Udacity Machine Learning Engineer Nanodegree

Capstone Project Report

Svetoslav Paregov
June 17, 2019

Single Image Super Resolution through a CNN

I. Definition

Project Overview

Single image super-resolution is the task of inferring a high-resolution image from a single low-resolution input. Image resolution enhancement is something that we used to see in many movies, but it is not something trivial to achieve. There are a lot of algorithms used in different software products that increase the image resolution with different degree of success.
Machine Learning has been used in many different image processing tasks with Convolutional Neural Networks. Detection of different objects, face recognition and other. In the last few years it is also used to improve the image resolution.

What it can be used for – increasing the size of pictures from old phones for printing or some other purpose; Increase the quality of old security footages; Use it as base for other Machine Learning tasks to provide better quality images;

There are a lot of researches on the topic from different universities. Inspiration for my project came from a research from Max Planck Institute for Intelligent Systems, Germany [1].

Problem Statement

For the moment there are no algorithms or models that increase the quality of low-resolution images that restore the details. I'm not saying I will produce such a model, but this is a goal I would like to achieve in a long run. This is something that I'm interested in a very long time so, finding the research papers was very intriguing for me.

Traditionally, the performance of algorithms for this task is measured using pixel-wise reconstruction measures such as peak signal-to-noise ratio (PSNR) which have been shown to correlate poorly with the human perception of image quality. As a result,

algorithms minimizing these metrics tend to produce over-smoothed images that lack high-frequency textures and do not look natural despite yielding high PSNR values. In other words, for the computer looks good but not for the humans. A model or algorithm that is improving for human perception is needed.

I propose a CNN model that will be optimized for sharper and close as possible to the reality images. I hope to achieve this with correctly implemented loss functions. Still this is a hard problem since during the scale down a lot of information is lost and there so many variations when scaling up. So, part of the missing information should be synthesized by the CNN.

Metrics

Evaluation can be done visually or by feeding the enhanced images to an object recognition model. I'm going to rely mostly on visual evaluation showing the output to multiple people. If I have time will train an object recognition model for automated evaluation.

II. Analysis

Data Exploration

I have two datasets – MS COCO (Common Objects in Context) and CelebA, which will be used separately to train the model. Both datasets are very popular and used in a lot of examples and research papers.

First will start with the MS COCO. It can be downloaded from - http://cocodataset.org/ . Here is what they write about their dataset:

"We present a new dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context."

This dataset contains all kind of images – animals, people, cars, buildings and many more. I think this will be helpful so the model is more general when enhancing random image.

The second dataset is only people – mostly faces, but there are also full body pictures. It can be downloaded from - http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. Here is how they describe the dataset:

"CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter."

In this case I want the model to be less generic. I hope this will give a better result when enhancing pictures of humans.

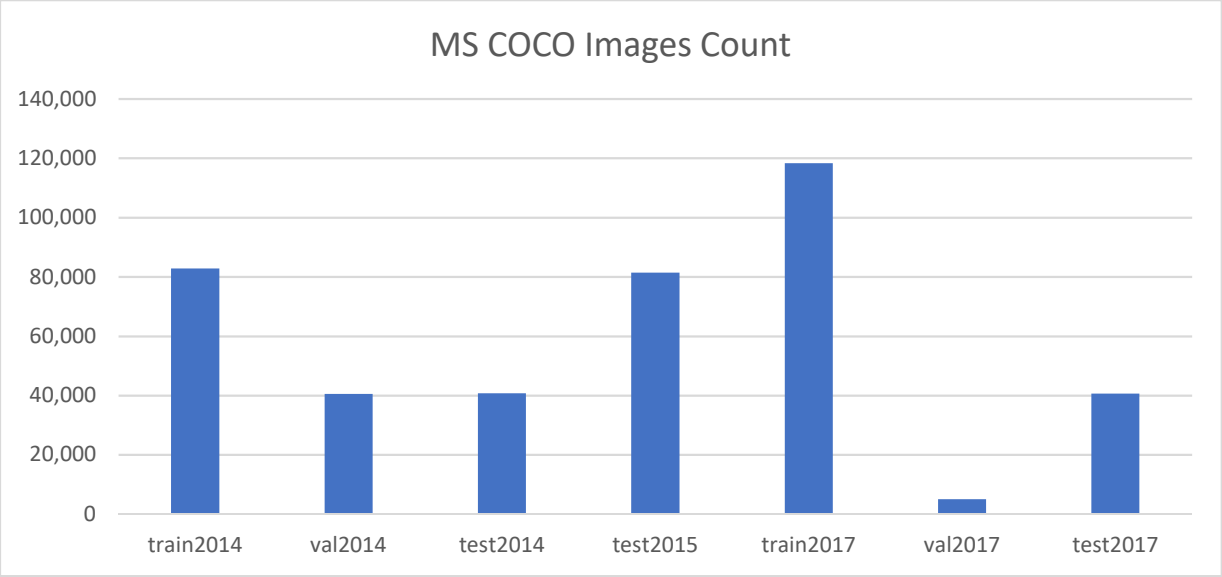
The low-resolution image that I'm going to feed into the model input is 32x32 pixels and the output image will be 128x128 pixels. Selected resolution increase ratio is 4 times.

Some of the images from MS COCO.

Some of the images from CelebA

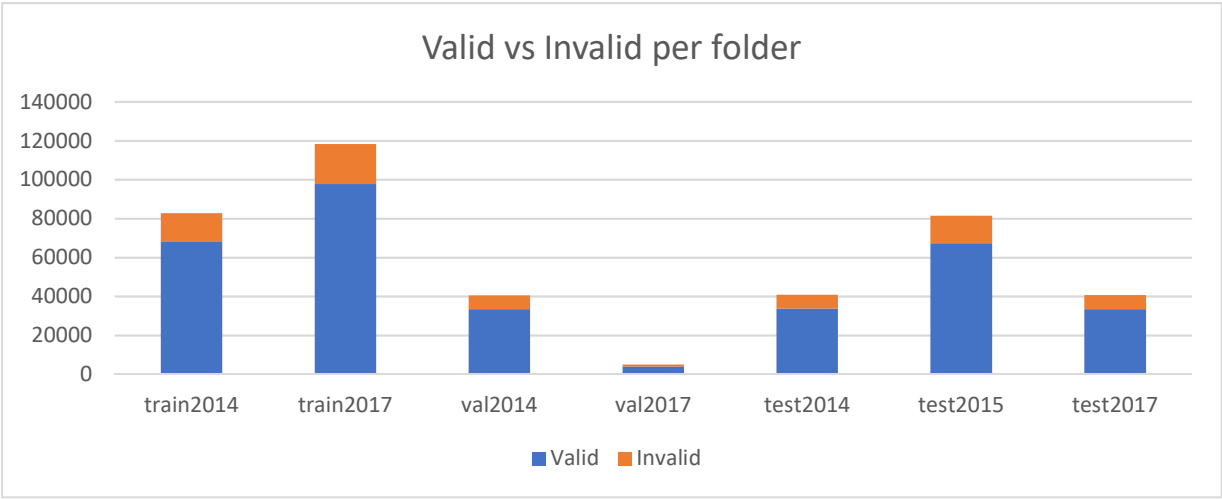
Exploratory Visualization

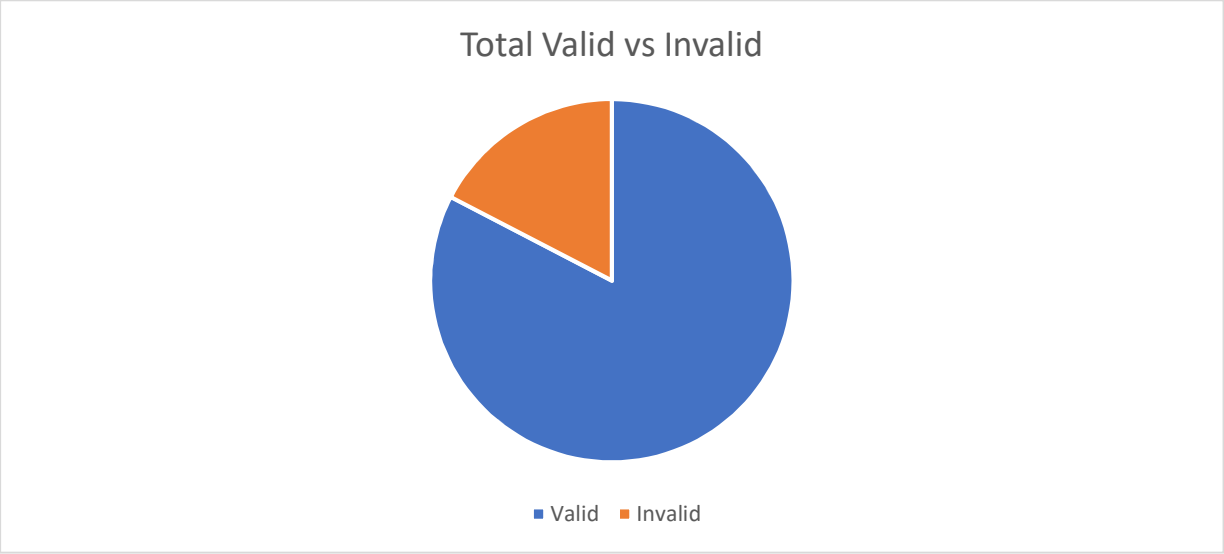
Total images count for MS COCO is 409,453 which are separated in multiple folders by the provider.



