Image super-resolution exercise

The purpose of this assignment is to get familiar with construction and training of fully convolutional networks. we will specifically use the task of image super-resolution, and we’ll construct several different architectures and compare the results achieved by each of them.

**Stage 1:**

Create dataset – we would like to create a dataset and a data-loader for the training of super-resolution network  
We will use self-supervision to create our dataset

Our basic dataset for this task will be the PascalVOC 2007 dataset which is available [here](http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar)

Your first task was to create a dataset with images of 3 different sizes:  
X - 72x72x3 y\_mid – 144x144x3 y\_large – 288x288x3

You may choose to either persist these different sizes to disk or create them on-the-fly within your data loader (data-generator)

Once you have loaded and generated the above arrays of input images, we would like to split them into training and validating our model, for simplicity, we will use the first 1000 images (~20%) for validation and the rest for training. (note you should have 5011 images in total but are welcome to add more if you wish to)

\* note – I strongly recommend to only work on a sample of the data to make the process of loading and processing faster. This is a good practice for quick development of the loading pipeline and initial model creation. Once everything is working the way we expect it to work, we can increase the number of images we load and be sure that the process runs smoothly.

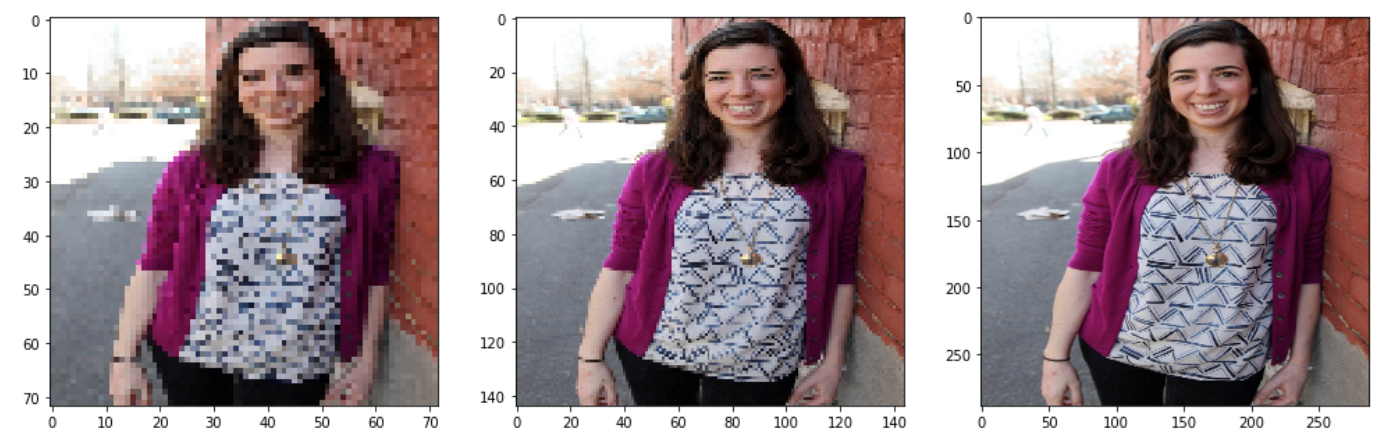
Next, present few images so that we can compare the training with our desired labels, this process is good for verifying that we have got the input we want and will also be useful for visual assessment of our model’s results so make sure you write it as a function for later reuse.

You should get something like this as an output:

X\_train[0]

y\_mid\_train[0]

y\_large\_train[0]



Step 2:

Create an initial model

In this step, create an initial fully convolutional model with the following architecture:

output

Upsampling2D

Conv2d

3 filters

Kernel size 1

Conv2d

64 filters

Kernel size 3

Conv2d

64 filters

Kernel size 3

input

then fit your model to our data using X\_train and y\_mid\_train only

Step 3:

add another block to your model so that you’ll have both 144x144x3 output along with 288x288x3 output as follows:

(make sure you understand the dimensions in each step of the process)

input

Conv2d

64 filters

Kernel size 3

Conv2d

64 filters

Kernel size 3

Upsampling2D

Conv2d

3 filters

Kernel size 1

Output (mid)

Upsampling2D

Conv2d

3 filters

Kernel size 1

Output (large)

Step 4:

Add residual blocks into the process

spend a moment to think, what should be the input and output of the residual-block model

Output (model)

**Conv2d**

**32 filters**

**Kernel size 3**

**Conv2d**

**32 filters**

**Kernel size 3**

**Input with shape (H,W,32)**

input

Residual block

Residual block

Upsampling2D

Conv2d

3 filters

Kernel size 1

Output (mid)

Upsampling2D

Residual block

Conv2d

3 filters

Kernel size 1

Output (large)

Step 5:

replace the residual blocks we defined above with a dilated (Atrous) convolutional block as described below:

**Input with shape (H,W,32)**

**Conv2d**

**32 filters**

**Kernel size 3**

**Conv2d dilation 4**

**32 filters**

**Kernel size 3**

**Conv2d dilation 1**

**32 filters**

**Kernel size 3**

**Conv2d dilation 2**

**32 filters**

**Kernel size 3**

concatenate

Output (model)

input

Dilated conv. block

Dilated conv. block

Upsampling2D

Conv2d

3 filters

Kernel size 1

Output (mid)

Upsampling2D

Dilated conv. block

Conv2d

3 filters

Kernel size 1

Output (large)

Step 6:

Add pretrained network (either efficientnet / resnet or any other) feature extractor to the network (note that the input to the network is only being read once)

Upsampling2D

Conv2d

3 filters

Kernel size 1

**Concatenate on the channels dimension**

Conv2d

64 filters

Kernel size 3

Conv2d

64 filters

Kernel size 3

Output (mid)

input

Upsampling2D

**Feature extractor from VGG16 block1conv2 layer**

Conv2d

3 filters

Kernel size 1

Output (large)

(optional) Step 7: replace the Upsampling2D layer with [tf.nn.depth\_to\_space](https://www.tensorflow.org/api_docs/python/tf/nn/depth_to_space) with parameters (x,scale)

\* Note that the number of filters in the preceding layer (num of input channels to the [tf.nn.depth\_to\_space](https://www.tensorflow.org/api_docs/python/tf/nn/depth_to_space)) should be divisible by (scale^2)

depth\_to\_space

Conv2d

3 filters

Kernel size 1

depth\_to\_space

Conv2d

3 filters

Kernel size 1

Output (mid)

input

Conv2d

64 filters

Kernel size 3

Conv2d

64 filters

Kernel size 3

**Concatenate on the channels dimension**

**Feature extractor from VGG16 block1conv2 layer**

Output (large)

8. add a custom callback that creates plots of one of the validation images over the epochs – to visually exemplify the improvements achieved during the convergence process. You may (but not have to) create a gif that shows the aggregation of these plots as an animation

Summary:

Take one or more of the networks that you have trained during the exercise (or another network you can think of) and train it to get the best result you can both for the optimization metric and in terms of visual results

Create a report that describes your work and compares the different results you got from the various models that you have trained.

Use both visual examples of results you got in each stage and a relevant metrics table

Refer to PSNR (peak signal to noise ratio) metric in addition to any other metric you consider relevant

You may want to consider using a custom PSNR loss as well