Super resolution

In [1]:
```
import fastai
from fastai.vision import *
from fastai.callbacks import *

from torchvision.models import vgg16_bn
```

In [2]:
```
import os
from os import listdir
from os.path import isfile, join
import shutil

```

In [3]:
```
# styled_files = [styled_file.split('style_')[1] for styled_file in os.listd
# unprocessed_content = [content_img for content_img in os.listdir(content_i

# for content_img in [content_img for content_img in unprocessed_content if
#     print(content_img)
#     out_img = style_transfer(content_img)
#     out_img.save(output_dir + 'style_' + content_img)
```

In [4]:
```
styled_src = Path('../../TampaAI/N_A_A_Style/output/pets_monet_sunset')
full_size_img_src_dir = '/home/ml1/.fastai/data/oxford-iiit-pet/images/'
img_dir = Path('images')
full_size_dir = img_dir/'orig'
styled_dir = img_dir/'styled'
small_96_dir = img_dir/'small-96'
small_256_dir = img_dir/'small-256'
```

In [5]:
```
styled_files = [f for f in listdir(styled_src) if isfile(join(styled_src, f)
root_of_styled_files = [styled_file.split('style_')[1] for styled_file in st
root_of_styled_files
```

Out[5]:
```
['great_pyrenees_177.jpg',
 'newfoundland_181.jpg',
 'Bombay_37.jpg',
 'staffordshire_bull_terrier_137.jpg',
 'japanese_chin_36.jpg',
 'Persian_138.jpg',
 'Ragdoll_90.jpg',
 'scottish_terrier_198.jpg',
 'boxer_197.jpg',
 'Birman_183.jpg',
 'Siamese_30.jpg',
 'keeshond_65.jpg',
 'staffordshire_bull_terrier_73.jpg',
 'Maine_Coon_108.jpg',
 'staffordshire_bull_terrier_107.jpg',
 'english_cocker_spaniel_117.jpg',
 'pug_170.jpg',
 'Siamese_122.jpg',
 'miniature_pinscher_4.jpg',
```

In [6]:
```
content_img_dir = '/home/ml1/.fastai/data/oxford-iiit-pet/images/'
# processed_content = [content_img for content_img in os.listdir(content_img
# processed_content
```

In [7]:
```
for file_name in root_of_styled_files:
    full_file_name = os.path.join(full_size_img_src_dir, file_name)
    shutil.copy(full_file_name, full_size_dir)
```

In [8]:
```
for file_name in root_of_styled_files:
    full_file_name = os.path.join(styled_src, 'style_'+file_name)
    shutil.copy(full_file_name, styled_dir)
    style_file_name = 'style_' + file_name
    shutil.move(styled_dir/style_file_name, styled_dir/file_name )
```

In [ ]:
```

```

In [9]:
```
# from pathlib import Path
# import glob
# # path = untar_data(URLs.PETS)
# styled_src = Path('../..')
# for file_name in glob.glob(styled_src):
#     print(file_name)

```

In [10]:
```
# path_hr = path/'images'
# path_lr = path/'small-96'
# path_mr = path/'small-256'
path_hr = styled_dir
path_lr = small_96_dir
path_mr = small_256_dir
```

In [11]:
```
il = ImageItemList.from_folder(full_size_dir)
```

In [12]:
```
# f_list = il.to_df().head()
```

In [13]:
```
def resize_one(fn,i):
#     dest = path_lr/fn.relative_to(path_hr)
    dest = path_lr/fn.relative_to(full_size_dir)
    dest.parent.mkdir(parents=True, exist_ok=True)
    img = PIL.Image.open(fn)
    targ_sz = resize_to(img, 96, use_min=True)
    img = img.resize(targ_sz, resample=PIL.Image.BILINEAR).convert('RGB')
    img.save(dest, quality=60)
```

In [14]:
```
# to create smaller images, uncomment the next line when you run this the fi
# parallel(resize_one, il.items)
```

In [15]:
```
bs,size=32,128
arch = models.resnet34

src = ImageImageList.from_folder(path_lr).random_split_by_pct(0.1, seed=42)
```