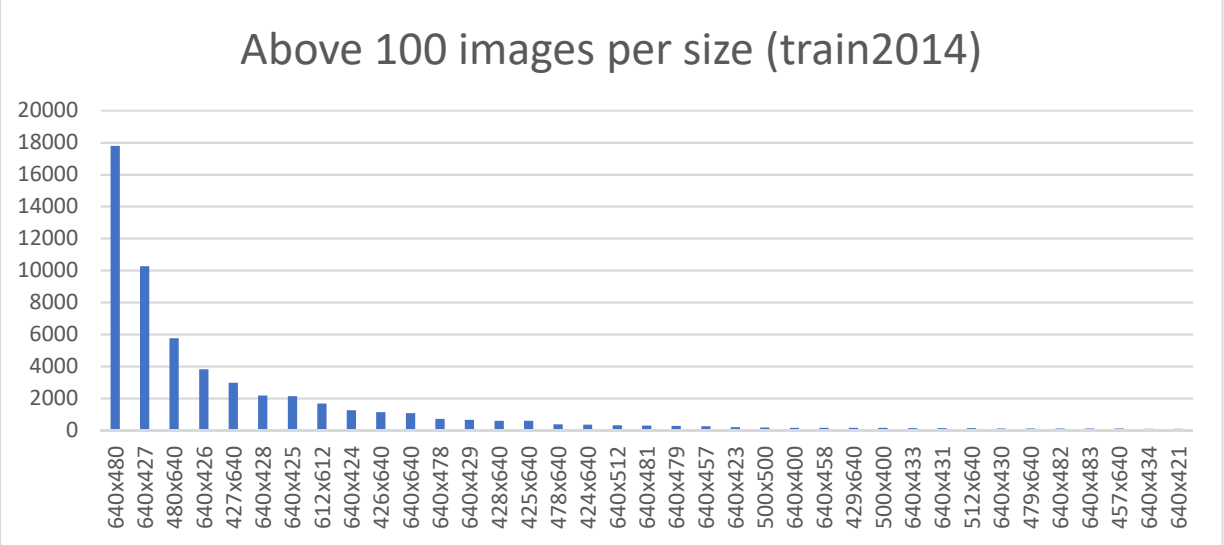
MC COCO Images Count as provided per folder

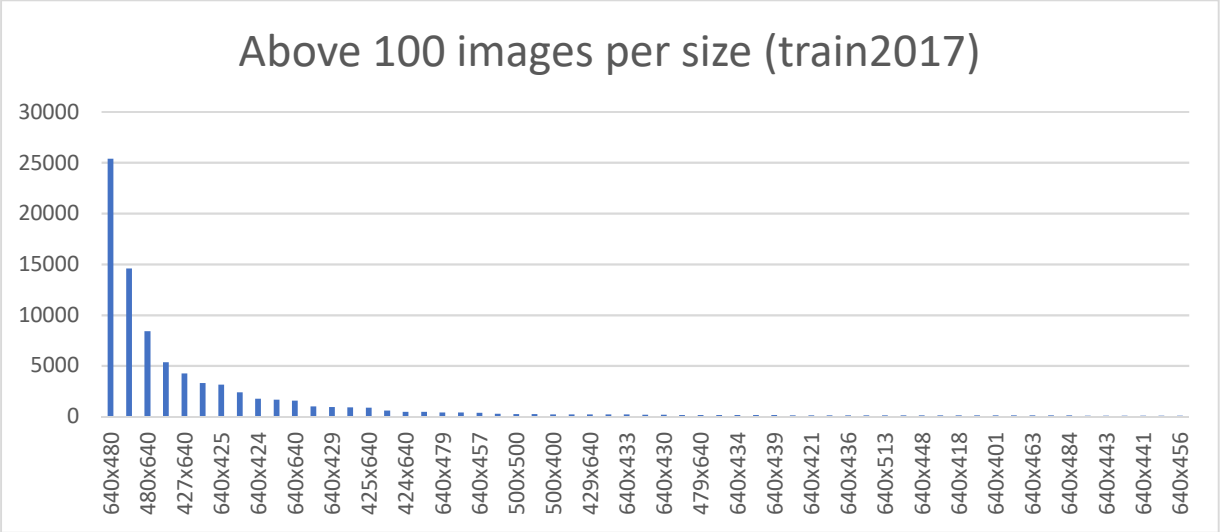
Here are some statistics per folder. I am going to show the count of images per size. Since there are so many sizes I'm showing only those sizes that have more than 100 images. For the training purpose I have decided to use only images that have width and height more than 384 pixels. I call them "valid" and those that one of the dimensions is smaller than 384 pixels are called "invalid".

