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 [4]:

```
path.ls()
```

Out[4]:

```
[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)]
```

```
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]:

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

In [0]:

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

In [0]:

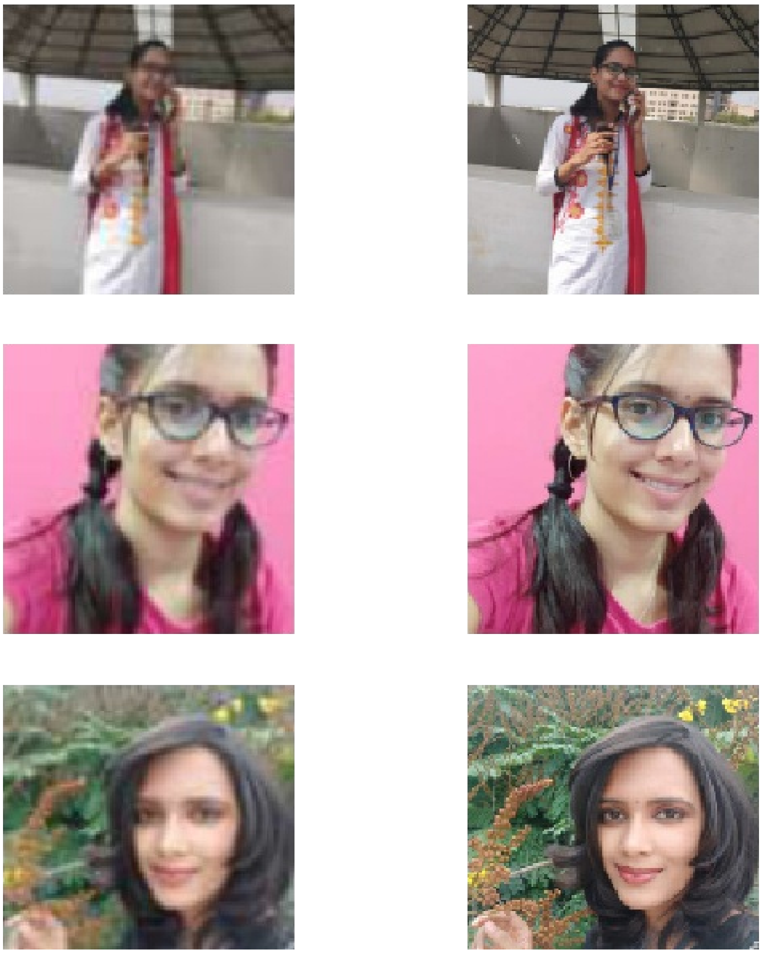
```
# loading the data or images after trnasform from databunch object
def get_data(bs,size):
    data = (src.label_from_func(lambda x: path_hr/x.name)
           .transform(get_transforms(max_zoom=2.), size=size, tfm_y=True)
           .databunch(bs=bs).normalize(imagenet_stats, do_y=True))

    data.c = 3
    return data
```

In [0]:

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

In [15]:

```
# 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 [16]:

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

Out[16]:

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

In [17]:

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

Out[17]:

```
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 [19]:

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

Out[19]:

```
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 [21]:

```
# .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 [00:21<00:00, 25.7MB/s]
```

In [22]:

```
# 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[22]:

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

In [23]:

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

Out[23]:

