In [0]:

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

from torchvision.models import vgg16_bn
```

In [0]:

```
folder = 'celebrities'
file = 'celebrities.txt'
```

In [0]:

```
from fastai.vision import *
path = Path('gdrive/My Drive/')
dest = path/folder
dest.mkdir(parents=True, exist_ok=True)
```

In [6]:

```
path.ls()
```

Out[6]:

```
[PosixPath('gdrive/My Drive/celebrities')]
```

In [0]:

```
verify_images('gdrive/My Drive/celebrities/', delete=True, max_size=500)
```

In [0]:

```
path = Path('gdrive/My Drive/')
```

In [0]:

```
path_hr = path/'celebrities'
path_lr = path/'small-96'
path_mr = path/'small-256'
```

In [0]:

```
# path for original folder images
il = ImageList.from_folder(path_hr)
```

In [0]:

```
# resize images to jpeg quality and move them different folder
def resize_one(fn, i, path, size):
    dest = path/fn.relative_to(path_hr)
    dest.parent.mkdir(parents=True, exist_ok=True)
    img = PIL.Image.open(fn)
    # ressize to particular size
    targ_sz = resize_to(img, size, use_min=True)
    # save to JPEG quality which is 60
    img = img.resize(targ_sz, resample=PIL.Image.BILINEAR).convert('RGB')
    img.save(dest, quality=60)
```

In [0]:

```
# create smaller image sets the first time this nb is run
sets = [(path_lr, 96), (path_mr, 256)]
```

```python
for p,size in sets:
    if not p.exists():
        print(f"resizing to {size} into {p}")
        parallel(partial(resize_one, path=p, size=size), il.items)
```

In [0]:

```python
# batch size and Base_Model
bs,size=32,128
arch = models.resnet34
```

In [0]:

```python
# Creating Validation set
src = ImageImageList.from_folder(path_lr).split_by_rand_pct(0.1, seed=42)
```

In [0]:

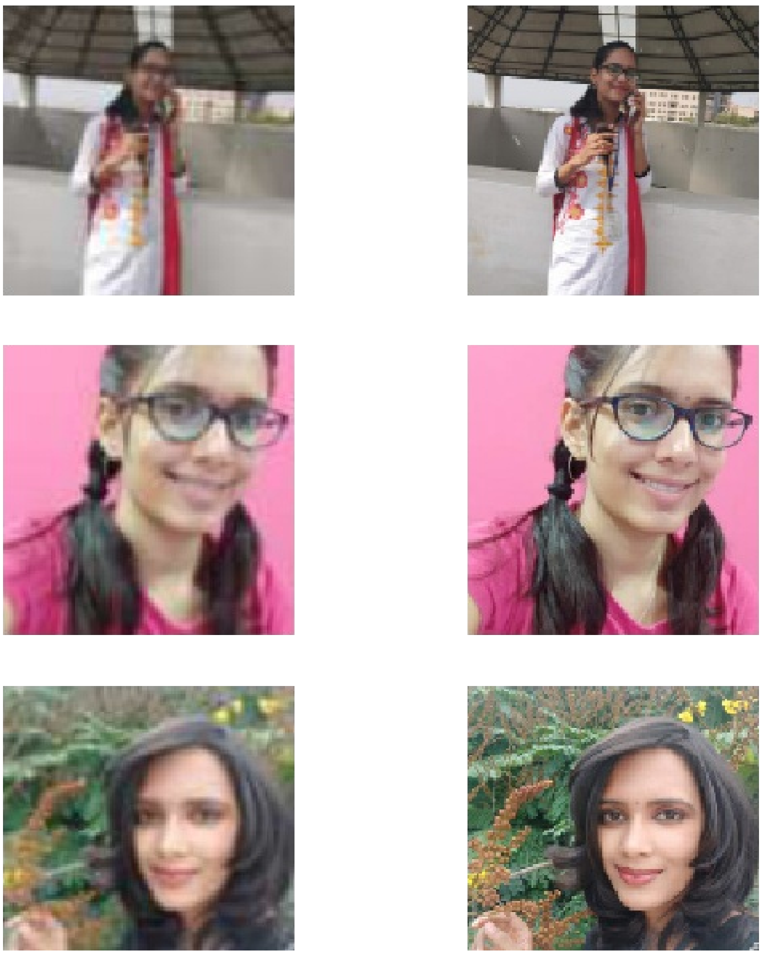
```python
# loading the data or images after trnasform from databunch object
def get_data(bs,size):
    data = (src.label_from_func(lambda x: path_hr/x.name)
           .transform(get_transforms(max_zoom=2.), size=size, tfm_y=True)
           .databunch(bs=bs).normalize(imagenet_stats, do_y=True))

    data.c = 3
    return data
```

In [0]:

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

In [17]:

```python
# getting both the images---Blur and HD from the both paths
data.show_batch(ds_type=DatasetType.Valid, rows=3, figsize=(9,9))
```




Feature LOsss

In [18]:

```
t = data.valid_ds[0][1].data
t.shape
```

Out[18]:

```
torch.Size([3, 128, 128])
```

In [19]:

```
# We use this for gram matrix
t = torch.stack([t,t])
t.shape
```

Out[19]:

```
torch.Size([2, 3, 128, 128])
```

Gram Matrix

In [0]:

```
# Gram matrix
def gram_matrix(x):
    n,c,h,w = x.size()
    # gram matrix at each layer is (c,c) shape
    x = x.view(n, c, -1)
    return (x @ x.transpose(1,2))/(c*h*w)
```

In [21]:

```
# we  usually take loss of the Gram Matrix
gram_matrix(t).shape
```

Out[21]:

```
torch.Size([2, 3, 3])
```

In [0]:

```
# Loss of the Gram Matrices of both the real and Fake Images
base_loss = F.l1_loss
```

In [23]:

```
# .features has convolutn model and no head
# eval mode because we do not train the weights
# requires_grad because we do not update the weights of the model
vgg_m = vgg16_bn(True).features.eval()
requires_grad(vgg_m, False)
```

```
Downloading: "https://download.pytorch.org/models/vgg16_bn-6c64b313.pth" to
/root/.cache/torch/checkpoints/vgg16_bn-6c64b313.pth
100%|██████████| 528M/528M [00:25<00:00, 21.7MB/s]
```

In [24]:

```
# we want to get all the maxpool layers of the model which do conatin the features at gram matrix
# Why max pool because thats where the grid size changes
blocks = [i-1 for i,o in enumerate(children(vgg_m)) if isinstance(o,nn.MaxPool2d)]
# layer no just before the maxPool
```

```
# these Layers are where we drag our features
blocks
```

Out[24]:

```
[5, 12, 22, 32, 42]
```

In [25]:

```
[vgg_m[i] for i in blocks]
```

Out[25]:

```
[ReLU(inplace=True),
 ReLU(inplace=True),
 ReLU(inplace=True),
 ReLU(inplace=True),
 ReLU(inplace=True)]
```

In [0]:

```
class FeatureLoss(nn.Module):
    def __init__(self, m_feat, layer_ids, layer_wgts):
        super().__init__()
        #m_feat is the model on which we want to generate feature losses on
        self.m_feat = m_feat
        # grab the layers for which u want to create feature losses
        self.loss_features = [self.m_feat[i] for i in layer_ids]
        # hook those outputs of those layers
        self.hooks = hook_outputs(self.loss_features, detach=False)
        # store their weights in layer_wgts
        self.wgts = layer_wgts
        self.metric_names = ['pixel',] + [f'feat_{i}' for i in range(len(layer_ids))
              ] + [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):
      # make features calls the target which ois the VGG model with feature losses or original ima
ge feature losses
        out_feat = self.make_features(target, clone=True)
        # input in output of the generator which is input to the target
        in_feat = self.make_features(input)
        # base_losses is pixel loss between input and target
        self.feat_losses = [base_loss(input,target)]
        # activations losses at layer's mentioned below
        self.feat_losses += [base_loss(f_in, f_out)*w
                             for f_in, f_out, w in zip(in_feat, out_feat, self.wgts)]
        # gram matrix losses of each of the leayer's
        self.feat_losses += [base_loss(gram_matrix(f_in), gram_matrix(f_out))*w**2 * 5e3
                             for f_in, f_out, w in zip(in_feat, out_feat, self.wgts)]

        # metricsa is used because prints out all the losses
        self.metrics = dict(zip(self.metric_names, self.feat_losses))
        # feat_losses contains sum of the losses
        # pixel losses + activations losses + gram Matrix losses
        return sum(self.feat_losses)

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

In [0]:

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

Train

In [28]:

```
wd = 1e-3
```

```
# unet trainer with VGG and callback is layer losses
learn = unet_learner(data, arch, wd=wd, loss_func=feat_loss, callback_fns=LossMetrics,
                     blur=True, norm_type=NormType.Weight)
gc.collect();
```

```
Downloading: "https://download.pytorch.org/models/resnet34-333f7ec4.pth" to
/root/.cache/torch/checkpoints/resnet34-333f7ec4.pth
100%|██████████| 83.3M/83.3M [00:03<00:00, 24.2MB/s]
```

In [0]:

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

In [0]:

```
# creating a function to train,save model and save Results
def do_fit(save_name, lrs=slice(lr), pct_start=0.9):
    learn.fit_one_cycle(35, lrs, pct_start=pct_start)
    learn.save(save_name)
    learn.show_results(rows=1, imgsize=5)
```

In [31]:

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

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0	gram_1	gram_2	time
0	6.137766	6.153541	0.981176	0.390436	0.472886	0.196494	1.790737	1.991948	0.329865	01:59
1	5.915826	5.695018	0.788763	0.376742	0.471296	0.195356	1.618298	1.917891	0.326673	02:00
2	5.669826	5.215996	0.616992	0.353730	0.459746	0.181175	1.460945	1.837314	0.306093	01:59
3	5.471190	4.729066	0.420828	0.333231	0.460066	0.172133	1.291438	1.751437	0.299934	02:00
4	5.277553	4.516310	0.324188	0.327545	0.450855	0.158211	1.262287	1.708761	0.284463	02:00
5	5.096451	4.112163	0.294933	0.304294	0.432304	0.146690	1.047286	1.610270	0.276387	01:59
6	4.921548	3.891963	0.276243	0.292251	0.415347	0.138044	0.964232	1.537335	0.268512	01:59
7	4.747077	3.693619	0.250582	0.282668	0.394491	0.130399	0.900290	1.474519	0.260670	01:59
8	4.587447	3.548858	0.232035	0.273244	0.378216	0.127815	0.846069	1.429219	0.262259	01:59
9	4.433180	3.434479	0.211707	0.265316	0.367478	0.123553	0.822800	1.389004	0.254621	01:59
10	4.296134	3.303581	0.187899	0.257663	0.355068	0.119779	0.791694	1.341029	0.250449	01:59
11	4.172534	3.262242	0.207798	0.252203	0.349011	0.120362	0.760227	1.320359	0.252282	01:59
12	4.069788	3.198142	0.195231	0.251682	0.344695	0.118947	0.744664	1.290251	0.252671	02:00
13	3.980131	3.168584	0.185377	0.248182	0.340010	0.117503	0.745941	1.281371	0.250200	02:00
14	3.897669	3.095296	0.209594	0.242145	0.332924	0.114102	0.709410	1.241727	0.245393	02:00
15	3.818690	3.065633	0.186547	0.239394	0.332150	0.113929	0.709489	1.240278	0.243845	02:00
16	3.751221	2.990372	0.198815	0.237464	0.325763	0.112443	0.678967	1.195189	0.241731	02:00
17	3.684710	3.009167	0.204471	0.239576	0.326260	0.112356	0.689678	1.196098	0.240727	01:59
18	3.623567	2.999739	0.203540	0.239439	0.323285	0.113233	0.696491	1.180167	0.243584	02:00
19	3.568455	2.957442	0.197422	0.234364	0.321128	0.115233	0.668234	1.171534	0.249526	02:00
20	3.515370	3.057944	0.205935	0.242677	0.331360	0.116780	0.690775	1.220010	0.250407	02:00
21	3.475170	3.317312	0.367540	0.245468	0.340833	0.119512	0.727884	1.259519	0.256554	02:00
22	3.447414	3.206914	0.308184	0.247668	0.345040	0.120994	0.672463	1.261006	0.251559	02:00
23	3.421541	3.107179	0.247052	0.240310	0.327965	0.116086	0.704275	1.220956	0.250534	02:00
24	3.393019	2.940509	0.235666	0.232621	0.316226	0.112569	0.651703	1.146703	0.245021	02:00
25	3.358772	2.893595	0.179152	0.235152	0.319987	0.113516	0.638329	1.159158	0.248301	02:00
26	3.327120	3.006944	0.216984	0.236900	0.325603	0.117107	0.676926	1.181746	0.251678	02:00

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0	gram_1	gram_2	time
27	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	02:00
28	3.283154	3.078723	0.288245	0.237965	0.326772	0.114919	0.666258	1.196666	0.247898	02:00
29	3.261698	3.076674	0.222363	0.245453	0.335565	0.117453	0.692516	1.214335	0.248988	02:00
30	3.241544	3.054961	0.281111	0.238641	0.329889	0.117470	0.644058	1.193085	0.250708	02:00
31	3.223059	2.934269	0.187651	0.238045	0.322236	0.113358	0.663641	1.165812	0.243526	02:00
32	3.204104	2.889692	0.192613	0.234081	0.321574	0.113931	0.628937	1.151724	0.246832	02:00
33	3.184755	2.826497	0.192390	0.230738	0.315301	0.114342	0.602354	1.123877	0.247494	01:59
34	3.161766	2.810848	0.186717	0.230421	0.313704	0.113728	0.602368	1.117777	0.246134	01:59

Input / Prediction / Target

TEST

In [0]:

```
# using the same unet learner
learn = unet_learner(data, arch, loss_func=F.l1_loss, blur=True, norm_type=NormType.Weight)
```

In [0]:

```
# creating a data bunch from the earlier trained 256 size images
data_mr = (ImageImageList.from_folder(path_mr).split_by_rand_pct(0.1, seed=42)
          .label_from_func(lambda x: path_hr/x.name)
          .transform(get_transforms(), size=size, tfm_y=True)
          .databunch(bs=1).normalize(imagenet_stats, do_y=True))
data_mr.c = 3
```

In [0]:

```
# loading the previous model
learn.load('1a');
```

In [0]:

```
# data of  256 size
learn.data = data_mr
```

In [36]:

```
# taking the validation data from the data of size 256
fn = data_mr.valid_ds.x.items[0]; fn
```

Out[36]:

```
PosixPath('gdrive/My Drive/small-256/IMG_20190218_112923.jpg')
```

In [37]:

In [37]:

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

Out[37]:

```
torch.Size([3, 341, 256])
```

In [0]:

```
# predicting the 256 size image using earlier trained model
p,img_hr,b = learn.predict(img)
```

In [39]:

```
# Original Image
show_image(img, figsize=(18,15), interpolation='nearest');
```

In [40]:

```
# Predicted Image
Image(img_hr).show(figsize=(18,15))
```

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

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