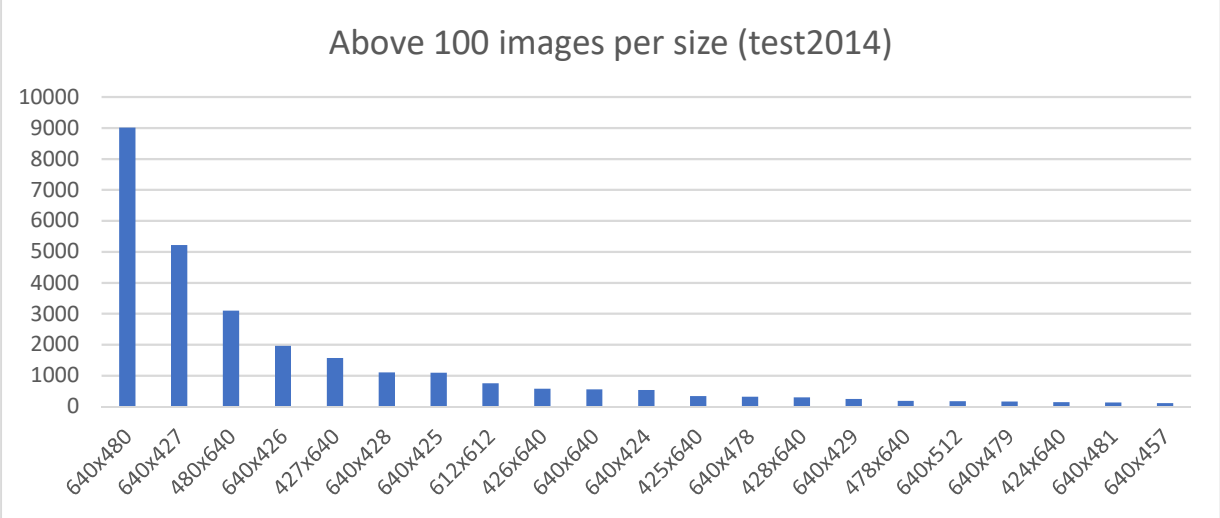


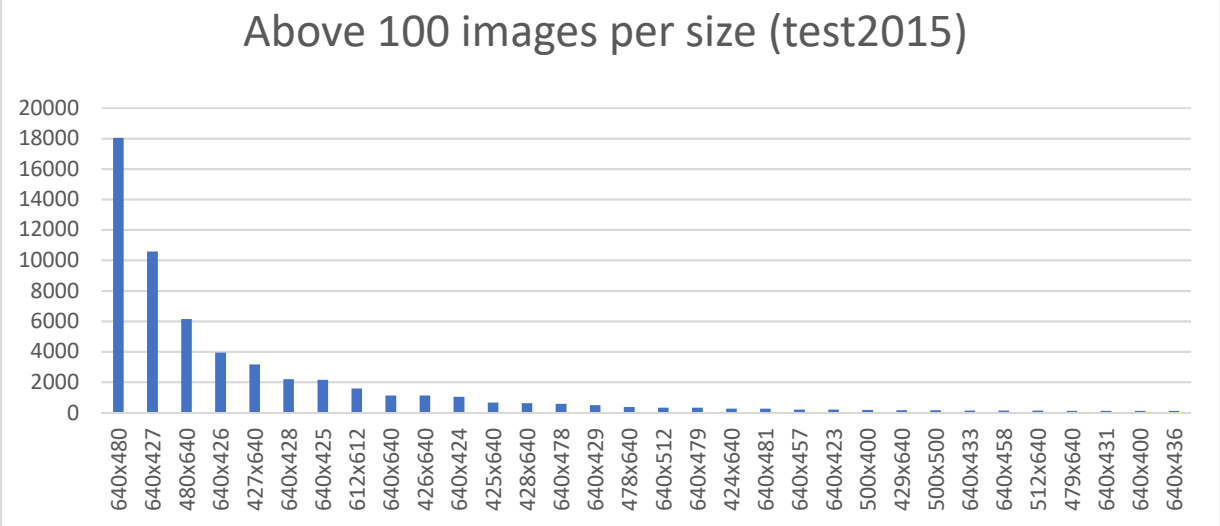


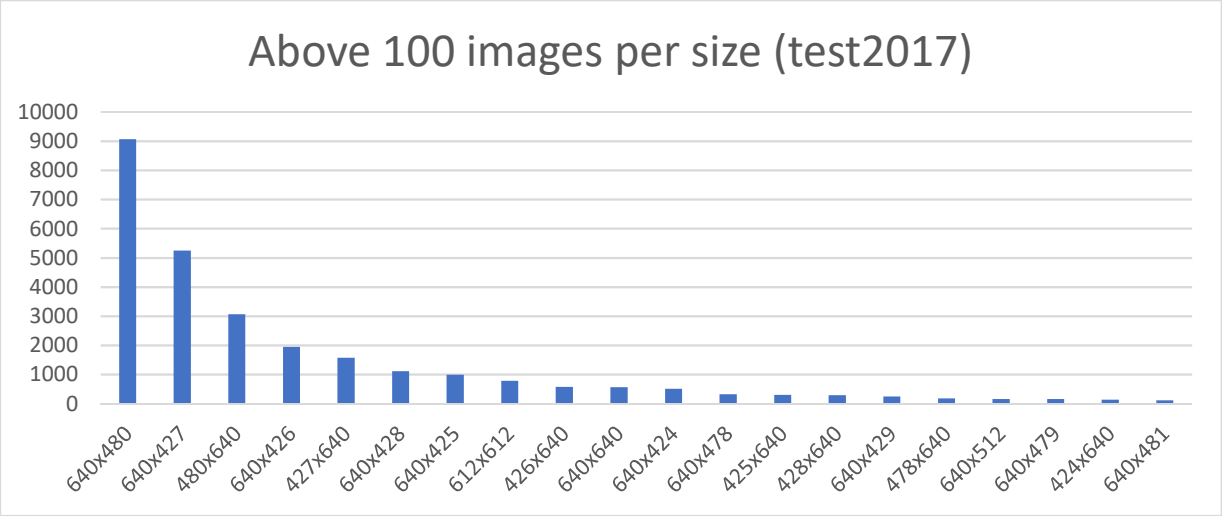
We can see that per folder and in total there are much valid (338,318) than invalid (71,135).

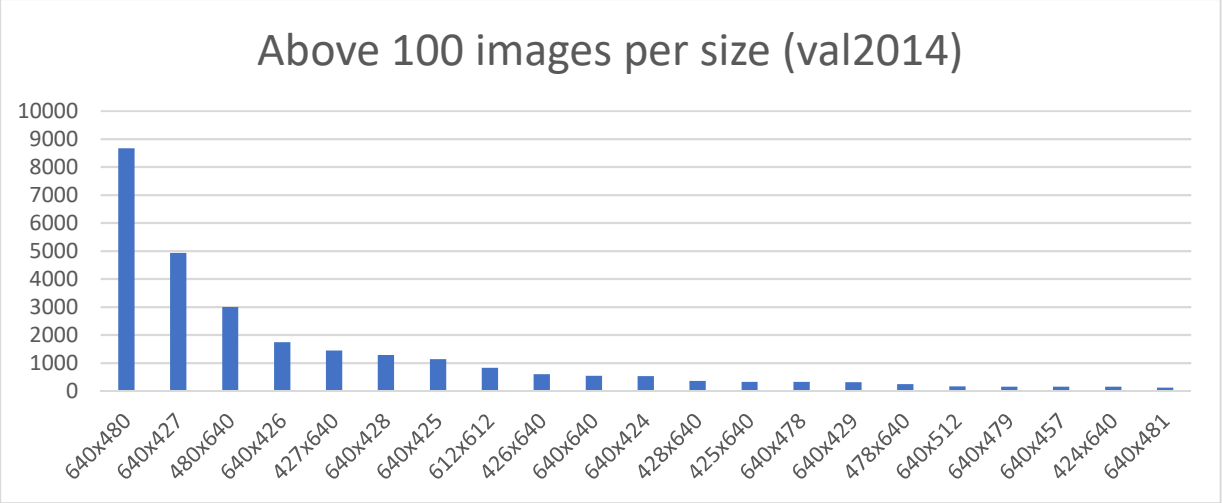


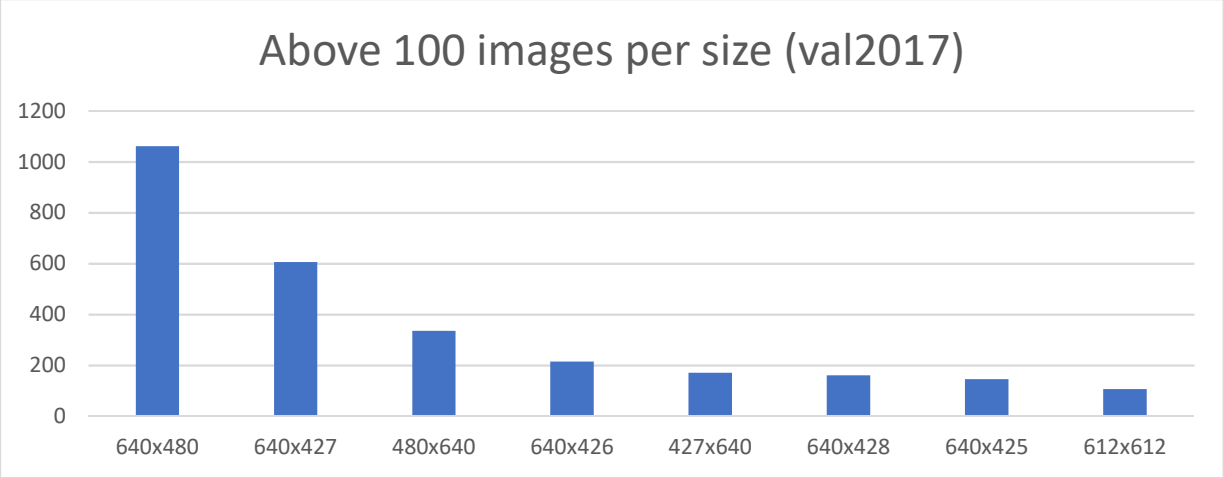

Above 100 images per size (train2017)
30000
25000
20000
15000
10000
5000
0
640x480
480x640
427x640
640x425
640x424
640x640
640x429
425x640
424x640
640x479
640x457
500x500
500x400
429x640
640x433
640x430
479x640
640x434
640x439
640x421
640x436
640x513
640x448
640x418
640x401
640x463
640x484
640x443
640x441
640x456


Above 100 images per size (test2014)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
640x480
640x427
480x640
640x426
427x640
640x428
640x425
612x612
426x640
640x640
640x424
425x640
640x478
428x640
640x429
478x640
640x512
640x479
424x640
640x481
640x457


Above 100 images per size (test2015)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
640x480
640x427
480x640
640x426
427x640
640x428
640x425
612x612
640x640
426x640
640x424
425x640
428x640
640x478
640x429
478x640
640x512
640x479
424x640
640x481
640x457
640x423
500x400
429x640
500x500
640x433
640x458
512x640
479x640
640x431
640x400
640x436

