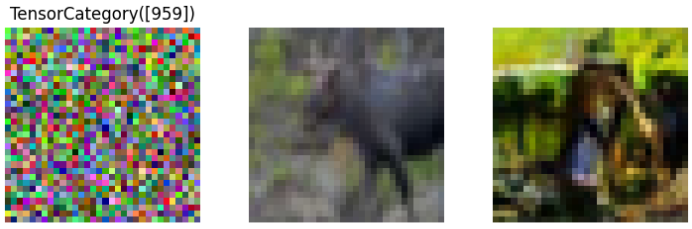
TensorCategory([919])

TensorCategory([519])

TensorCategory([879])



TensorCategory([959])



TensorCategory([119])

TensorCategory([519])

TensorCategory([599])

TensorCategory([759])

```
#learn.save('distil2')
```

```
Path('models/distil2.pth')
```

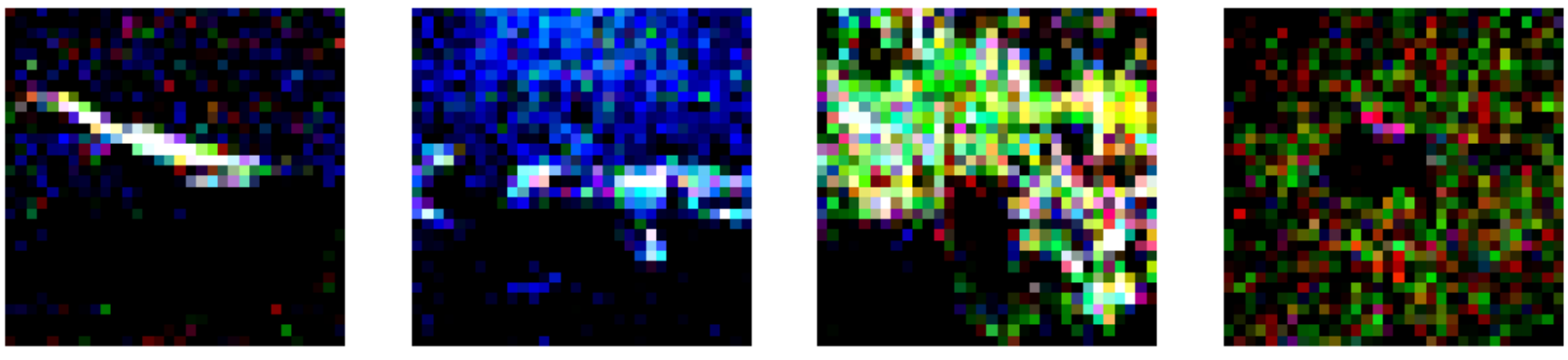
```
load_model('models/distil2.pth', student,opt=None,with_opt=False)
```

```
sample_imgs=dls.one_batch()
```

```
ts=999*torch.ones(4,1,dtype=torch.long).cuda()
xs=torch.randn(4,3,32,32).cuda()
zs=sampler(xs,ts,sample_n_steps=40)
```

```
def z_to_x(sampler,z1,z2,ts,n_steps=2):
    a1=sampler.ns.alpha_bar[ts.flatten()]
    a2=sampler.ns.alpha_bar[(ts.flatten()-sampler.ns.alpha_bar.shape[0]/sampler.n_steps*n_steps).long().clamp(min=0)]
    a1,a2=type(z1)(a1),type(z1)(a2)
    sig2o1=(1-a2)**0.5/(1-a1)**0.5
    noise_mean=a2-sig2o1*a1+0.000001
    x=z2-sig2o1[...,None,None,None]*z1
    return x/noise_mean[...,None,None,None]
```

