Super resolution

In [ ]:

```
import fastai
from fastai.vision import *
from fastai.callbacks import *

from torchvision.models import vgg16_bn
```

In [ ]:

```
path = untar_data(URLs.PETS)
path_hr = path/'images'
path_lr = path/'small-96'
path_mr = path/'small-256'
```

In [ ]:

```
il = ImageItemList.from_folder(path_hr)
```

In [ ]:

```
def resize_one(fn,i):
    dest = path_lr/fn.relative_to(path_hr)
    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 [ ]:

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

In [ ]:

```
bs,size=32,128
arch = models.resnet34

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

In [ ]:

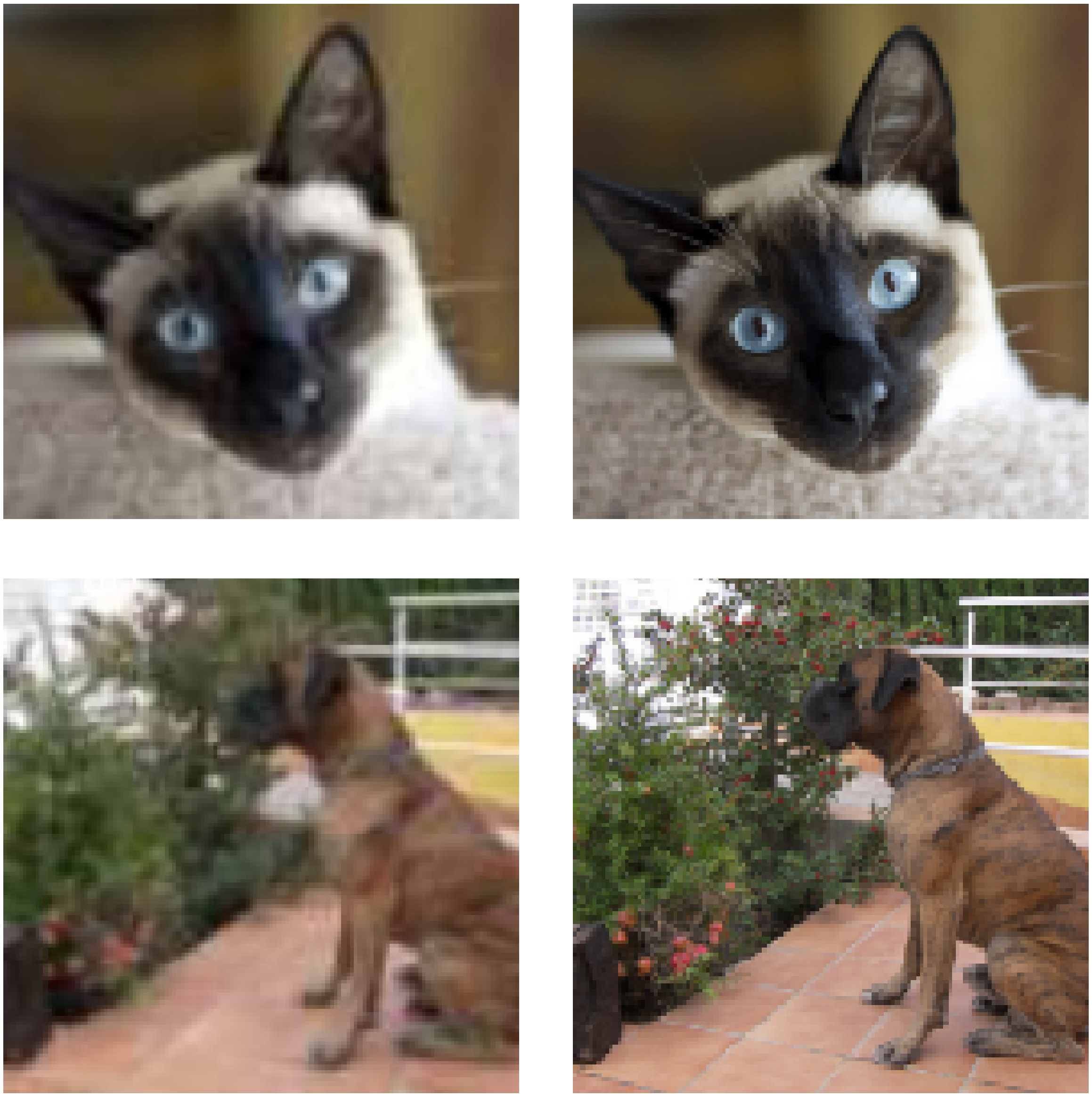
```
def get_data(bs,size):
    data = (src.label_from_func(lambda x: path_hr/x.name)
           .transform(get_transforms(max_zoom=2.), size=size, tfm_y=True)
           .databunch(bs=bs).normalize(imagenet_stats, do_y=True))

    data.c = 3
    return data
```