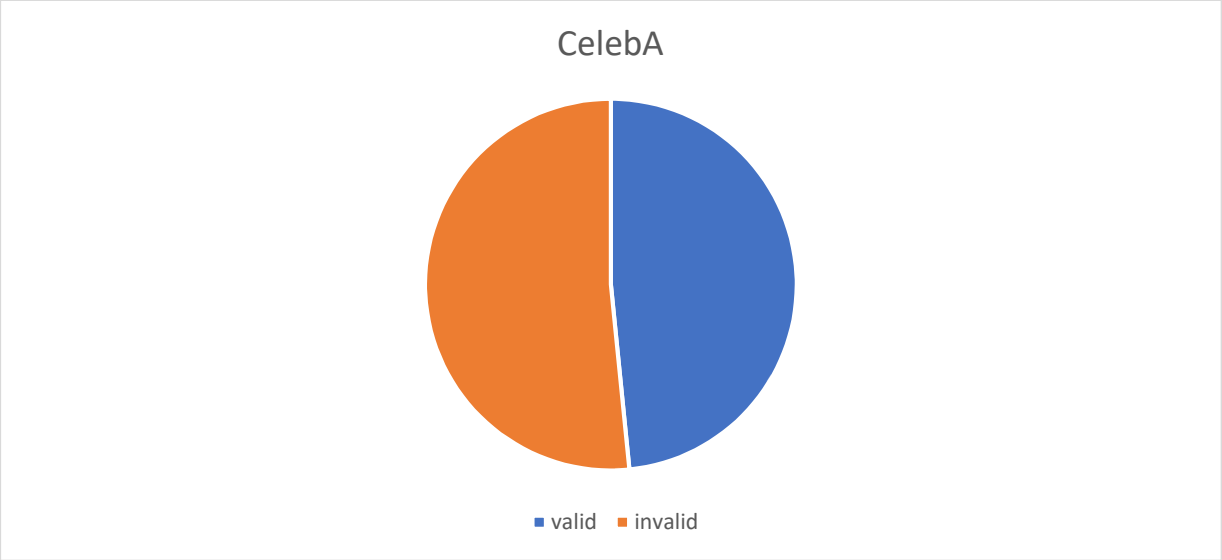




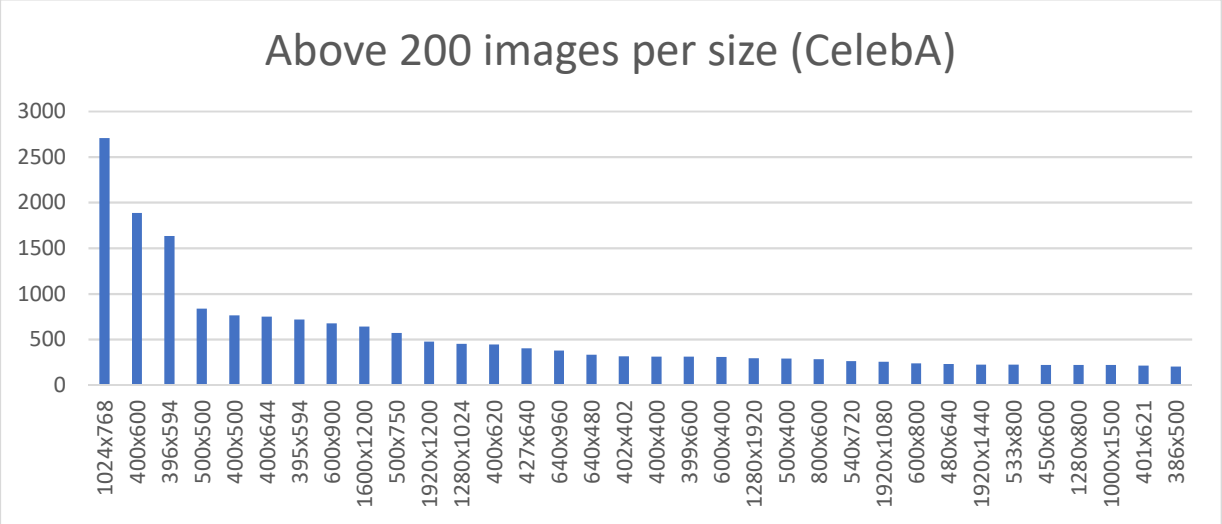


We can see that in all of the sets the majority of the images are 640x480 pixels.

CelebA contains 202,599 images of celebrities. Mostly face and portrait images, but there are also a full body images. All of the images are provided in one folder.



Here I'm showing the image sizes that has more than 200 images, since there were too many over 100.



In conclusion. I have 338,318 images from MS COCO and 98,102 images from CelebA which I can use for training the model.

Algorithms and Techniques

I propose to use a Convolutional Neural Network (CNN) that learns how to synthesize the missing pixels. I'm going to use standard CNN training with calculation of the accuracy based on a validation set. The other approach I want to explorer is Generative Adversarial Network (GAN) to train my model. Training on different datasets with different loss functions will be performed and tested. I plan to train the model using different loss functions and compare the results.

Benchmark

Automated benchmark that can be performed in such a task is passing the produced images to another model that classifies objects. The original images should be passes to the same classifier and calculate different results. The same calculation should be done for images enhanced with other methods and the results compared with my model.

Unfortunately, I don't have enough time to do all of this for this project and the goal is to achieve realistic images for human perception so I'm going to compare visually the results. The compare will be done with a result from software product that increase the image resolution and the real high-resolution image. I'm going to show the results to other people and will gather the statistics.

Another benchmark I'm going to use is a PSNR. PSNR is used to calculate the ratio between the maximum possible signal power and the power of the distorting noise which affects the quality of its representation. This ratio between two images is computed in decibel form. The PSNR is usually calculated as the logarithm term of decibel scale because of the signals having a very wide dynamic range. This dynamic range varies between the largest and the smallest possible values which are changeable by their quality.

III. Methodology

Data Preprocessing

Images are filtered so they have at least 384 pixels on the short side. Then the images are being cropped to a rectangular image using the short side for base size. Cropped images are then resized to 256x256 pixels, so they don't lose too much details. Selected size for low-resolution image is 32x32 pixels. Since our selected increase ratio is 4, that means the high-resolution image is 128x128 pixels.

Filtered images are being merged for the MS COCO dataset and then split to train, validation and test. Since the CelebA images are already together, I just filter them and split to train, validation and test.

I have selected to use 75% for train, 20% for validation and 5% for test. 5% for test is enough since the test will be a manual process. If needed I can change the percentage. Validation is not needed for GAN training, but I use validation for some tests.

Used Jupyter notebook – *./pre_processing/image_filter_and_split.ipynb*

Since I'm not going to upload the original dataset, here is a link with the split:

https://drive.google.com/open?id=1h11IyvatipkNJ-JwPzCfECOh8f-i8TT5

I have created also a smaller subset for train, validation and test sets for faster evaluation of the models.

Implementation

For start I have decided to go with regular training and validation to evaluate the performance of the model. I created few models which I have trained with different loss functions. All of the models contain only convolution layers and up sampling layers. No fully connected or pulling layers are used. This allow that the model to be used after that with images of any size.

All convolutional layers use '*uniform*' initializer and the kernel size I selected is 3.

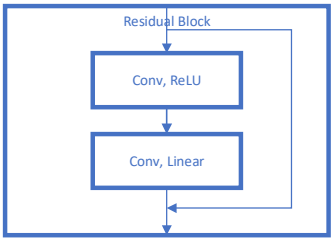
Used loss functions are in loss_functions.py

One of the loss functions is perceptual similarity measure. Rather than computing distances in image space, both I_{est} and I_{HR} are first mapped into a feature space by a differentiable function before computing their distance. This allows the model to generate outputs that may not match the ground truth image with pixel-wise accuracy but instead encourages the network to produce images that have similar feature representations.