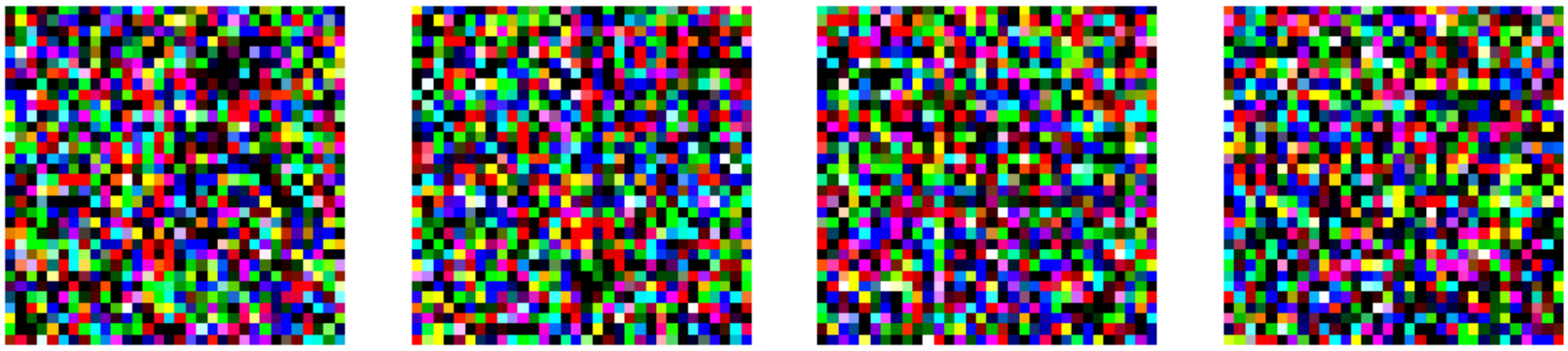
```
show_norm(zs)
```



```
show_norm(z_to_x(sampler,xs,zs,ts))
```

```
NameError: name 'show_norm' is not defined
```

```
show_norm(student_sampler(torch.randn(4,3,32,32).cuda(),999*torch.ones(4,1,dtype=torch.long).cuda()))
```



```
torch.randn(1,3,32,32).mean()
```

```
tensor(-0.0322)
```

```
sample_imgs[0][[0,17]][0].mean()
```

```
TensorImage(-0.0148, device='cuda:0')
```

```
show_norm(sample_imgs[3][[0,17]])
```

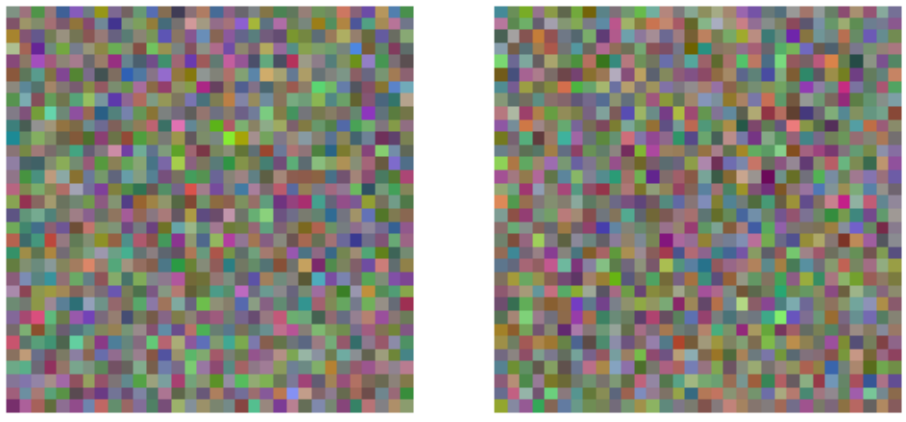
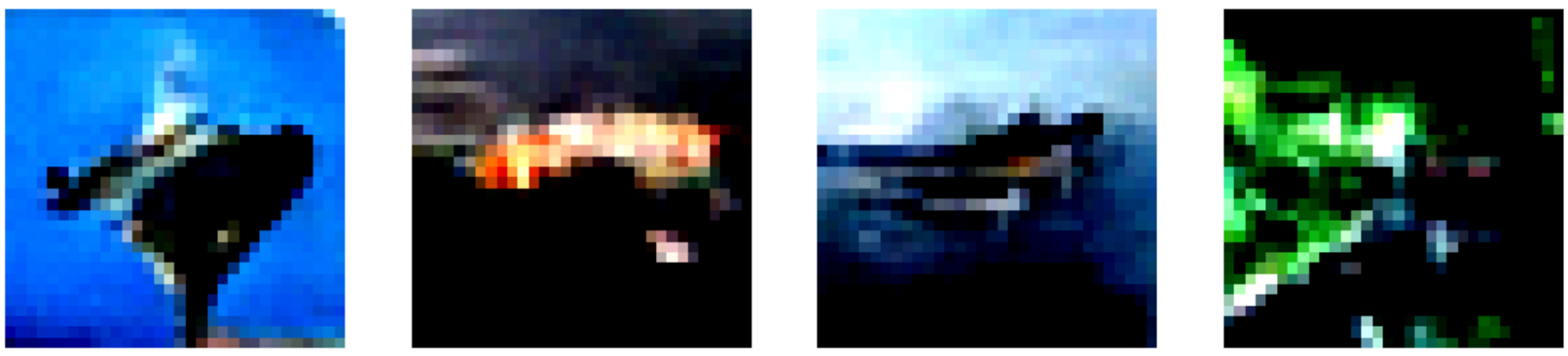


image-2-image

```
show_norm(sampler(torch.randn(4,3,32,32).cuda(),999*torch.ones(4,1,dtype=torch.long).cuda()))
```



```
star_img=Resize((32,32),method=ResizeMethod.Squish)(PILImage.create(Path('star.jpg')))
```