In [ ]:

```
data = get_data(bs,size)
```

In [ ]:

```
data.show_batch(ds_type=DatasetType.Valid, rows=2, figsize=(9,9))
```



Feature loss

In [ ]:

```
t = data.valid_ds[0][1].data
t = torch.stack([t,t])
```

In [ ]:

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

In [ ]:

```
gram_matrix(t)
```

Out[ ]:

```
tensor([[[0.0759, 0.0711, 0.0643],
         [0.0711, 0.0672, 0.0614],
         [0.0643, 0.0614, 0.0573]],

        [[0.0759, 0.0711, 0.0643],
         [0.0711, 0.0672, 0.0614],
         [0.0643, 0.0614, 0.0573]]])
```

In [ ]:

```
base_loss = F.l1_loss
```

In [ ]:

```
vgg_m = vgg16_bn(True).features.cuda().eval()
requires_grad(vgg_m, False)
```

In [ ]:

```
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[ ]:

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

In [ ]:

```
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 [ ]:

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

Train

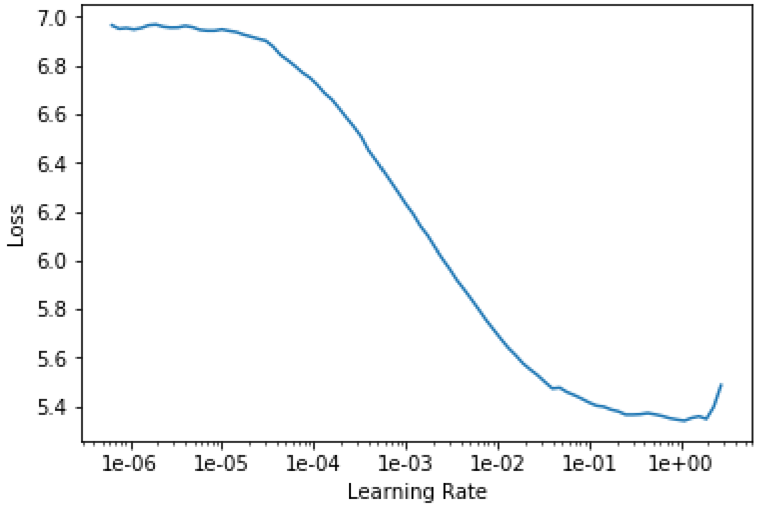
In [ ]:

```
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();
```

In [ ]:

```
learn.lr_find()
learn.recorder.plot()
```

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



In [ ]:

```
lr = 1e-3
```

In [ ]:

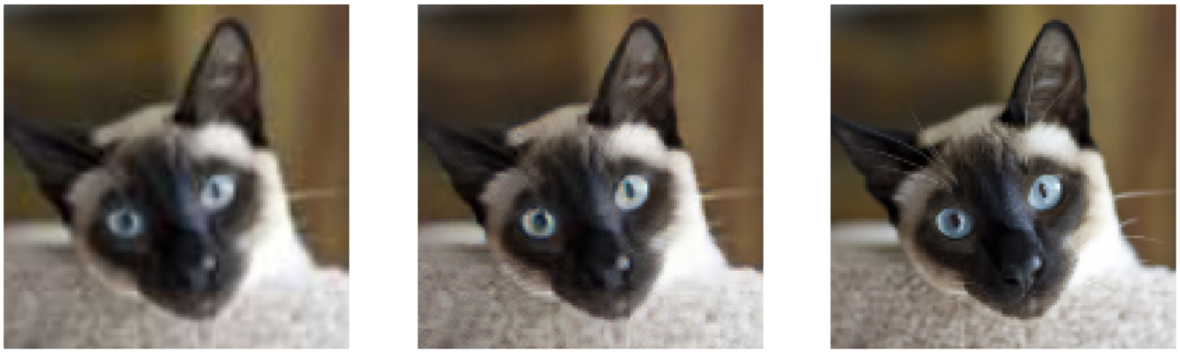
```
def do_fit(save_name, lrs=slice(lr), pct_start=0.9):
    learn.fit_one_cycle(10, lrs, pct_start=pct_start)
    learn.save(save_name)
    learn.show_results(rows=1, imgsize=5)
```