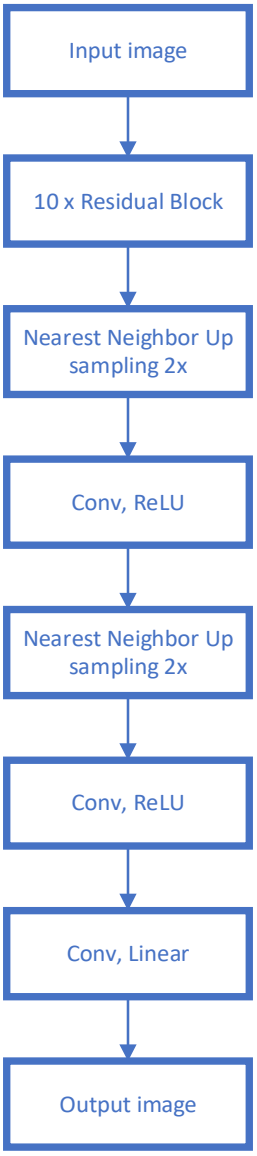
For the feature map, I use a pre-trained implementation of the popular VGG-19 network. It consists of stacked convolutions coupled with pooling layers to gradually decrease the spatial dimension of the image and to extract higher-level features in higher layers. To capture both low-level and high-level features, we use a combination of the second and fifth pooling layers and compute the MSE on their feature activations. Other loss functions I have implemented use the third pooling layer respectively from VGG-19 and VGG-16.

For style transfer loss I use a texture patch-wise during training to enforce locally similar textures between I_{est} and I_{HR}. The network therefore learns to produce images that have the same local textures as the high-resolution images during training. While the task of generating arbitrary textures is more demanding than single-texture synthesis, the LR image and high-level contextual cues give our network more information to work with, enabling it to generate varying high-resolution textures. Empirically, we found a patch size of 16×16 pixels to result in the best balance between faithful texture generation and the overall perceptual quality of the images.

The base model is with 10 residual blocks.

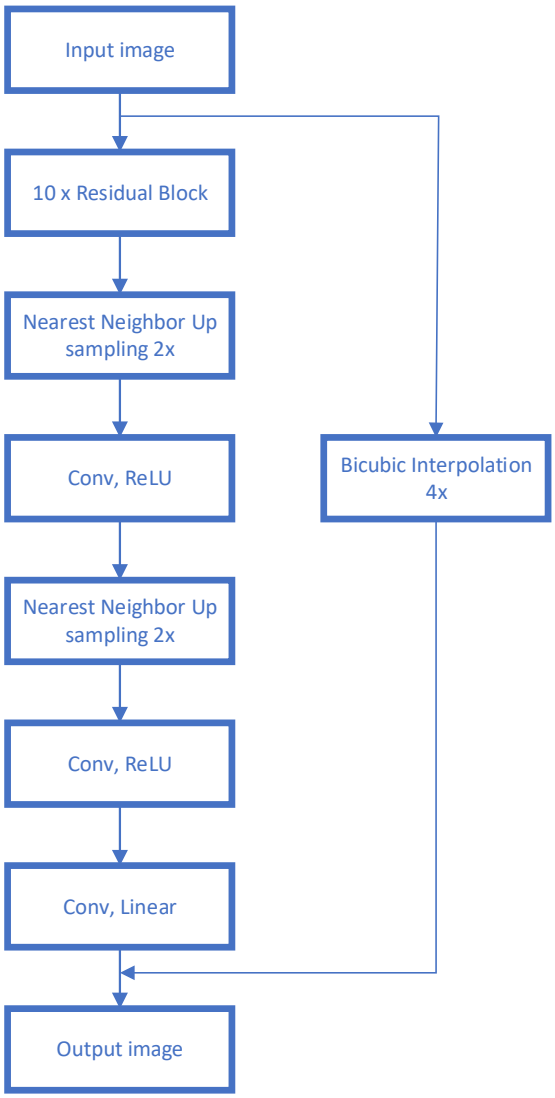


Residual block



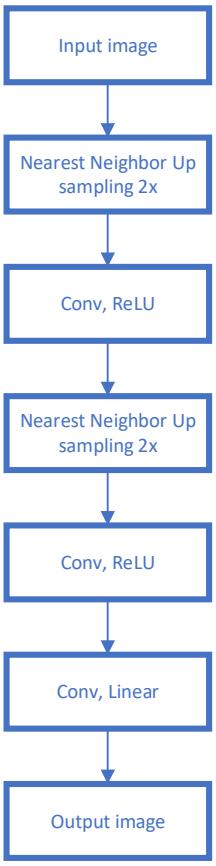
Base model. Used mostly in the initial evaluation. **(**CNN 1**)**

Then I have few models with added direct connection from the input image through a bicubic interpolation and summed with the last convolutional layer to produce the output image. The idea is here is that the model has to learn less since we have the bicubic interpolation in place.



Model with added bicubic interpolation. (CNN 2)

The last model I evaluated is without the residual blocks and without the bicubic interpolation. This is just for compare.

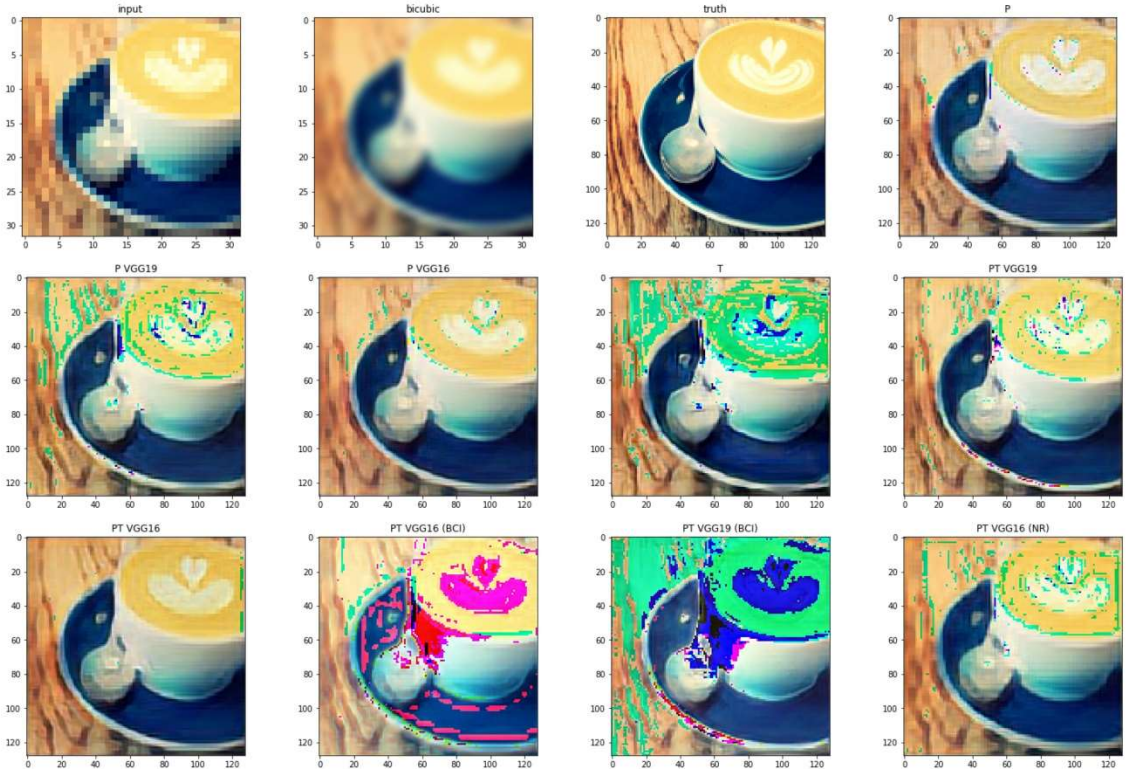


Model without residual blocks and without bicubic interpolation. (CNN 3)