```
[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 [26]:

```
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 [00:00<00:00, 112MB/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(25, 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	5.641373	5.498374	0.844165	0.370181	0.474101	0.184040	1.477963	1.838874	0.309051	01:34
1	5.338466	5.063099	0.593446	0.350815	0.462964	0.172919	1.385129	1.801466	0.296360	01:36
2	5.116603	4.522452	0.390784	0.332236	0.460178	0.160182	1.186272	1.705067	0.287733	01:36
3	4.960727	4.326937	0.417425	0.316914	0.449193	0.155959	1.068051	1.632384	0.287011	01:37
4	4.800539	4.148689	0.390912	0.306781	0.434017	0.144265	1.020429	1.582016	0.270269	01:37
5	4.655005	4.034017	0.294854	0.303316	0.428823	0.142439	1.010527	1.581414	0.272644	01:36
6	4.524212	3.790830	0.266744	0.289757	0.410256	0.134925	0.911761	1.512163	0.265226	01:36
7	4.384341	3.624384	0.265941	0.278713	0.388793	0.126882	0.867215	1.441482	0.255358	01:37
8	4.253201	3.491101	0.241710	0.271199	0.375089	0.123964	0.831575	1.394567	0.252996	01:37
9	4.141680	3.411016	0.232800	0.263722	0.365691	0.121813	0.811932	1.363183	0.251876	01:37
10	4.038402	3.270058	0.209880	0.256985	0.353392	0.119658	0.764045	1.313477	0.252621	01:37
11	3.949620	3.165552	0.206386	0.247506	0.340388	0.116415	0.744136	1.261048	0.249673	01:37
12	3.857996	3.258403	0.262151	0.250139	0.346240	0.120877	0.760779	1.264687	0.253531	01:37
13	3.792228	3.158397	0.225925	0.247237	0.339255	0.118736	0.718965	1.256199	0.252080	01:37
14	3.727139	3.239476	0.288163	0.248202	0.341612	0.120019	0.707246	1.281096	0.253139	01:37
15	3.680533	3.162971	0.233770	0.244719	0.337235	0.118874	0.712075	1.261025	0.255272	01:37
16	3.619112	3.090356	0.248464	0.241922	0.331612	0.116482	0.690356	1.213088	0.248432	01:37
17	3.569936	3.271095	0.274900	0.248922	0.348787	0.122567	0.742620	1.279405	0.253893	01:37
18	3.532045	3.136049	0.244310	0.248227	0.337140	0.119522	0.714045	1.220962	0.251844	01:37
19	3.488273	3.031827	0.212454	0.239764	0.328352	0.116342	0.689861	1.194702	0.250351	01:37
20	3.455469	3.067832	0.242926	0.238743	0.329088	0.118486	0.669269	1.216039	0.253282	01:37
21	3.425224	3.013276	0.199147	0.240380	0.326446	0.117328	0.668463	1.205998	0.255514	01:37
22	3.393905	3.221187	0.301874	0.246124	0.341176	0.122997	0.692956	1.256531	0.259528	01:37
23	3.371483	3.121779	0.265750	0.241931	0.330437	0.119115	0.680845	1.227818	0.255884	01:37
24	3.345480	3.034579	0.231662	0.239369	0.325350	0.116629	0.661361	1.207877	0.252331	01:37

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 [43]:

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

Out[43]:

```
PosixPath('gdrive/My Drive/small-256/vlcsnap-4255-10-19-23h48m36s337.png')
```

In [44]:

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

Out[44]:

```
torch.Size([3, 256, 609])
```

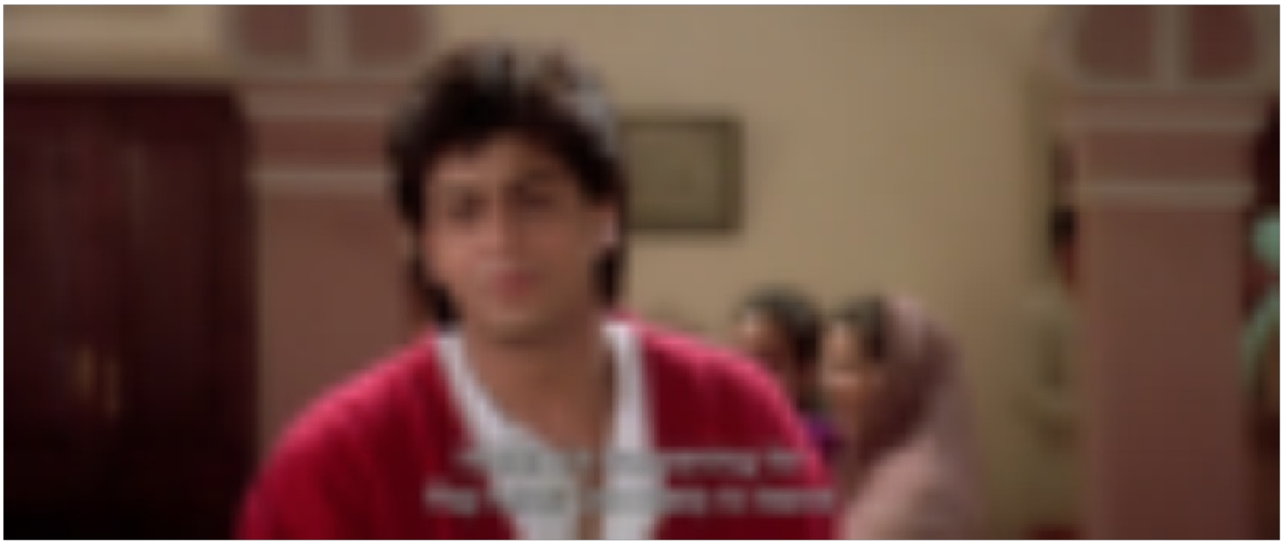
In [0]:

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

In [46]:

```
# Original Image
```

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



In [47]:

```
# 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
integers).
```