In [ ]:

```
do_fit('1a', slice(lr*10))
```

```
Total time: 11:16

epoch  train_loss  valid_loss     pixel    feat_0    feat_1    feat_2    gram_0
    1    3.873667    3.759143  0.144560  0.229806  0.314573  0.226204  0.552578
    2    3.756051    3.650393  0.145068  0.228509  0.308807  0.218000  0.534508
    3    3.688726    3.628370  0.157359  0.226753  0.304955  0.215417  0.522482
    4    3.628276    3.524132  0.145285  0.225455  0.300169  0.211110  0.497361
    5    3.586930    3.422895  0.145161  0.224946  0.294471  0.205117  0.472445
    6    3.528042    3.394804  0.142262  0.220709  0.289961  0.201980  0.478097
    7    3.522416    3.361185  0.139654  0.220379  0.288046  0.200114  0.471151
    8    3.469142    3.338554  0.142112  0.219271  0.287442  0.199255  0.462878
    9    3.418641    3.318710  0.146493  0.219915  0.284979  0.197340  0.455503
   10    3.356641    3.187186  0.135588  0.215685  0.277398  0.189562  0.432491
```

Input / Prediction / Target



In [ ]:

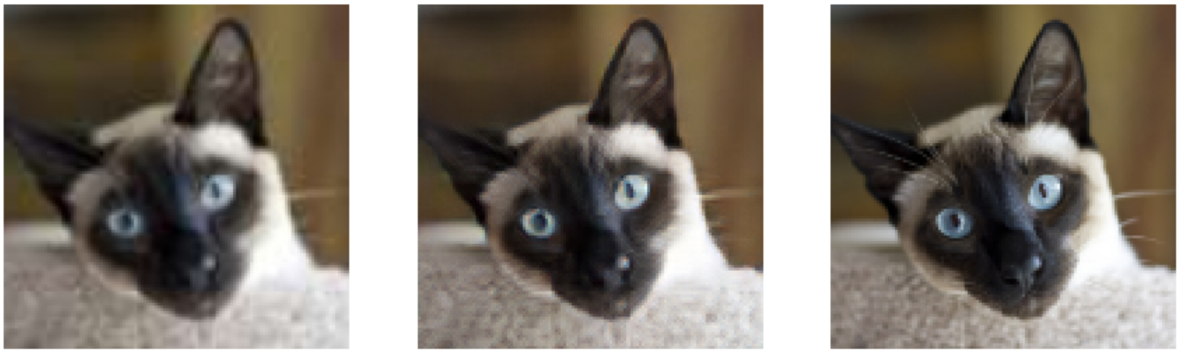
```
learn.unfreeze()
```

In [ ]:

```
do_fit('1b', slice(1e-5,lr))
```

Total time: 11:39

```
epoch  train_loss  valid_loss     pixel    feat_0    feat_1    feat_2    gram_0
    1    3.303951    3.179916  0.135630  0.216009  0.277359  0.189097  0.430012
    2    3.308164    3.174482  0.135740  0.215970  0.277178  0.188737  0.428630
    3    3.294504    3.169184  0.135216  0.215401  0.276744  0.188395  0.428544
    4    3.282376    3.160698  0.134830  0.215049  0.275767  0.187716  0.427314
    5    3.301212    3.168623  0.135134  0.215388  0.276196  0.188382  0.427277
    6    3.299340    3.159537  0.135039  0.214692  0.275285  0.187554  0.427840
    7    3.291041    3.159207  0.134602  0.214618  0.275053  0.187660  0.428083
    8    3.285271    3.147745  0.134923  0.214514  0.274702  0.187147  0.423032
    9    3.279353    3.138624  0.136035  0.213191  0.273899  0.186854  0.420070
   10    3.261495    3.124737  0.135016  0.213681  0.273402  0.185922  0.416460
```



Input / Prediction / Target

In [ ]:

```
data = get_data(12,size*2)
```

In [ ]:

```
learn.data = data
learn.freeze()
gc.collect()
```

Out[ ]:

```
0
```

In [ ]:

```
learn.load('1b');
```

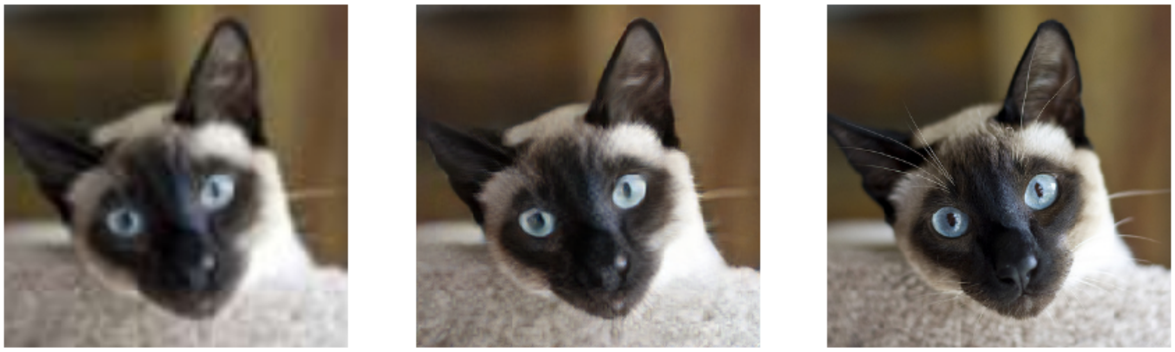
In [ ]:

```
do_fit('2a')
```

Total time: 43:44

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0
1	2.249253	2.214517	0.164514	0.260366	0.294164	0.155227	0.385168
2	2.205854	2.194439	0.165290	0.260485	0.293195	0.154746	0.374004
3	2.184805	2.165699	0.165945	0.260999	0.291515	0.153438	0.361207
4	2.145655	2.159977	0.167295	0.260605	0.290226	0.152415	0.359476
5	2.141847	2.134954	0.168590	0.260219	0.288206	0.151237	0.348900
6	2.145108	2.128984	0.164906	0.259023	0.286386	0.150245	0.352594
7	2.115003	2.125632	0.169696	0.259949	0.286435	0.150898	0.344849
8	2.109859	2.111335	0.166503	0.258512	0.283750	0.148191	0.347635
9	2.092685	2.097898	0.169842	0.259169	0.284757	0.148156	0.333462
10	2.061421	2.080940	0.167636	0.257998	0.282682	0.147471	0.330893

Input / Prediction / Target



In [ ]:

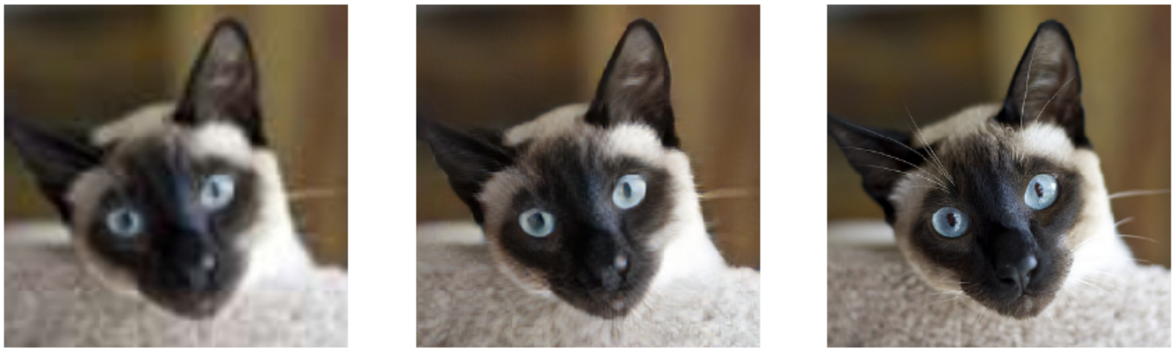
```
learn.unfreeze()
```

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

Total time: 45:19

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0
1	2.061799	2.078714	0.167578	0.257674	0.282523	0.147208	0.330824
2	2.063589	2.077507	0.167022	0.257501	0.282275	0.146879	0.331494
3	2.057191	2.074605	0.167656	0.257041	0.282204	0.146925	0.330117
4	2.050781	2.073395	0.166610	0.256625	0.281680	0.146585	0.331580
5	2.054705	2.068747	0.167527	0.257295	0.281612	0.146392	0.327932
6	2.052745	2.067573	0.167166	0.256741	0.281354	0.146101	0.328510
7	2.051863	2.067076	0.167222	0.257276	0.281607	0.146188	0.327575
8	2.046788	2.064326	0.167110	0.257002	0.281313	0.146055	0.326947
9	2.054460	2.065581	0.167222	0.257077	0.281246	0.146016	0.327586
10	2.052605	2.064459	0.166879	0.256835	0.281252	0.146135	0.327505

Input / Prediction / Target



Test

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

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

Out[ ]:
```
819.2
```

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

Out[ ]:
```
1280.0
```

In [ ]:
```
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
```

Out[ ]:

```
PosixPath('/data1/jhoward/git/course-v3/nbs/dl1/data/oxford-iiit-pet/small-256/
Siamese_178.jpg')
```

In [ ]:

```
img = open_image(fn); img.shape
```

Out[ ]:

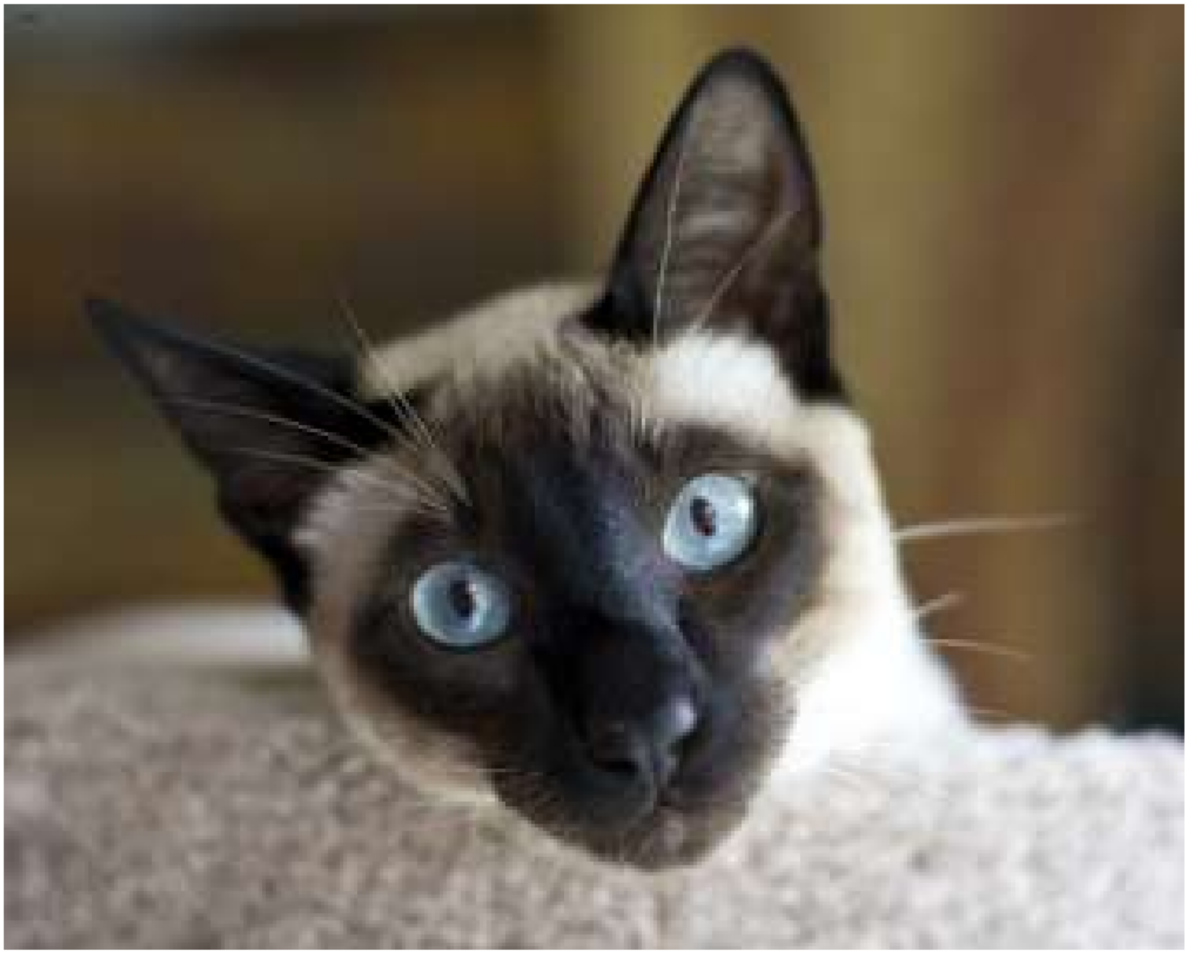
```
torch.Size([3, 256, 320])
```

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))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

In [ ]:

```

```