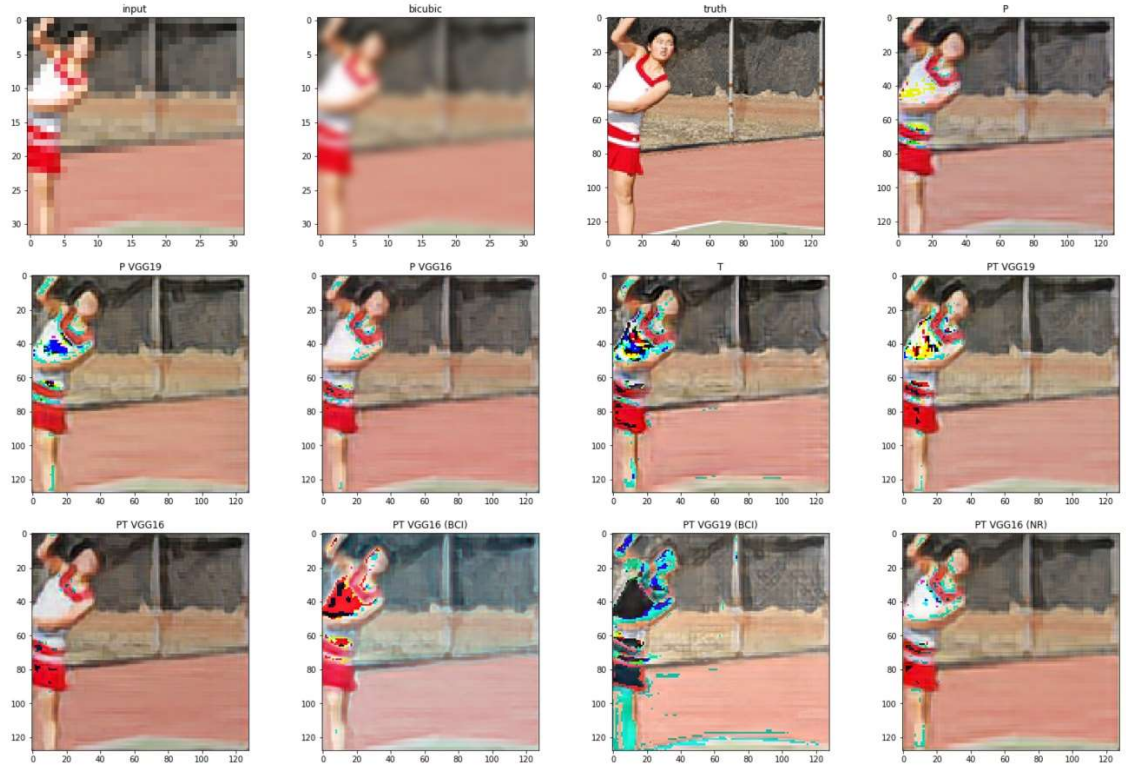
I performed an initial evaluation of the models with the small datasets (600 train images, 80 validation images and 20 test images). This was done on MS COCO and CelebA datasets. I used the standard train function in Keras, passing the validation data. Optimizer is Adam with the default learning rate of 0.001. Since I did the training on my laptop the batch size was set to 8 and after running this multiple time it was clear that 50 epochs are enough.

Used Jupyter notebook: *image_super_resolution_initial.ipynb* with these parameters:
use_small_dataset = True
use_dataset_celeba = False
disable_training = False
load_weights_before_training = False
All of the models have to be enabled.
train_epochs = 100
train_batch_size = 32
test_image_index_to_show = range(20)
optimizer = Adam(lr=0.001)

Here are some of the test results for MS COCO and CelebA.

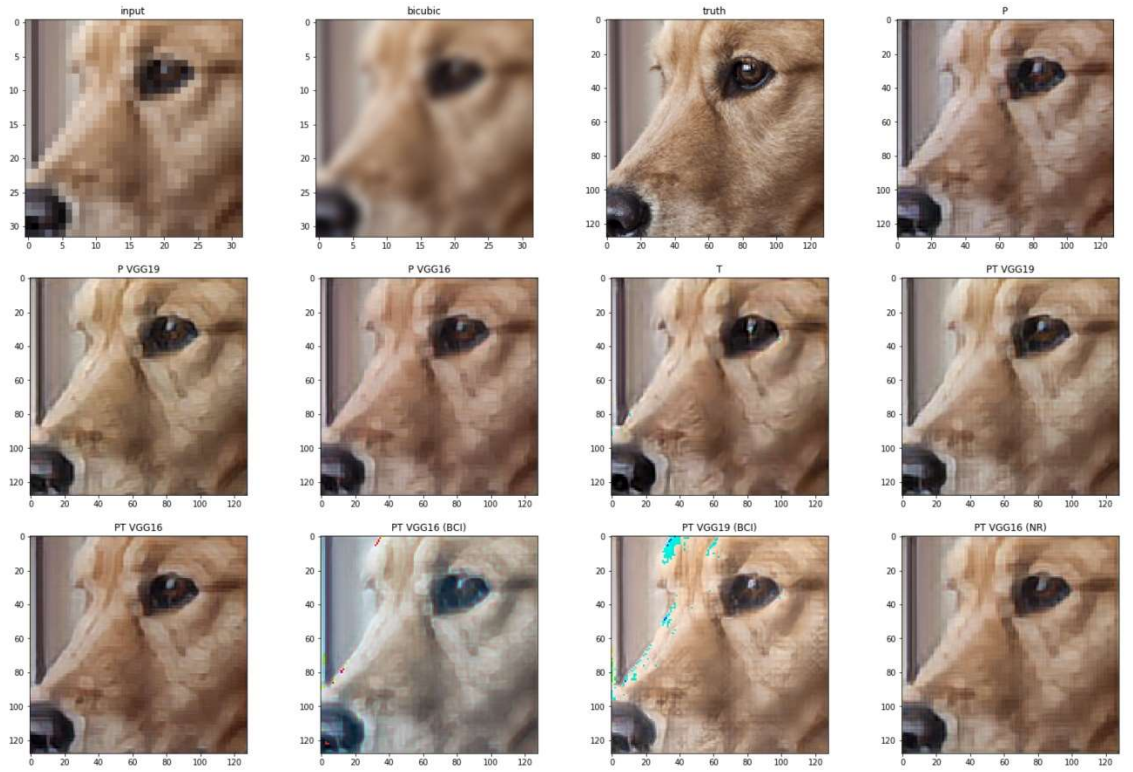


MS COCO initial evaluation. Image 1.



MS COCO initial evaluation. Image 3.

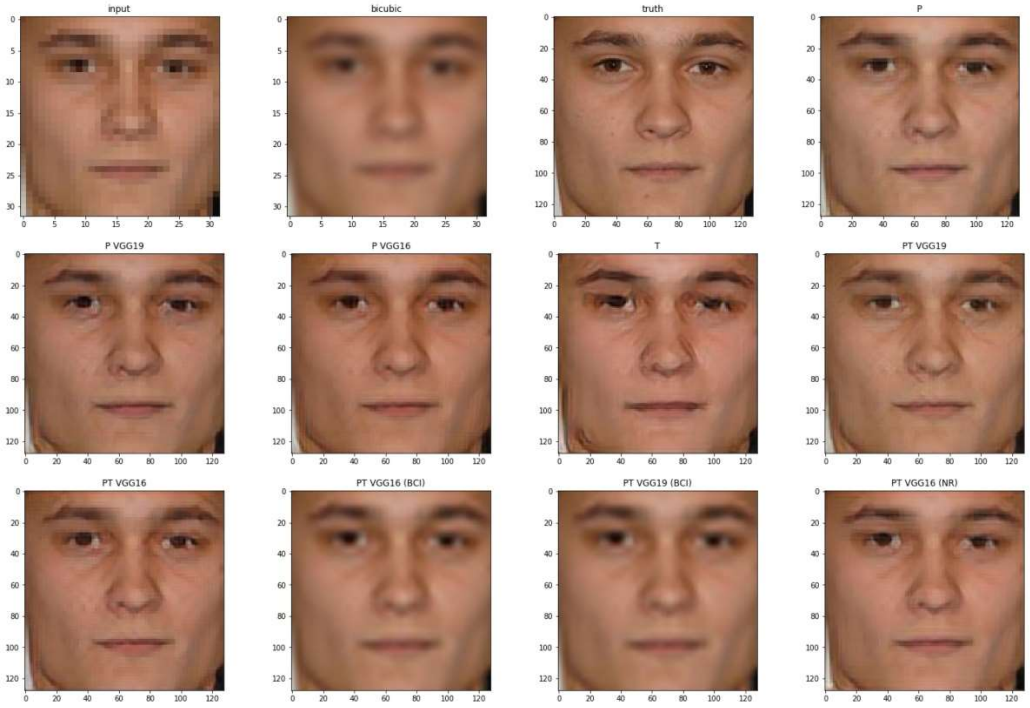
MS COCO initial evaluation. Image 5.