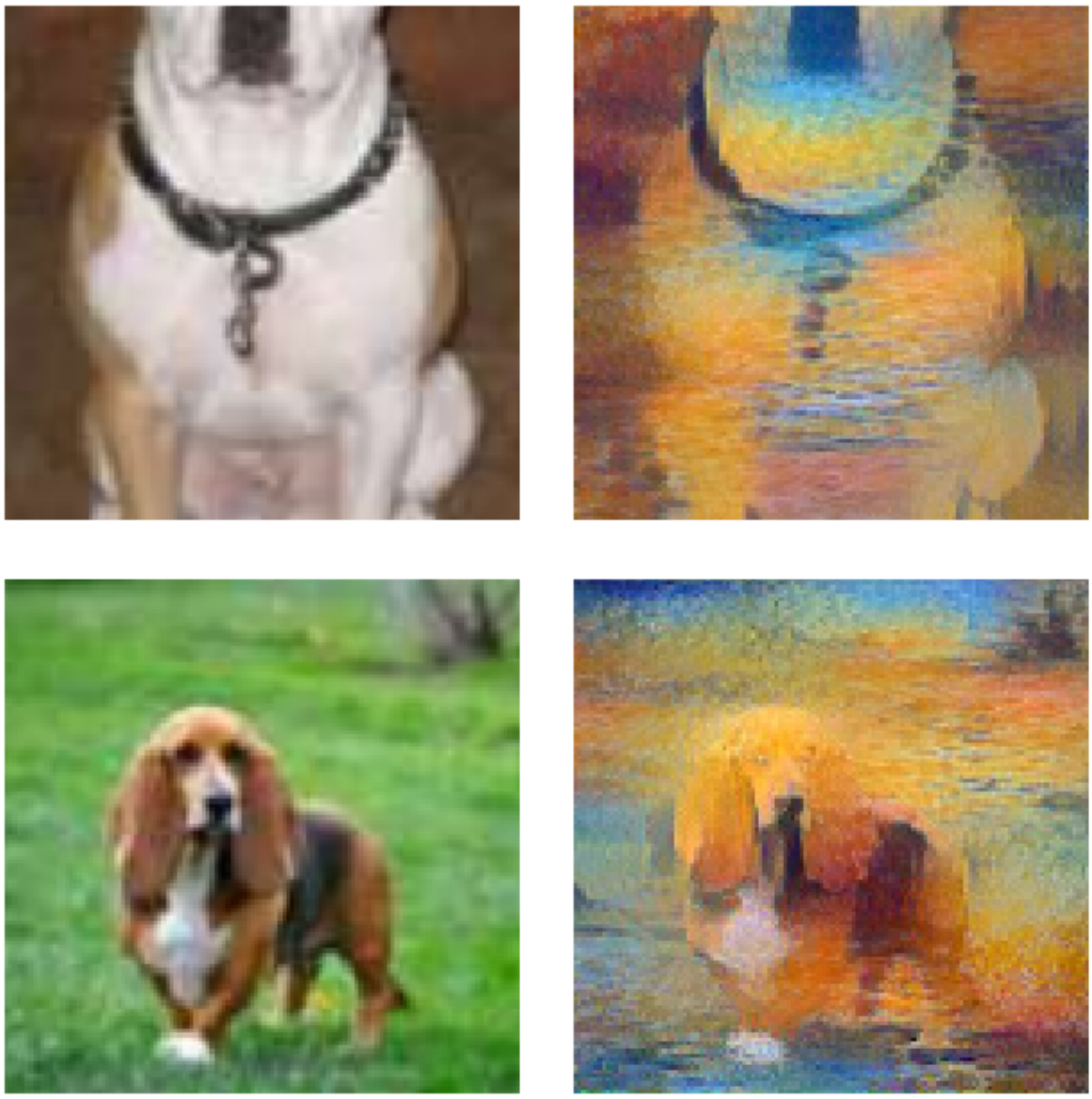
In [16]:
```
src
```

Out[16]:
```
ItemLists;

Train: ImageImageList (2753 items)
[Image (3, 96, 128), Image (3, 96, 128), Image (3, 96, 125), Image (3, 96, 12
8), Image (3, 96, 128)]...
Path: images/small-96;

Valid: ImageImageList (305 items)
[Image (3, 175, 96), Image (3, 96, 107), Image (3, 96, 125), Image (3, 96, 12
8), Image (3, 96, 144)]...
Path: images/small-96;

Test: None
```

In [17]:
```
def get_data(bs,size):
    data = (src.label_from_func(lambda x: path_hr/x.name)
           .transform(get_transforms(max_zoom=2.), size=size, tfm_y=True)
           .databunch(bs=bs).normalize(imagenet_stats, do_y=True))

    data.c = 3
    return data
```

In [18]:
```
1 data = get_data(bs,size)
```

In [19]:
```
1 data.show_batch(ds_type=DatasetType.Valid, rows=2, figsize=(9,9))
```



Feature loss

In [20]:
```
1 t = data.valid_ds[0][1].data
2 t = torch.stack([t,t])
```

In [21]:
```
1 def gram_matrix(x):
2     n,c,h,w = x.size()
3     x = x.view(n, c, -1)
4     return (x @ x.transpose(1,2))/(c*h*w)
```

In [22]:
```
gram_matrix(t)
```

Out[22]:
```
tensor([[[0.1315, 0.1074, 0.0764],
         [0.1074, 0.0923, 0.0687],
         [0.0764, 0.0687, 0.0562]],

        [[0.1315, 0.1074, 0.0764],
         [0.1074, 0.0923, 0.0687],
         [0.0764, 0.0687, 0.0562]]])
```

In [23]:
```
base_loss = F.l1_loss
```

In [24]:
```
vgg_m = vgg16_bn(True).features.cuda().eval()
requires_grad(vgg_m, False)
```

In [25]:
```
blocks = [i-1 for i,o in enumerate(children(vgg_m)) if isinstance(o,nn.MaxPo
blocks, [vgg_m[i] for i in blocks]
```

Out[25]:
```
([5, 12, 22, 32, 42],
 [ReLU(inplace), ReLU(inplace), ReLU(inplace), ReLU(inplace), ReLU(inplace)])
```

In [26]:
```
class FeatureLoss(nn.Module):
    def __init__(self, m_feat, layer_ids, layer_wgts):
        super().__init__()
        self.m_feat = m_feat
        self.loss_features = [self.m_feat[i] for i in layer_ids]
        self.hooks = hook_outputs(self.loss_features, detach=False)
        self.wgts = layer_wgts
        self.metric_names = ['pixel',] + [f'feat_{i}' for i in range(len(lay
              ] + [f'gram_{i}' for i in range(len(layer_ids))]

    def make_features(self, x, clone=False):
        self.m_feat(x)
        return [(o.clone() if clone else o) for o in self.hooks.stored]

    def forward(self, input, target):
        out_feat = self.make_features(target, clone=True)
        in_feat = self.make_features(input)
        self.feat_losses = [base_loss(input,target)]
        self.feat_losses += [base_loss(f_in, f_out)*w
                             for f_in, f_out, w in zip(in_feat, out_feat, se
        self.feat_losses += [base_loss(gram_matrix(f_in), gram_matrix(f_out)
                             for f_in, f_out, w in zip(in_feat, out_feat, se
        self.metrics = dict(zip(self.metric_names, self.feat_losses))
        return sum(self.feat_losses)

    def __del__(self): self.hooks.remove()
```

In [27]:
```
feat_loss = FeatureLoss(vgg_m, blocks[2:5], [5,15,2])
```

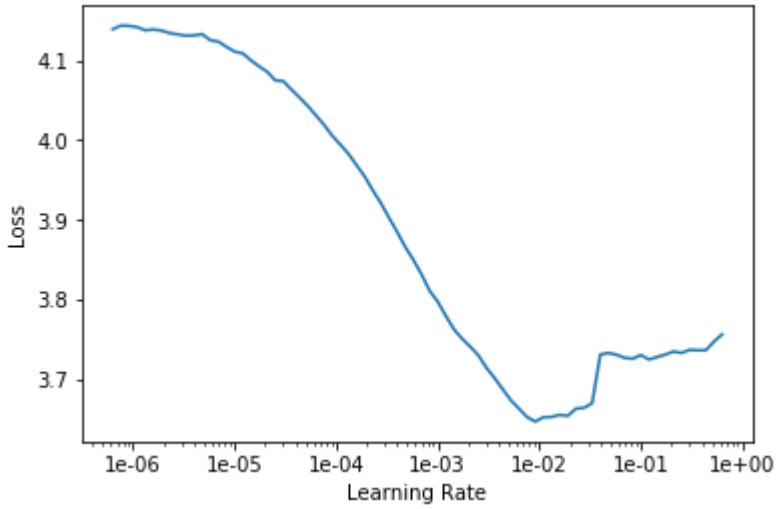
Train

In [28]:
```
wd = 1e-3
learn = unet_learner(data, arch, wd=wd, loss_func=feat_loss, callback_fns=Lo
                     blur=True, norm_type=NormType.Weight)
gc.collect();
learn.data.batch_size
```

Out[28]: 32

In [29]:
```
learn.lr_find()
learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



In [30]:
```
lr = 1e-3
```

In [31]:
```
def do_fit(save_name, lrs=slice(lr), pct_start=0.9):
    learn.fit_one_cycle(10, lrs, pct_start=pct_start)
    learn.save(save_name)
    learn.show_results(rows=1, imgsize=5)
```

In [32]:
```
1 do_fit('1a', slice(lr*10))
```

Total time: 04:27

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0
1	2.888878	2.802722	0.400150	0.311663	0.346429	0.139423	0.516962
2	2.760338	2.711827	0.380862	0.307217	0.342594	0.137599	0.483044
3	2.710657	2.681858	0.385092	0.307107	0.336578	0.135424	0.471399
4	2.684432	2.710310	0.432943	0.308145	0.332644	0.134307	0.467406
5	2.672602	2.658380	0.393814	0.303303	0.332759	0.132464	0.472350
6	2.645839	2.582036	0.370325	0.300303	0.327458	0.131722	0.448097
7	2.623099	2.554564	0.368303	0.299261	0.327110	0.131516	0.430205
8	2.611051	2.564203	0.379349	0.300568	0.326092	0.130733	0.432082
9	2.586885	2.528744	0.371698	0.299624	0.323526	0.130359	0.421029
10	2.542103	2.467600	0.349594	0.297600	0.321580	0.128988	0.405693

Input / Prediction / Target

In [33]:
```
1 learn.unfreeze()
```

In [34]:
```
1 do_fit('1b', slice(1e-5,lr))
```

Total time: 04:38

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0
1	2.508552	2.470243	0.348827	0.298201	0.321827	0.129125	0.405801
2	2.502695	2.466722	0.347024	0.297699	0.321056	0.129063	0.406118
3	2.494808	2.463147	0.347236	0.297211	0.321384	0.129191	0.405461
4	2.498787	2.449929	0.345081	0.297055	0.320709	0.128554	0.400691
5	2.492712	2.469727	0.345163	0.296634	0.320326	0.128987	0.409961
6	2.487676	2.465504	0.346660	0.296866	0.320187	0.129350	0.406975
7	2.476392	2.437048	0.343815	0.295747	0.318131	0.127907	0.397501
8	2.477364	2.429764	0.342185	0.296188	0.318579	0.128037	0.394582
9	2.476491	2.434588	0.340103	0.295186	0.318055	0.127891	0.399896
10	2.466644	2.422903	0.338207	0.295324	0.317424	0.127604	0.394378

Input / Prediction / Target

In [35]:
```
data = get_data(12,size*2)
data
```

Out[35]:
```
ImageDataBunch;

Train: LabelList
y: ImageItemList (2753 items)
[Image (3, 256, 342), Image (3, 256, 341), Image (3, 256, 335), Image (3, 256,
341), Image (3, 256, 341)]...
Path: images/small-96
x: ImageImageList (2753 items)
[Image (3, 96, 128), Image (3, 96, 128), Image (3, 96, 125), Image (3, 96, 12
8), Image (3, 96, 128)]...
Path: images/small-96;

Valid: LabelList
y: ImageItemList (305 items)
[Image (3, 468, 256), Image (3, 256, 287), Image (3, 256, 335), Image (3, 256,
341), Image (3, 256, 384)]...
Path: images/small-96
x: ImageImageList (305 items)
[Image (3, 175, 96), Image (3, 96, 107), Image (3, 96, 125), Image (3, 96, 12
8), Image (3, 96, 144)]...
Path: images/small-96;

Test: None
```

In [36]:
```
learn.data = data
learn.freeze()
gc.collect()
```

Out[36]:
```
19688
```

In [37]:
```
learn
```

Out[37]:
```
Learner(data=ImageDataBunch;

Train: LabelList
y: ImageItemList (2753 items)
[Image (3, 256, 342), Image (3, 256, 341), Image (3, 256, 335), Image (3, 25
6, 341), Image (3, 256, 341)]...
Path: images/small-96
x: ImageImageList (2753 items)
[Image (3, 96, 128), Image (3, 96, 128), Image (3, 96, 125), Image (3, 96, 12
8), Image (3, 96, 128)]...
Path: images/small-96;

Valid: LabelList
y: ImageItemList (305 items)
[Image (3, 468, 256), Image (3, 256, 287), Image (3, 256, 335), Image (3, 25
6, 341), Image (3, 256, 384)]...
Path: images/small-96
x: ImageImageList (305 items)
[Image (3, 175, 96), Image (3, 96, 107), Image (3, 96, 125), Image (3, 96, 12
```

In [38]:
```
1 learn.data.batch_size
```

Out[38]: 12

In [39]:
```
1 learn.data.batch_size = learn.data.batch_size//2
2 learn.data.batch_size
```

Out[39]: 6

In [40]:
```
1 learn.load('1b');
```

In [41]:
```
1 gc.collect()
```

Out[41]: 0

In [42]:
```
1 do_fit('2a')
```

3	1.714106	1.718336	0.356518	0.307389	0.269749	0.094273	0.258037
4	1.688799	1.716100	0.356088	0.307136	0.269905	0.094825	0.256647
5	1.684214	1.706422	0.353547	0.307031	0.268583	0.094448	0.252158
6	1.683811	1.709307	0.358351	0.306697	0.268672	0.093963	0.255058
7	1.670471	1.685609	0.347387	0.305498	0.267441	0.094377	0.247408
8	1.666559	1.681222	0.348621	0.305030	0.267201	0.093405	0.246055
9	1.668058	1.673358	0.344755	0.305114	0.268340	0.094490	0.242541
10	1.641732	1.659688	0.339444	0.305622	0.265179	0.093266	0.239610

Input / Prediction / Target

In [43]:
```
1 learn.unfreeze()
```

In [44]:
```
1 do_fit('2b', slice(1e-6,1e-4), pct_start=0.3)
```

Total time: 20:30

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0
1	1.633440	1.661697	0.338929	0.305611	0.264935	0.093316	0.240834
2	1.636672	1.662439	0.338239	0.305816	0.265915	0.093722	0.240812
3	1.630876	1.651365	0.336715	0.305728	0.264273	0.092895	0.236705
4	1.630698	1.652336	0.336751	0.305440	0.263674	0.092918	0.237603
5	1.628027	1.652598	0.335233	0.305314	0.264373	0.092932	0.238961
6	1.622456	1.652236	0.334720	0.305754	0.264049	0.092988	0.239227
7	1.614159	1.647799	0.333673	0.304889	0.264373	0.093023	0.237176
8	1.613826	1.650696	0.333413	0.305395	0.264024	0.092950	0.238165
9	1.616199	1.646712	0.332440	0.305456	0.264184	0.092992	0.237340
10	1.623145	1.649282	0.333545	0.305586	0.263895	0.092899	0.237928

Input / Prediction / Target

Test

In [ ]:
```
1 learn = None
2 gc.collect();
```

In [ ]:
```
1 256/320*1024
```

In [ ]:
```
1 256/320*1600
```

In [ ]:
```
1 learn = unet_learner(data, arch, loss_func=F.l1_loss, blur=True, norm_type=N
```

In [ ]:
```
data_mr = (ImageImageList.from_folder(path_mr).random_split_by_pct(0.1, seed
          .label_from_func(lambda x: path_hr/x.name)
          .transform(get_transforms(), size=(1280,1600), tfm_y=True)
          .databunch(bs=1).normalize(imagenet_stats, do_y=True))
data_mr.c = 3
```

In [ ]:
```
learn.load('2b');
```

In [ ]:
```
learn.data = data_mr
```

In [ ]:
```
fn = data_mr.valid_ds.x.items[0]; fn
```

In [ ]:
```
img = open_image(fn); img.shape
```

In [ ]:
```
p,img_hr,b = learn.predict(img)
```

In [ ]:
```
show_image(img, figsize=(18,15), interpolation='nearest');
```

In [ ]:
```
Image(img_hr).show(figsize=(18,15))
```

In [ ]:
```

```