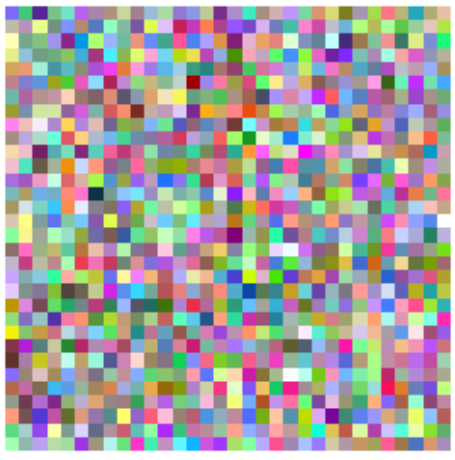
```
star_img
```

```
star_tensor=Normalize.from_stats(*cifar_stats)(IntToFloatTensor()(ToTensor()(star_img)).cuda())
```

```
show_norm(star_tensor)
```

```
noise=torch.randn_like(star_tensor)
```

```
show_norm(diffusion(star_tensor,800*torch.ones(1,1,dtype=torch.long)[:,0].cuda(),noise))
```



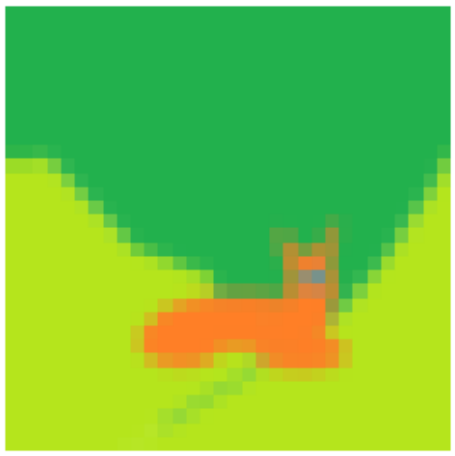
```
show_norm(sampler(diffusion(star_tensor,800*torch.ones(1,1,dtype=torch.long)[0].cuda(),noise),800*torch.ones(1,1,dtype=torch.
```

Cat Thing

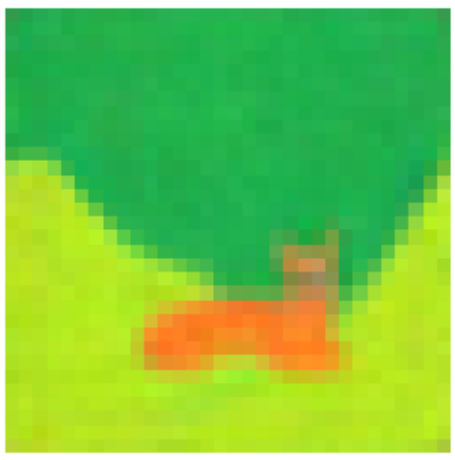
```
cat_img=Resize((32,32),method=ResizeMethod.Squish)(PILImage.create(Path('cat_thing.jpg')))
```

```
cat_img
```

```
cat_tensor=Normalize.from_stats(*cifar_stats)(IntToFloatTensor()(ToTensor()(cat_img)).cuda())
show_norm(cat_tensor)
```



```
noise=torch.randn_like(cat_tensor)
show_norm(sampler(diffusion(cat_tensor,100*torch.ones(1,1,dtype=torch.long)[0].cuda(),noise),100*torch.ones(1,1,dtype=torch.l
```



```
SamplerDDIM??
```

```
wandb: Network error (ReadTimeout), entering retry loop.
wandb: ERROR Error while calling W&B API: internal database error (<Response [500]>)
wandb: Network error (ReadTimeout), entering retry loop.
wandb: Network error (ReadTimeout), entering retry loop.
```

```
#learn.save('distil1')
```

```
Path('models/distil1.pth')
```

::: {.cell 0=‘h’ 1=‘i’ 2=‘d’ 3=‘e’ execution_count=20}

```
#from nbdev import nbdev_export
#nbdev_export()
```

:::

```
#from datasets import load_dataset
#load_dataset("huggan/CelebA-HQ")
```