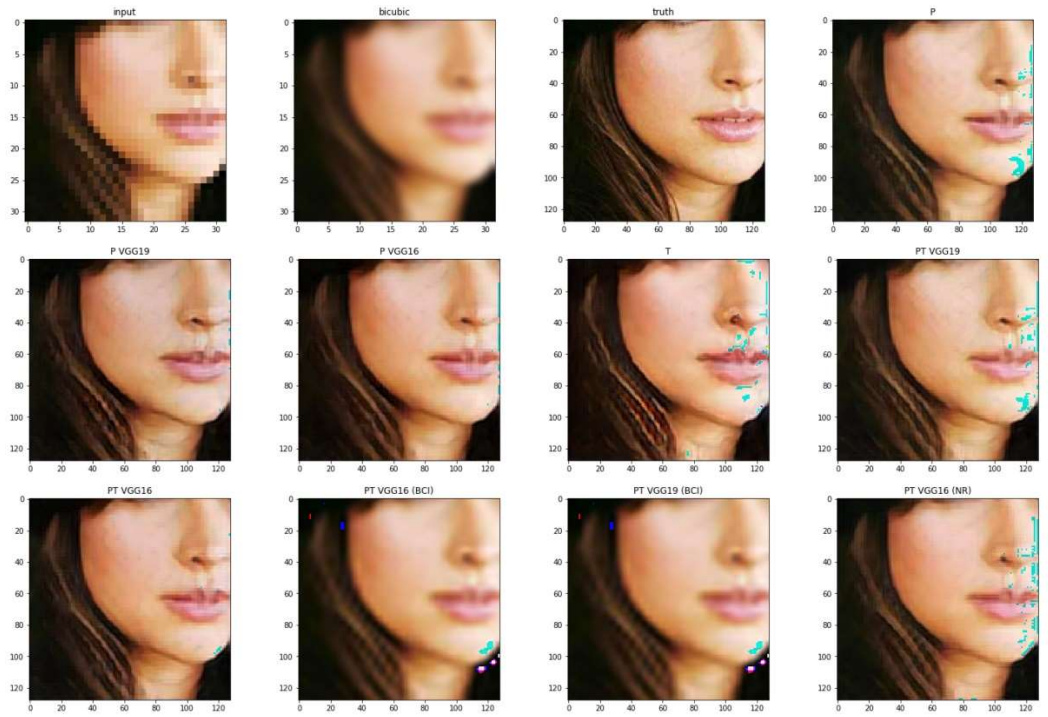
MS COCO initial evaluation. Image 13.

For the initial evaluation of CelebA data set I used the same parameters as for the MS COCO, except for this one:

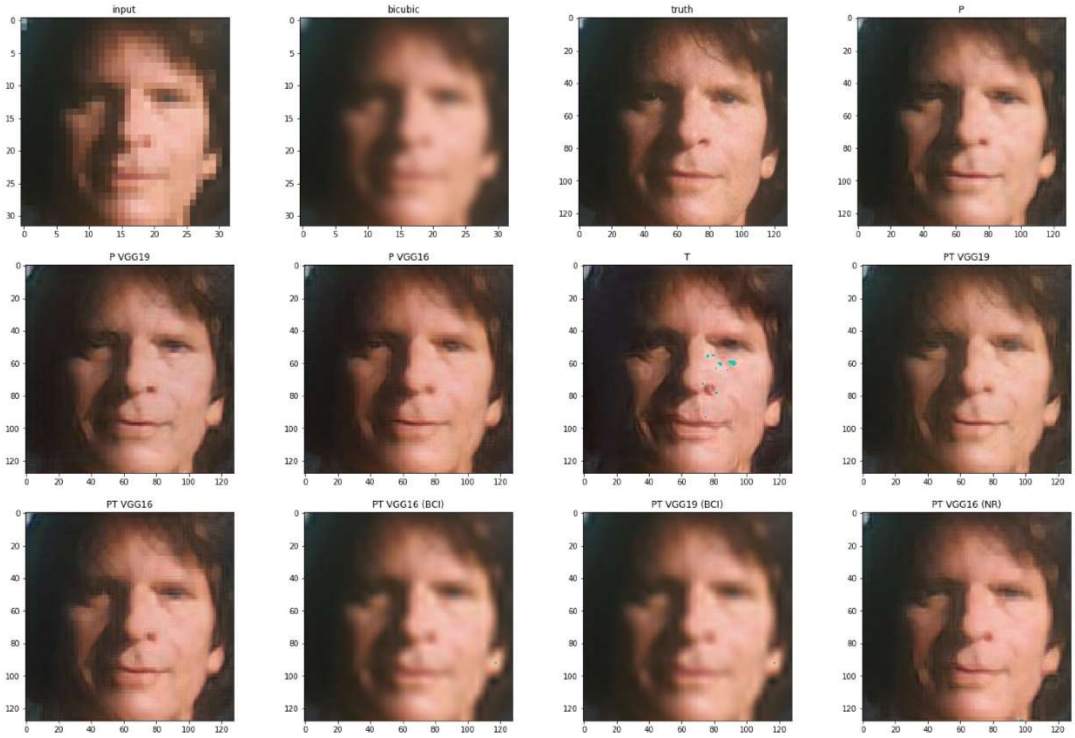
use_dataset_celeba = True



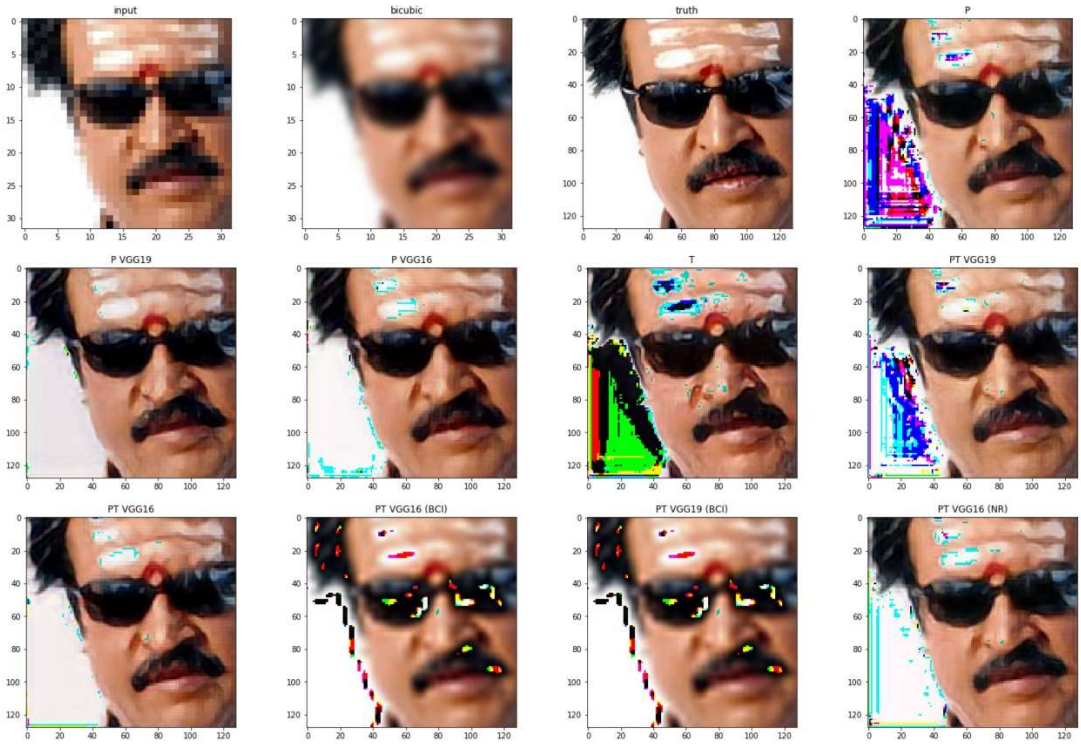
CelebA initial evaluation. Image 3.



CelebA initial evaluation. Image 4.



CelebA initial evaluation. Image 5.



CelebA initial evaluation. Image 15.

CelebA initial evaluation. Image 18.

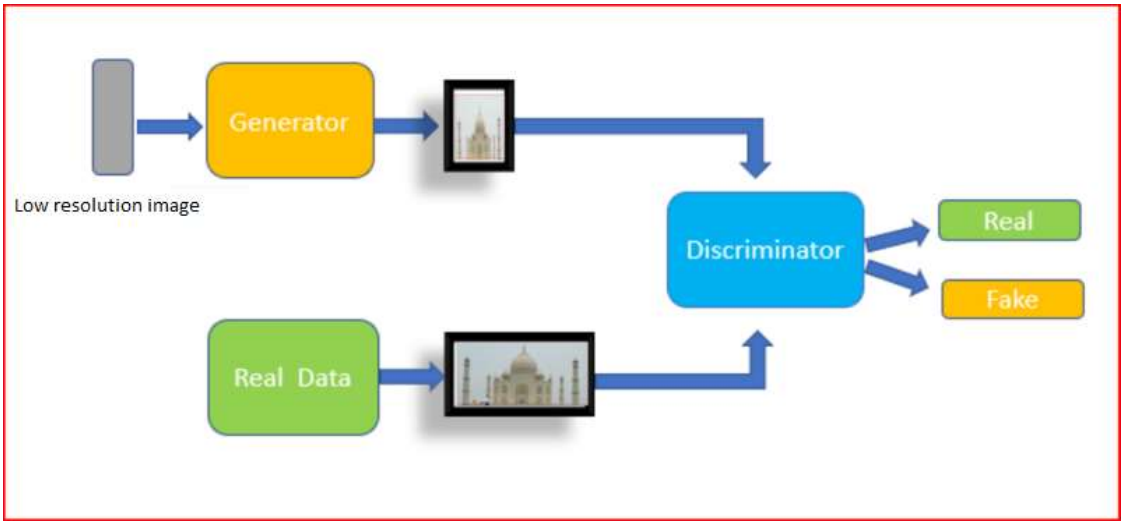
After the initial evaluation I decided to focus on only few models and train them on the full datasets. When I tried to train my model on ~250,000 images (MS COCO) I was not able because of not enough memory on my computer. So far, I was trying to load all of the images in the memory and then start the training. This is faster for training but it doesn't work very well on large datasets and with not enough resources. I was using the *fit()* function in Keras. They have a *fit_generator()* function, which is useful for large datasets. I implemented my generator (*DataGenerator*) which loads the data on batches. The drawback of using the generator is the GPU is not as well utilized as when all of the images are in memory.

Use the python script - *image_super_resolution_generator.py*

The last approach is using a Generative Adversarial Network (GAN). It can be started from *image_super_resolution.py*.

What is GAN. Generative Adversarial Networks (GAN) is one of the most promising recent developments in Deep Learning. GAN, introduced by Ian Goodfellow in 2014, attacks the problem of unsupervised learning by training two deep networks, called Generator and Discriminator, that compete and cooperate with each other. In the course of training, both networks eventually learn how to perform their tasks. Generator to produce as close as possible result to the real image in this case and the discriminator to recognize which image is real and which is generated.

In order to train the discriminator, we pass two set of inputs – real images and generated images.



I selected 3 models from the initial once I trained them on the full dataset. One as suggested in the research paper I based my project on - PT BCI – I trained with GAN for 24 hours.

The other two PT and PT16 I trained without GAN. From all 3 of the models I pick PT16. This a model that is based on a perceptual loss calculated with the help of VGG16 network and texture loss.

Refinement

I did a lot of refinement of the loss functions on the phase of initial evaluation. First, I started with all weights of the losses set to 1 and then played with them. At the end I picked the suggested weight for the texture loss as in the research paper. For the perceptual loss I kept the weight of 1. The other thing I tried is using different pooling layers as output. “block3_pool” was the best for me.

The way training is done was modified as well as mentioned above. I was initially loading all of the images which turned out to be a problem when loading a lot of them. Even though the GPU load was 99% when images a pre-loaded I picked to use loading on the fly. It is faster to test new things. Loading all of the images first sometimes takes about 30 minutes.

I trained the model PT16 with different values of the learning rate and the best for me is 0.0005. The previous used were 0.01, which gave me the best results on the smaller dataset in the initial training. The other option I tried was 0.0001, which didn’t provide good results.

IV. Results

Model Evaluation and Validation

Evaluation was done by conducting a survey asking to select an image that is as close to the original as possible. Since on some images my model produced different colors on pixels that were too bright in the input image the participants in the survey had a hard time on them to decide which is better. Over all they agreed that my model enhance better than the other models presented.

Additional evaluation of the model was performed by calculating the PSNR for the final images I have selected.

They are in folder “. \test_results\final_compare\CelebA”. Image_01 to Image_10 subfolders.

Image	Photoshop	PT16
Image_01	21.58	**22.86**
Image_02	24.18	**25.68**
Image_03	23.88	**24.54**
Image_04	26.90	**28.85**
Image_05	**23.48**	21.08
Image_06	25.89	**27.77**
Image_07	22.98	**23.04**
Image_08	**18.56**	17.69
Image_09	**19.46**	18.47
Image_10	18.96	**19.54**

As we can see in 70% of the cases the suggested model is giving better PSNR than the Photoshop enhanced image.

The software I used to calculate PSNR:

https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Others/PSNR.shtml

Justification

Even the results are not as I expected, they are still very good and satisfying. The enhancement is still better than Paint (no brainer here, so I don't even present it in the visualization) and Photoshop. Even though photoshop has always the correct colors, the image is not as sharp as the generated from my model.

We can see this also from the PSNR evaluation and the results in the table above.

The only drawback I saw so far with the model is that often if the input image has bright pixels, they are being converted to some other color pixels. Mostly green, but sometimes

black. This is the only thing I cannot trust the model to be released for mass use. Besides that, it is very robust and produce very good results.

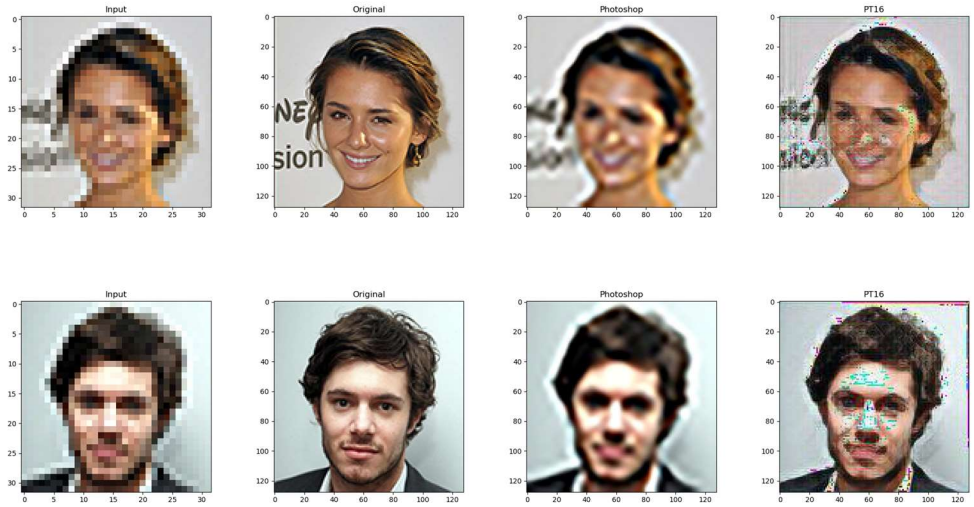
With all that said I think the model itself can be used as a base so other people can build on top of it if they need to solve similar issue.

V. Conclusion

Free-Form Visualization

You can see here some images reduced from the original size 4 times and fed to the model (PT16) and to Photoshop for enhancement. I picked the best photoshop enhancement algorithm I saw (I tried all of them).

The next images were reduced too much so it is really hard to enhance.



Reflection

The process used for this project can be summarized with these steps:

1. Research about the selected problem to see what work was done already
2. The data was download and preprocessed. Filtered the small images and split to the corresponding functions.
3. Loss functions were implemented
4. Convolutional Neural Network models were selected
5. Initial evaluation of the models was performed on smaller dataset
6. Training was performed on a large dataset on only few models
7. Training was performed on a large dataset using GAN

Training on the large datasets turn out to be very challenging and time consuming. I had to implement some additional functionality so the data is loaded on demand. So far all of the examples were showing loading on start, but this doesn't work well on large datasets.

Overall this was very useful and productive in a sense that I learned a lot. Unfortunately, the results are not so good as I expected and I have more work to do if this project need to be production ready.

This project showed me that with the use of ML and CNN and a little bit more time dedicated on this we can achieve a lot in enhancing image quality. Even professional grade software like Photoshop using just computer algorithms cannot do as good as a CNN trained for few days.

Improvement

The GAN grid trainer (Grid Search) class can be finished and a long exhausting training can be performed, to find better parameters for the loss functions and the learning rate. I can test even with different neural networks.

I was not able to find why sometimes the white on the input image got converted to green, black or some other dark color on the output (estimated) image. This is something that need to be fixed.

The entire structure of the project can be made a little more organized and some code can be combined between the current files.

References

[1] https://arxiv.org/abs/1612.07919
[2] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In NIPS, 2015.
[3] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer