AFEKA TEL-AVIV ACADEMIC COLLEGE OF ENGINEERING

Electrical Engineering Department

שם הפרויקט: שילוב של סופר רזולוציה וסגמנטציה.

Project's Name: Joint Super-Resolution and Semantic Segmentation.

Projects Definition: Project Book

Students Name: Ran Mishael
Moderator's Name: Dr. Alexander Apartsin

Table Of Content

1. Executive Summary 5

2. Introduction 6

3. Project Objectives 8

4. Research plan 9

5. Potential Applications 10

6. Project metrics 10

7. Training and evaluating the datasets 10

8. Prior Art 11

9. Our Contribution 12

10. Alternatives to the project implementation 13

11. Block Diagram 13

12. Project's Outcomes 14

13. Means and tools required 14

14. The knowledge gaps i overcame 14

15. Preliminary Design: SS and SR models 14

16. Dataset preparation 16

17. Semantic Segmentation on Super-Resolved Images 17

18. Training Semantic Segmentation with Super Resolved images. 21

19. Super-Resolution with Segmentation Priors. 22

20. System Requirements / Libraries Required 22

21. Project Outputs 22

22. Conclusions 23

23. Future suggestions 23

24. Acknowledgement 23

25. References

26. Poster 27

Figures:

Figure. 1. Super Resolution Architecture by Deep CNN with Skip Connection and Network in Network.
Figure. 2. Semantic Segmentation illustration of the SegNet architecture.
Figure. 3. High-level diagram of a joint segmentation and super-resolution neural inference model.
Figure. 4: Generation of an LR Image from an HR Image by DownSampling. (Solid Squares) LR Image Pixels x and (Squares) Missing HR Pixels y
Figure. 5: Comparing different SR approaches with downsampling factor ×4 focusing on a small, heavily detailed areas.

Tables:

Table 1: Available datasets for semantic segmentation.
Table 2: The performance of state-of-the art semantic segmentation Architecture.
Table 3: The performance of state-of-the art super-resolution methods.
Table 4: SS and SR performed on different scale factors.

Diagrams:

Diagram 1: The serial combination of super-resolution and segmentation models.
Diagram 2: Joint segmentation and superresolution model.

Graphs:

Graph 1: PSNR - Scale factor results on dcscn SR model.
Graph 2: PACC - Scale Factor before and after applying super resolution to the dataset.
Graph 3: PACC - Scale factor improvement in % of the super resolved images prior to SS results in improving the overall class pixel accuracy.

Images:

image. 1. *Semantic Segmentation [38] results on x2 scaled image*
image. 2. *Super Resolution Result on x4 scaled image, LR image on the left, SR output on the right,*
image. 3. *Image Sharpen with matlab on Bicubic LR image.*
Image. 4. *This image is an example on how the super resolution helps the semantic segmentation to identify the legs of the bike rider. On the left, a x4 scaled image and on the right the x4 scaled super resolved image.*

1. Executive Summary

In most digital imaging applications, high resolution images or videos are usually desired for later image processing and analysis. The desire for high image resolution stems from two principal application areas: improvement of pictorial information for human interpretation; and helping representation for automatic machine perception. Image resolution describes the details contained in an image, the higher the resolution, the more image details. The resolution of a digital image is most commonly described in pixel resolution.

Modern image sensor is typically a charge-coupled device (CCD) or a complementary metal-oxide-semiconductor (CMOS) active-pixel sensor. These sensors are typically arranged in a two dimensional array to capture two-dimensional image signals. The sensor size or equivalently the number of sensor elements per unit area in the first place determines the spatial resolution of the image to capture. The higher density of the sensors, the higher spatial resolution possible of the imaging system. An imaging system with inadequate detectors will generate low resolution images with blocky effects, due to the aliasing from low spatial sampling frequency. In order to increase the spatial resolution of an imaging system, one straightforward way is to increase the sensor density by reducing the sensor size. However, as the sensor size decreases, the amount of light incident on each sensor also decreases, causing the so called shot noise. Also, the hardware cost of sensor increases with the increase of sensor density or correspondingly image pixel density. Therefore, the hardware limitation on the size of the sensor restricts the spatial resolution of an image that can be capture

While the image sensors limit the spatial resolution of the image, the image details (high frequency bands) are also limited by the optics, due to lens blurs (associated with the sensor point spread function (PSF)), lens aberration effects, aperture diffractions and optical blurring due to motion. Constructing imaging chips and optical components to capture very high-resolution images is prohibitively expensive and not practical in most real applications, e.g., widely used surveillance cameras and cell phone built-in cameras. Besides the cost, the resolution of a surveillance camera is also limited in the camera speed and hardware storage. In some other scenarios such as satellite imagery, it is difficult to use high resolution sensors due to physical constraints. Another way to address this problem is to accept the image degradations and use signal processing to post process the captured images, to trade off computational cost with the hardware cost. These techniques are specifically referred as SuperResolution (SR) reconstruction.

Super-resolution (SR) are techniques that construct high-resolution (HR) images from several observed low-resolution (LR) images, thereby increasing the high frequency components and removing the degradations caused by the imaging process of the low resolution camera. The basic idea behind SR is to combine the non-redundant information contained in multiple low-resolution frames to generate a high-resolution image.

Semantic image segmentation is a fundamental task in computer vision. It predicts dense labels for all pixels in the image, and is regarded as a very important

task that can help deep understanding of scene, objects, and human. Development of recent deep convolutional neural networks (CNNs) makes remarkable progress on semantic segmentation. The effectiveness of these networks largely depends on the sophisticated model design regarding depth and width, which has to involve many operations and parameters. CNN-based semantic segmentation mainly exploits fully convolutional networks (FCNs). It is common wisdom now that increase of result accuracy almost means more operations, especially for pixel-level prediction tasks like semantic segmentation.

In this project we strive to improve the overall accuracy of the Semantic Segmentation techniques by training and using the Super resolved images.

2. Introduction

There are two most common types of single image processing tasks: image enhancement and object recognition. The image enhancement tasks [1] deal with the removal of noise and other undesirable distortions while boosting image content which is important for human observer [2] or for downstream processing tasks [3]. The recognition or detection tasks deal with labeling an input image or some part of the image with semantic information regarding the type or/and attributes of objects appearing in the image[4].

The approach of super-resolution[5] image enhancement takes the idea of image content boosting to extremes by adding or “hallucinating” content that wasn’t present in the original noisy and distorted image [6]. The added content is generated based on the probabilistic model of the most plausible high-resolution content given its low resolution [7], noisy and distorted version of the same content (e.g. image patch) as shown in Figure 1. This approach is similar to manual retouching of photographs that was common before the photoshop era [8]. An artist would recognize image patch as belonging to a certain texture or object and, then, would add the most plausible details given the context and perceived object types in the image. Modern super-resolution techniques are usually based on the deep learning models 9] that are trained using a large number of examples [10]. Each example is represented using a pair of low-resolution image patch and desirable high-resolution output. Usually, some regularization is added to ensure consistency between neighboring output patches [9].

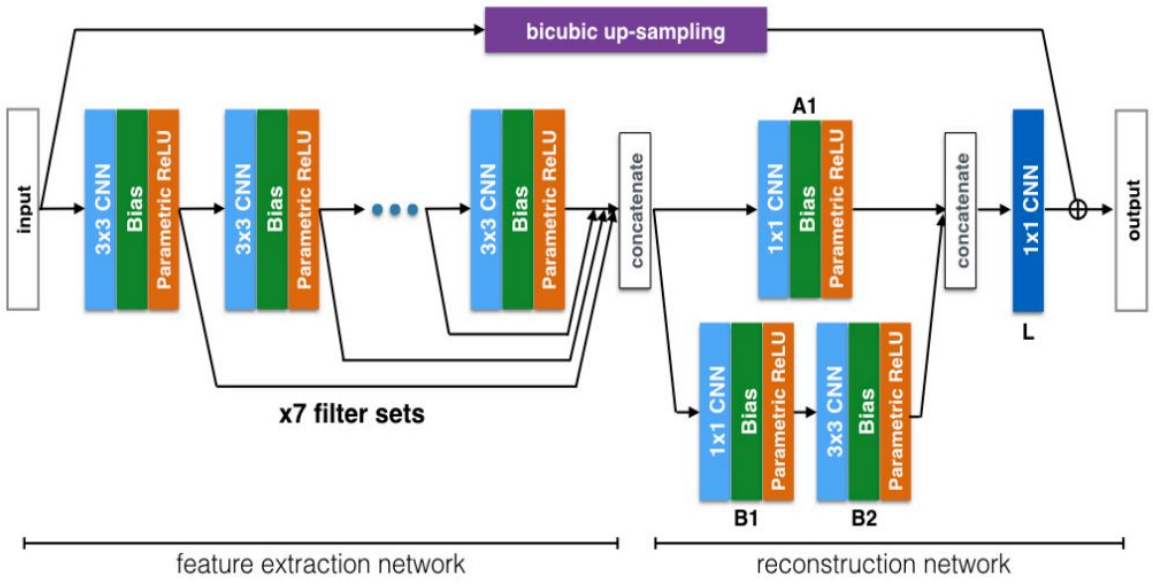


Figure 1. *Super Resolution Architecture [11] by Deep CNN with Skip Connection and Network in Network.*

The semantic segmentation task [12] is the most basic recognition task (but not necessarily the easiest one). A semantic segmentation algorithm assigns a label to each pixel in the image that describes an object instance and types that each pixel belongs to as shown in Figure 2. Therefore, semantic segmentation partitions the image into an objects separating not only objects of different classes but also distinguishing between different objects of the same class. Modern semantic segmentation models are also based on deep neural networks that are trained with a large number of examples [13]. Each training example consists of the pair that includes an input image and desired semantic segmentation map (ground truth). The ground truth is usually created using manual [14] or semi-automatic [15] labeling of large image dataset.

Usually, the image enhancement processing and super-resolution, in particular, are applied as a preprocessing task before applying high-level computer vision tasks including object detection and recognition. However, for modern deep learning probabilistic model, there is a much complex relation between enhancement and recognition. While high-quality image content improves the accuracy of recognition, the knowledge about object types and context plays a critical role in the super-resolution process. Therefore, it makes sense to consider and elaborate the interplay between enhancement and recognition and, specifically, between super-resolution approaches for image enhancement and semantic image segmentation.

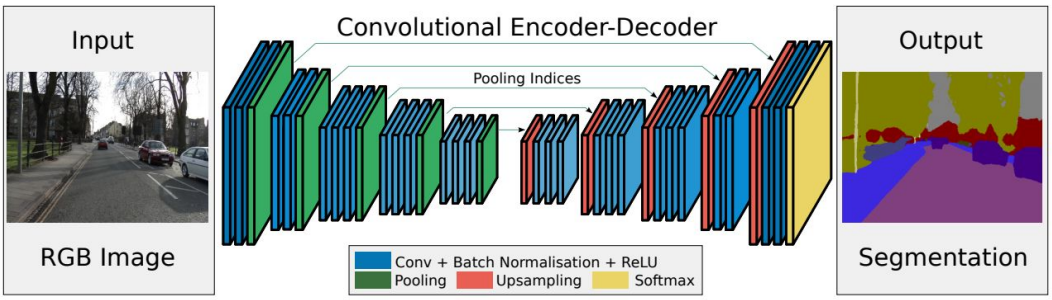


Figure 2. *Semantic Segmentation An illustration of the SegNet architecture [27]. A decoder up-samples its input using the transferred pool indices from its encoder to produce a sparse feature map(s).. The final decoder output feature maps are fed to a softmax classifier for pixel-wise classification.*

The goal of this research is to explore this relationship and to develop a number of models for joint super-resolution and semantic segmentation as shown in Figure 3. We will consider state-of-the-art deep learning architectures for super-resolution and semantic segmentation and will try to combine them sequentially and in parallel for joint training and inference.

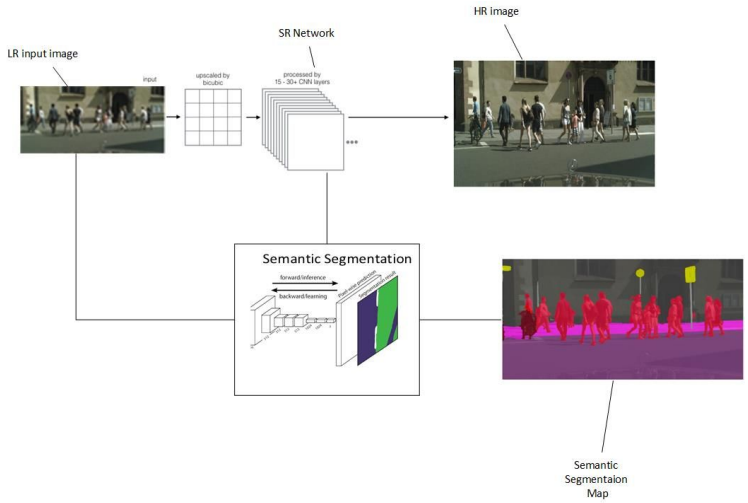


Figure. 3. *High-level diagram of a joint segmentation and super-resolution neural inference model.*

3. Project Objectives

The objective of this project is, to explore various neural network architectures for combining semantic segmentation and super-resolution. Ultimately, we'd like to construct a joint network that is trained for performing both tasks simultaneously. Given this goal, it important to decide what quality metric we wish to optimize. The super-resolution and semantic segmentation algorithms are usually trained and evaluated using completely different criteria. The super-resolution models are tuned for maximizing a specific image quality assessment metrics [20]. Among common image

quality metrics are PSNR and various version of perceived quality like SSIM [21]. On the other hand, semantic segmentation model is trained to minimize per-pixel classification error (e.g. pixel-wise cross entropy, [13]). Clearly, the two types of metrics are incompatible which pose certain difficulties with regard to combining super-resolution and semantic segmentation models

For this project, we are going to consider a number of alternatives for the evaluation criteria. First, we are going to consider SR-assisted semantic segmentation and segmentation -assisted super-resolution tasks. In these tasks, there is the main network(segmentation and enhancement respectively) that uses the output of the secondary network (enhancement and segmentation respectively) for improving the outcomes of the main networks. Perhaps, the serial combination of two networks[14] (secondary followed by main) seems to be the most adequate as depicted in the figure 3. In this framework, the overall evaluation criteria are chosen according to the task of the main network.

However, the most promising approach might be a unified model for super-resolution and semantic segmentation that is trained jointly to optimize combined criteria as schematically presented in the figure 3 One of the project goals is to design such combined criteria. One of the possible approaches might be to consider a combined network as an augmentation network that produces enhancement and labeled image for a next high-level vision task (e.g. pedestrian recognition for autonomous driving). In this case, we might use a performance of the downstream task as an evaluation criterion for the joint model.

4. Research plan

The overall project might be partitioned into the following sequence of research tasks

#	Task	Description	Due date
1	Dataset selection and baseline evaluation	Need to select dataset that can be used for both semantic segmentation and SR tasks. The data should contain semantic segmentation ground truth for training. We can always recreate SR training by downsampling the original images. Also, select initial SR and SS network architectures and evaluate their individual performance on the selected dataset	20/12/18

2	SR-assisted segmentation	Select state-of-the-art SR and SS network and try to chain them into a single network. First, by evaluation of individually trained models and then for jointed training	1/2/19
3	SS-assisted super-resolution	Reverse the order between networks. First, adopt the SR to work on 4 channel (RGB+label) and then train the combined network	28/3/19
4	Joint model training with combined criteria	Design and train a joint architecture for two tasks	1/6/19
5	Evaluation of high-level downstream task	Evaluate the joint model using a high-level downstream vision task	9/7/19

5. Potential Applications

Improving image quality and image segmentation might assist in a number of video analytic tasks including object detection, recognition, and tracking. Super Resolution could extend video analytic capabilities, to objects of smaller dimensions (e.g. extending operational distance for face recognition) or alternatively, could allow saving in camera sensors size or network bandwidth, due to the reduction in required image resolution.

6. Project metrics

Any improvement in the following metrics indicates the validity of the approach. We are going to explore three methods of improvement. The first method aims for PSNR [36] improvement which stands for peak signal to noise ratio, PSNR is most easily defined via the mean squared error (MSE). Given a noise-free m×n monochrome image “I” and its noisy approximation “K”, MSE is defined as:

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \cdot \sum_{i=0}^{m-1} \left[I(i,j) - K(i,j) \right]^2$$

The PSNR (in dB) is defined as:

$$PSNR = 10 \cdot \log_{10}\left(\frac{{MAX_I}^2}{MSE} \right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}} \right) = 20 \cdot \log_{10}\left(MAX_I \right) - 10 \cdot \log_{10}(MSE)$$

Here, MAX_I" is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. More generally, when samples are represented using linear PCM with B bits per sample, MAX"I" is 2B−1.

The 2nd method aims for PACC (pixel accuracy) improvement. In order to measure the performance of semantic segmentation results. We use the following evaluation metric. Specifically, let nii be the P number of pixels of class i predicted to be class j and ti will be the total number of pixels of class i.

$$PACC = \frac{\sum_i n_{ii}}{\sum_i t_i}$$

The 3rd method is aiming to improve the average between PSNR and PACC

7. Training and evaluating the datasets

We chose a dataset called CityScape by Google [40] containing 5000 images with ground truth that will be used to train the neural network. Semantic segmentation requires a detailed labeling of image pixels by object category. Information derived from local image patches is necessary to describe the detailed shape of individual objects.

8. Prior Art

Deep Learning (DL) is a machine learning technique that allows us to teach computers, giving them the ability to perform tasks that comes naturally to humans. Using DL you can perform classification tasks directly from an image, sound or text [16,17,18]. DL architectures such as deep neural networks, deep belief networks, and recurrent neural networks have been applied to fields including computer vision, speech

Table 4
Popular datasets for 2D/3D semantic segmentation.

Dataset	Scene type	Number		Annotation	Website
		images/scenes	classes		
MSRC [29]	outdoor	591	23	pixel-level labeling	https://www.microsoft.com/en-us/research/project/image-understanding/
LabelMe [151]	hybird	187,240	unlimited	bounding polygon	http://labelme2.csail.mit.edu/Release3.0/
PASCAL VOC [113]	hybird	21,738	20	hybrid	http://host.robots.ox.ac.uk/
ImageNet [152]	hybird	1,431,167	1000	ditto	http://image-net.org/download
MSC COCO [154]	hybird	328,000	91	object masks	http://mscoco.org/dataset/
SIFT-FLOW [93]	outdoor	2,688	33	pixel-level labeling	http://people.csail.mit.edu/celiu/LabelTransfer/code.html
SBD [32]	outdoor	715	8	pixel-level labeling	http://dags.stanford.edu/projects/scenedataset.html
CityScapes [155]	street	25,000	19	hybrid	https://www.cityscapes-dataset.com/
ADE20K [156]	hybird	20,000+	150	pixel-level labeling	http://groups.csail.mit.edu/vision/datasets/ADE20K/
NYU Depth V2 [157]	indoor	1,449	894	pixel-level labeling	http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
Washington RGB-D [158]	indoor	250,000	300	segmented object	http://rgbd-dataset.cs.washington.edu/
Cornell RGB-D [159]	indoor	52(point cloud)	undetailed	point level labeling	http://pr.cs.cornell.edu/sceneunderstanding/
Sun RGB-D [160]	indoor	10,335	undetailed	hybrid	http://rgbd.cs.princeton.edu/
ScanNet [161]	indoor	1513	20	voxel level labeling	http://www.scan-net.org/
VMR-Okaland-V2 [162]	outdoor	36(3D blocks)	7	point level labeling	http://www.cs.cmu.edu/%CB%9Cvmr/datasets/

Table 1: *Available datasets for semantic segmentation [28]*

recognition, natural language processing, audio recognition, social network filtering, machine translation and more, where they have produced results comparable to and in some cases superior to human experts. Deep learning models are vaguely inspired by information processing and communication patterns in biological nervous systems yet have various differences from the structural and functional properties of biological brains, which make them incompatible with neuroscience evidence.

Semantic segmentation is one of the first tasks we need in order to get a full understanding of an image's scene. The importance of scene understanding as a core computer vision problem is highlighted by the fact that an increasing number of applications nourish from inferring knowledge from imagery. Some of those applications include self-driving vehicles, human-computer interaction, virtual reality etc. With the popularity of deep learning in recent years, many semantic segmentation problems are being tackled using deep architectures, most often Convolutional Neural Networks, which surpass other approaches by a large margin in terms of accuracy and efficiency.

Table 2 summarizes state-of-the art results obtained by various semantic segmentation methods

9. Our Contribution

Previous work that has been done in the field as of today does not use both Semantic Segmentation and Super Resolution in order to improve the image quality.

This project is about researching and finding a way, to combine both methods, in order to gain a better result than the systems that exist at the moment.

Architecture	MS Test	Mean IoU	Pixel Accuracy	Overall Score	Training Time
ResNet18_dilated8 + c1_bilinear_deepsup	No	33.82	76.05	54.94	0.42 * 20 = 8.4 hours
	Yes	35.34	77.41	56.38	
ResNet18_dilated8 + ppm_bilinear_deepsup	No	38.00	78.64	58.32	1.1 * 20 = 22.0 hours
	Yes	38.81	79.29	59.05	
ResNet50_dilated8 + c1_bilinear_deepsup	No	34.88	76.54	55.71	1.38 * 20 = 27.6 hours
	Yes	35.49	77.53	56.66	
ResNet50_dilated8 + ppm_bilinear_deepsup	No	41.26	79.73	60.50	1.67 * 20 = 33.4 hours
	Yes	42.04	80.23	61.14	
ResNet101_dilated8 + ppm_bilinear_deepsup	No	42.19	80.59	61.39	3.82 * 25 = 95.5 hours
	Yes	42.53	80.91	61.72	
UperNet50	No	40.44	79.80	60.12	1.75 * 20 = 35.0 hours
	Yes	41.55	80.23	60.89	
UperNet101	No	41.98	80.63	61.34	2.5 * 25 = 62.5 hours
	Yes	42.66	81.01	61.84	

Table 2: *The performance of state-of-the art semantic segmentation Architecture [30]*

Eval. Mat	Scale	Bicubic	SC [50]	NE+LLE [4]	KK [25]	ANR [41]	A+ [41]	SRCNN
PSNR	2	33.66	-	35.77	36.20	35.83	36.54	**36.66**
	3	30.39	31.42	31.84	32.28	31.92	32.59	**32.75**
	4	28.42	-	29.61	30.03	29.69	30.28	**30.49**
SSIM	2	0.9299	-	0.9490	0.9511	0.9499	**0.9544**	0.9542
	3	0.8682	0.8821	0.8956	0.9033	0.8968	0.9088	**0.9090**
	4	0.8104	-	0.8402	0.8541	0.8419	0.8603	**0.8628**

Table 3: *The performance of state-of-the art super-resolution methods [29]*

10. Alternatives to the project implementation

- There are numerous algorithms for super-resolution and numerous algorithms for semantic segmentation a different project implementation can be done by choosing a different combination of these algorithms or using different architectures within [30].
- Changing the order between first recognizing the object and then improving the image with super-resolution.

- Training the deep network with different data sets will get different results to the project.

11. Block Diagram

We will test two possibilities regarding the construction of the system. In the 1st approach we will apply a Semantic Segmentation on the LR image. After we have the SS map we will apply SR on the LR image. This approach is illustrated by the diagram 1

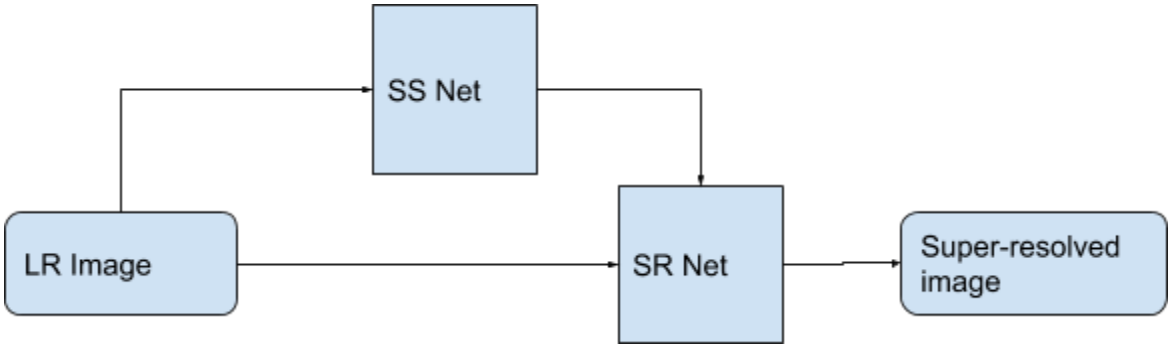


Diagram 1: *The serial combination of super-resolution and segmentation models*

We also will explore a joint model as described above. In the joint model, two networks will share information from intermediate layers as depicted in the diagram 2

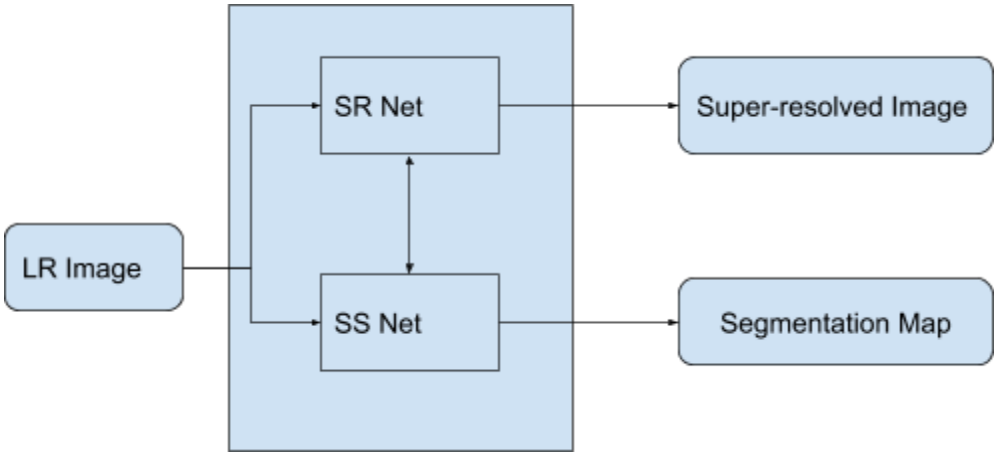


Diagram 2: *Joint segmentation and superresolution model*

12. Project's Outcomes

- A project will result in a machine learning model for joint segmentation and super-resolution task
- A software implementation for the model will be produced for evaluation and publications
- The evaluation results will be provided along with the model comparing various aspects of model performance against existing approaches

13. Means and tools required

Since this project focuses on the development of an image algorithm, the tools required are:

- A computer with relatively strong computing capabilities that can be used in the development stage
- A software language with the capabilities to run complex algorithms for example Python
- Python is an open source language so I will not need to buy a license for language development.
- Need a dataset with per-pixel labels of the objects that can be found on the internet
- Deep learning Python libraries such as TensorFlow, Numpy , openCV etc...

14. The knowledge gaps i overcame

- Udemy courses in the field of Machine learning (ML) ,Deep Learning (DL) and Computer Vision (CV).
- Learning the super-resolution model and implementation.
- Learning the model and implementation for object recognition and Semantic Segmentation.
- Obtaining a dataset with per-pixel labels of the objects for the network's training. and multiplying it into different image qualities.
- Creating multiple scale factor data sets with vast number of images.

15. Preliminary Design: SS and SR models

There are a number of Semantic Segmentation and Super Resolution models that has been developed recently. We have selected same-library implementations of the state-of-the-art semantic segmentation and super-resolution models. Consistent implementation makes integration and joint training of the model much simpler .The library chosen is TensorFlow [37] by Google, which is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end application programming interface for building applications with the framework, while executing those applications in high-performance C++.

The algorithm chosen for the Semantic Segmentation model [38] and Super Resolution model [39,42] both use TensorFlow as their computation library, Tensorflow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (central processing units (CPUs), graphical processing units (GPUs)), it

comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. Python version 3.6.6 was chosen as our Application Programming Interface (API).

image. 1. *Semantic Segmentation [38] results on x2 scaled image. The zoom-in segment of the image for other scale factors are displayed in the* ***table 4*** *below*

image. 2. *Super Resolution [39] Result on x4 scaled image, LR image on the left, SR output on the right,*

image. 3. *Image Sharpen with matlab on Bicubic LR image.*

For better understanding of the CNN Super Resolution difference from other algorithms i used the same input image with matlab's image sharpening tool as shown in image 3. It is easy to see the differences between the CNN based algorithm and the sharpening filter.

16. Dataset preparation

The dataset chosen for the Semantic Segmentation training is the Cityscape dataset [40], CityScape is a large-scale dataset that contains a diverse set of stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5 000 frames in addition to a larger set of 20 000 weakly annotated frames, each image scales are . The dataset is thus an order of magnitude larger than similar previous attempts, and contains 30 classes [44]. The original images have been downscaled and upscaled by factors of 2,3 and 4 using bicubic interpolation [43] .

In bicubic interpolation sixteen nearest neighbors of a pixel have been considered as shown in **Figure 4**. The intensity value assigned to point (x,y) is obtained using the equation:

$$V(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

Where the sixteen coefficients are determined from the sixteen equations in sixteen unknowns that can be written using the sixteen nearest neighbors of point (x,y). Bicubic interpolation is the standard used in commercial image editing programs, such as Adobe Photoshop and Corel Photo-paint [45].

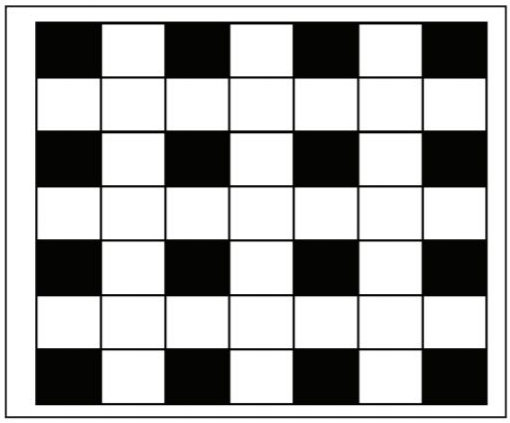


Figure. 4: *Generation of an LR Image from an HR Image by DownSampling. (Solid Squares) LR Image Pixels x and (Squares) Missing HR Pixels y*

Therefore, we have created 4 versions of each image corresponding to different level of details preserved in the image. The generated data will be used for baseline performance evaluation as well as the degradation in performance of SS and SR models as the function of image quality. Using this baseline, we will evaluate the gain in performance achieved through joint application and, later, joint training of super resolution and segmentation models.

17. Semantic Segmentation on Super-Resolved Images

We have evaluated pixel accuracy of the selected model on the original dataset and on downscaled versions of the same images. The resulting PACC evaluations serves as a baseline to our joint model testing if both SS and SR can benefit from sharing information as shown in **Diagram 2.**

The pixel accuracy evaluated while testing the pretrained weights on unscaled images is 0.90235%. We wanted to see the different accuracies with different scale factors going from non scaled images, all the way to x4 scaled images. For that cause, 4 scale factor datasets were made. After testing the sets on unresolved images we used Super Resolution on each set of the scaled images, creating three super resolved datasets (x2,x3,x4) of the same set we used for semantic segmentation. We tested the super resolved sets with the same pre-trained weights and found that the super

resolution had a positive effect on the semantic segmentation overall PACC performance as shown in **Graph 2.**

The Super Resolution model was Pre-trained for 1 epoch with the DIV2k dataset [41] with 150,000 48x48 images.

Image. 4. *This image is an example on how the super resolution helps the semantic segmentation to identify the legs of the bike rider. On the left, a x4 scaled image and on the right the x4 scaled, super resolved image.*

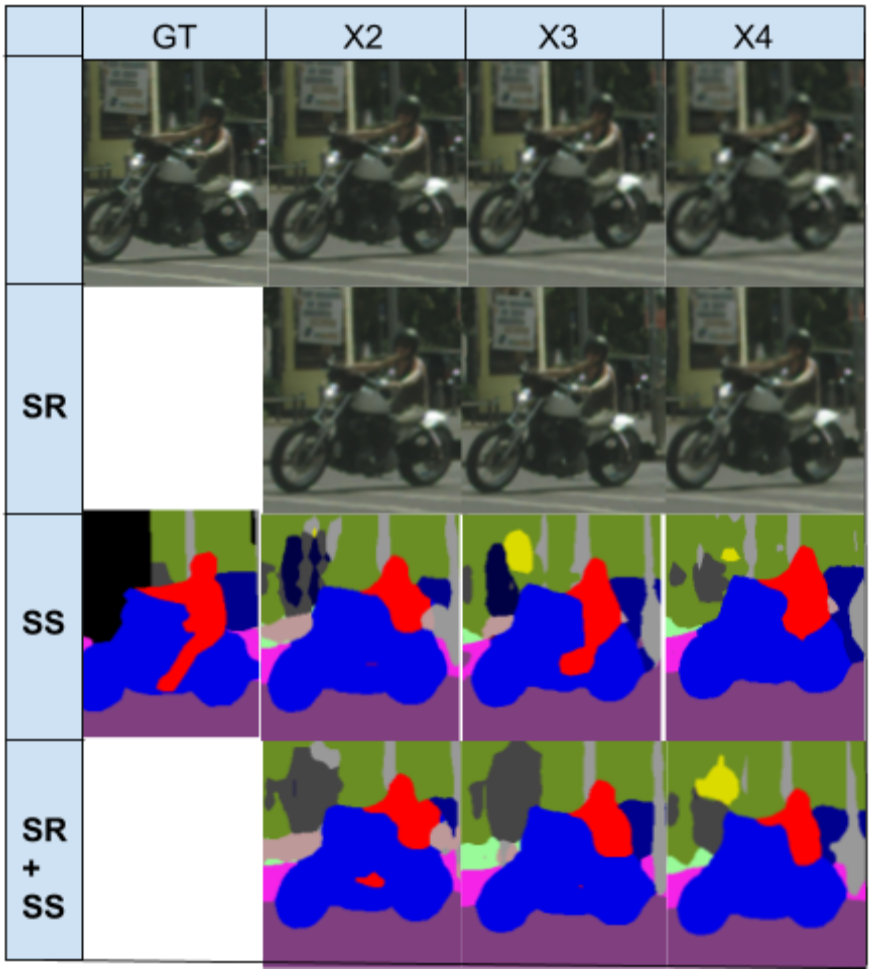
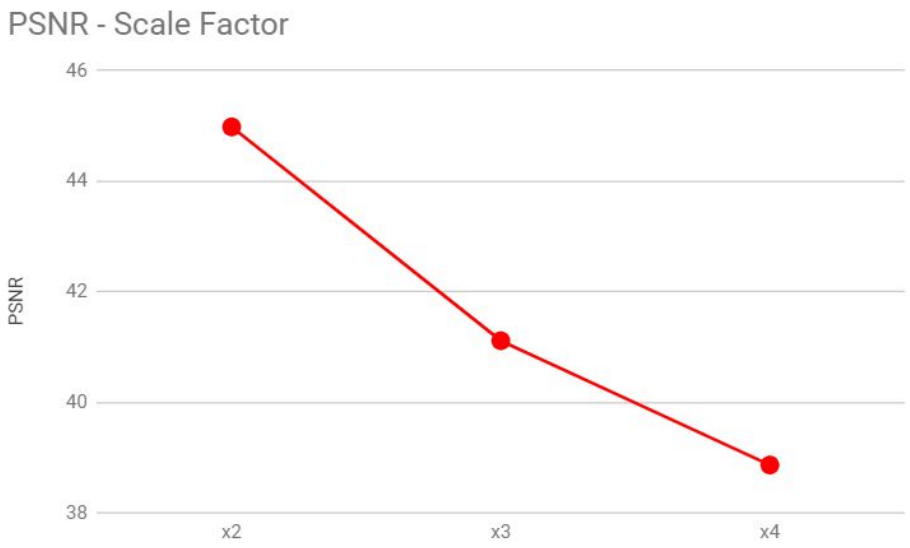


Table 4: *SS and SR performed on Cityscape dataset with different scale factors*

In **Table. 4** above there is an example of the segmentation map produced by the SS model on downscaled images from the CityScape set. The PACC is descending coordinated with the scale factor as shown in **Graph. 2.** after enhancing the overall pixel accuracy improved. The most significant improvement was on higher scale factors. We can assume that the higher the scale factor is, the better PACC improvement on super resolved images.

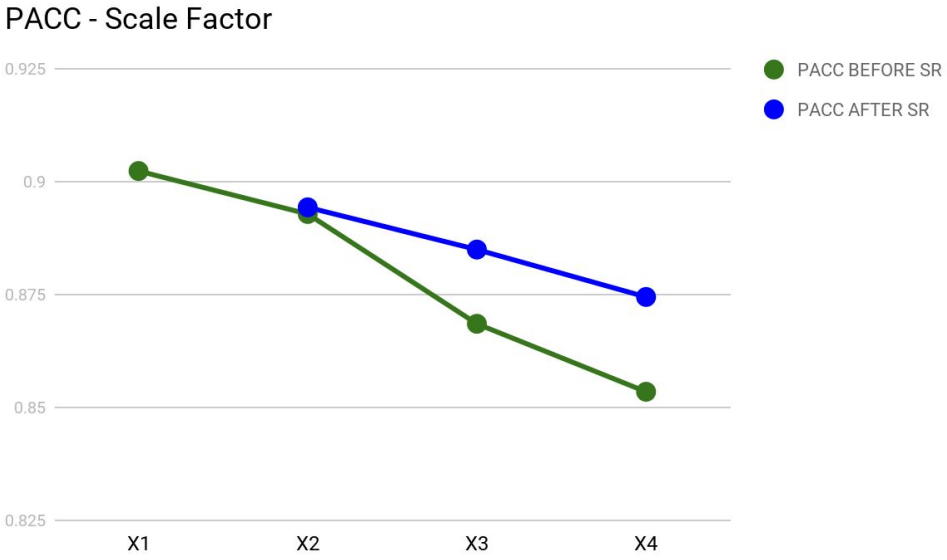
Super Resolution Results:



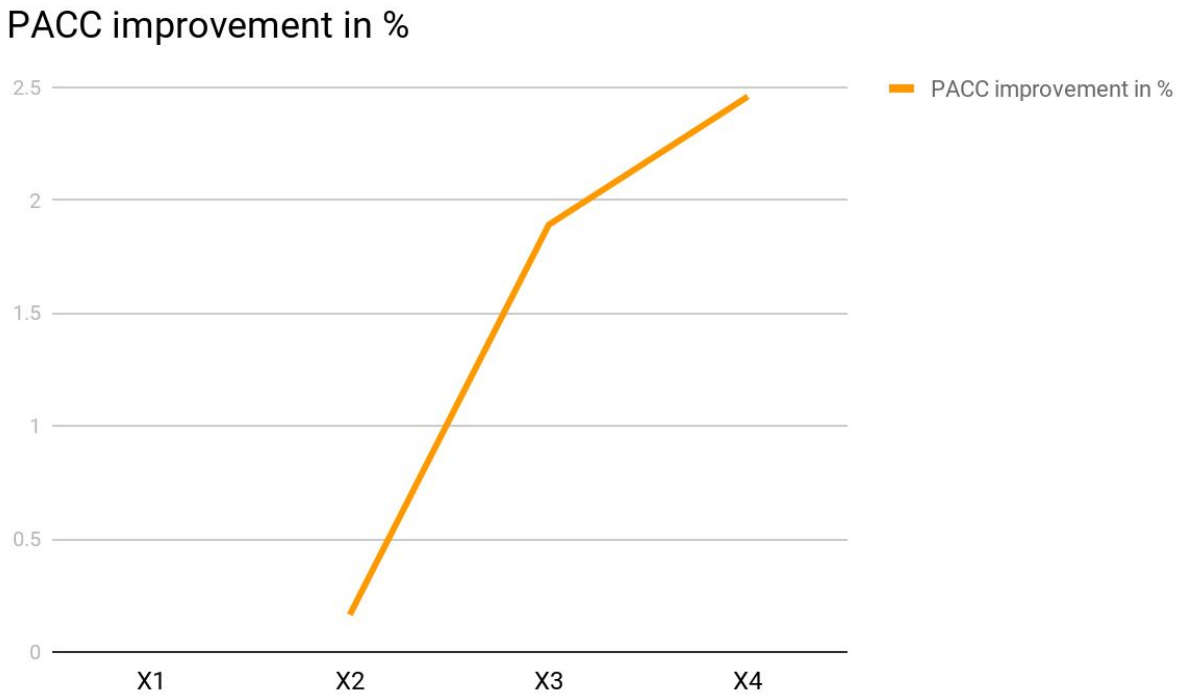
Graph 1: *PSNR - Scale factor results on DCSCN SR model*

PNSR - Scale factor is descending coordinated with the scale factor (Lower image resolution).

Semantic Segmentation Results:



Graph 2: *PACC - Scale Factor before and after applying super resolution to the dataset, results using the PSPNet in tensorflow.*



Graph 3: *PACC - Scale factor improvement in % of the super resolved images prior to SS results in improving the overall class pixel accuracy*

The test results indicates that Super Resolution aids the Semantic Segmentation accuracy. **Diagram 2** suggests the exact scenario.

18. Training Semantic Segmentation with Super Resolved images.

The results we received encouraged us to get even better PACC on the super resolved images. We wanted to train the semantic segmentation model giving it the ability to notice the differences between a super resolved image and a high resolution image. For that purpose we created a training and testing dataset built from 3000 super resolved images from the Google's CityScape set. After creating one scaled factor dataset (x4) with 3000 super resolved images we started training the Semantic Segmentation creating our own model.

The training phase had to be the same training as the original model. We used the same configuration as for 60000 steps each step had 2 images as an input and batch size of 2. The overall training duration was almost 10 days. After training was complete we started evaluating the results with the new model compared to the old model. Unfortunately after testing the new generated model we discovered that the results of the new model were slightly less successful than the results generated with

the original model. Therefore we can see the training on the HR images gets better results than the training on super resolved images. The PACC with the new generated model was around 82% when the HR model got 87.5%.

19. Super-Resolution with Segmentation Priors.

While working on the joint model we encountered a recently published paper implementing **Diagram 1.** [46] Their work approves the validity of the joint Semantic Segmentation and Super Resolution model, as we strive to increase the overall metric combination of both PSNR and PACC. As shown in **Figure. 5** the semantic priors assists with heavily detailed areas in the image.

Figure. 5. Comparing different SR approaches with downsampling factor ×4 focusing on a small, heavily detailed areas.

Using this paper combined with our proposed pipeline can lead to a new state-of-the-art model which can surpass the existing PSNR based super resolution models.

20. System Requirements / Libraries Required

- Python V3.6.
- Numpy, Scipy, Pillow, Scikit-image, OpenCV-Python.
- TensorFlow V1.12.
- Cityscape dataset with different scale factors (5000 images with ground truth ~11 GigaBytes of storage for each scale factor).
- Graphic card with more than 8GB of built in Memory.

21. Project Outputs

Preliminary Design Review (PDR) will include:

1) Selection of both Super Resolution and Semantic Segmentation state-of-the-art models.

2) Examination of different datasets choosing and preparing the data to fit both of models, creating different scale factor sets.

3) Overview of the model abilities testing results in PSNR and PACC.

Critical Design Review (CDR) will include:

1) Detailed design of the joint model, training methods and results.

2) Analytical analysis and simulation.

3) Experiments and reports.

Final Report – Project Book:

A report which will cover the project process in detail and will include all previous reports, final design and test reports.

22. Conclusions

We have explored the use of super resolution as an effective method for gaining better accuracy in the scene labeling task called semantic segmentation. We have discovered improvement in the scene labeling accuracy while using super resolution on low resolution images. We discovered that using super resolution on lower resolution images grants better accuracy improvement as we could see in the x2/x3/x4 PACC improvement graph at section 17. As the scale factor grows the PACC improvement rises. We can assume that it happens because Semantic segmentation is struggling labeling the pixeles where the edges are blurred, after using the super resolution we get a sharpened image assisting the algorithm with labeling the correct pixels. We believe that our discovery validates the approach of the suggested pipeline and we can declare the project was successful bringing new perspective to the image processing tasks.

23. Future suggestions

Our work combined with the recently published work on super resolution [46] using, semantic segmentation maps as categorical prior for constraining the plausible solution space in super resolution, might achieve state-of-the-art results in the near future as we suggested in Diagram 2. We believe this proposed method will be seen in the near future as super resolution and semantic segmentation papers are growing exponentially.

24. Acknowledgement

I would like to thank my moderator Dr. Alexander Apartsin, for his contribution in this project. He was available for any question 24/7, i learned from him alot about Machine/Deep Learning and Computer Vision research.

25. References

[1] R. Maini & H. Aggarwal A Comprehensive Review of Image Enhancement Techniques Journal of computing, volume 2, issue 3, March 2010.

[2] A.C. Berg, T. Berg, H. Daume III, J. Dodge, A. Goyal, X. Han, A. Mensch,M. Mitchell, A. Sood, K. Stratos, K. Yamaguchi, Understanding and Predicting Importance in Images http://tamaraberg.com/papers
[3] E. Rosten, T. Drummond, Machine Learning for High-Speed Corner Detection. Department of Engineering, Cambridge University, UK
[4] P. O. Pinheiro, R. Collobert, From Image-level to Pixel-level Labeling with Convolutional Networks IEEE Xplore
[5] S. Farsiu, M. Elad, P. Milanfar, A Practical Approach to Super-Resolution users.soe.ucsc.edu/~milanfar/publications/conf.
[6] D. Jyoti Bora, Importance of IMage Enhancement Techniques in color image segmentation. Department of Computer Science and Applications Barkatullah University, Bhopal, India.
[7] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence, 38(2):295–307, 2016.
[8] History of Retouching: Photographers and Retouchers Synergy in the Analog Photography Era.
https://retouchingacademy.com/history-of-retouching-photographers-and-retouchers-synergy-in-the-analog-photography-era/
[9] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung. O. Sorkine-Hornung, C. Schroers. A Fully Progressive Approach to Single-Image Super-Resolution.
[10] G. Freedman and R. Fattal. Image and video upscaling from local self-examples. ACM Transactions on Graphics, 30(2), 2011.
[11] W. S. Lai, J. B. Huang, N. Ahuja, M. H. Yang. Fast and Accurate Image Super-Resolution with Deep Laplacian Pyramid Networks.
[12] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, B. Catanzaro, NVIDIA Corporation, UC Berkeley. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs (CVPR) 2018.
[13] J.Long, E. Shelhamer, T. Darrell, UC Berkeley Fully Convolutional Networks for Semantic Segmentation.
[14] G. Carneiro, A. B. Chan, P. J. Moreno, N. Vasconcelos. Supervised Learning of Semantic Classes for Image Annotation and Retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 29 , Issue: 3 , March 2007).
[15] B. C. Russell, A. Torralba, K. P. Murphy, W. T. Freeman, LabelMe: A Database and Web-Based Tool for Image Annotation Springer Science+Business Media, LLC 2007.
[16] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.
[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In International Conference on Learning Representations, 2015.

[18] H. Hu, Z. Deng, G. Zhou. LabelBank: Revisiting Global Perspectives for Semantic Segmentation Mar 2017.
[19] W. Yang, X. Zhang, Y. Tian, W. Wang, J. Xue Deep Learning for Single Image Super-Resolution: A Brief Review in cornell University Library 2018.
[20] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, W. Shi, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.
[21] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, 13(4):600–612, 2004.
[22] List of reviewing papers Super Resolution created by LokLu. https://github.com/LokLu/Super_resolution_Survey
[23] A list of resources for Single Image Super-Resolution created by YapengTian. https://github.com/YapengTian/Single-Image-Super-Resolution
[24] J. Caballero, C. Ledig, A. Aitken, A. Acosta, and Totz. Realtime video super-resolution with spatio-temporal networks and motion compensation. In CVPR, 2017.
[25] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In ECCV, 2014
[26] S. J. Park, H. Son, S. Cho, K. Sang Hong, S. Lee, SRFeat: Single Image Super-Resolution with Feature Discrimination
[27] V. Badrinarayanan, A. Kendall, R. Cipolla, IEEE. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
[28] A list of all papers on Semantic Segmentation and the datasets they use. https://github.com/nightrome/really-awesome-semantic-segmentation
[29] Super resolution SRCNN model. https://github.com/LokLu/Super_resolution_Survey/blob/master/Reviews/SRCNN
[30] PyTorch implementation of semantic segmentation models on MIT ADE20K scene parsing dataset. https://github.com/CSAILVision/semantic-segmentation-pytorch
[31] Networks by architecture Semantic segmentation https://github.com/mrgloom/awesome-semantic-segmentation
[32] Guide to Semantic Segmentation with Deep Learning http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review
[34] D. Ciregan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification 16-21 June 2012 Providence, RI, USA
[35] Q. Huynh-Thu & M. Ghanbari The accuracy of PSNR in predicting video quality for different video scenes and frame rates Published 11 June 2010 © Springer Science+Business Media, LLC 2010
[36] Semantic segmentation based on deep learning. http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review
[37] TensorFlow website https://www.tensorflow.org

[38] Hellochick Semantic Segmentation Model.
https://github.com/hellochick/PSPNet-tensorflow.
[39] Super Resolution Model by J. Yamanaka, S. Kuwashima and T. Kurita.
https://github.com/jiny2001/dcscn-super-resolution.
[40] CityScape Dataset https://www.cityscapes-dataset.com/
[41] DIV2k Dataset https://data.vision.ee.ethz.ch/cvl/DIV2K/
[42] J. Yamanaka, S. Kuwashima, T. Kurita, Fast and Accurate Image Super Resolution by Deep CNN with Skip Connection and Network in Network *Submitted on 18 Jul 2017 (v1),* in Cornell University Library
[43] P. R. Rajarapollu, V. R. Mankar, Bicubic Interpolation Algorithm Implementation for Image Appearance Enhancement *IJCST Vol. 8, Issue 2, April - June 2017*
[44] Cityscape Class Definitions
https://www.cityscapes-dataset.com/dataset-overview/#class-definitions
[45] B. Dhivya, M. Sundaresan,"Performance analysis of interpolation methods for improving sub-image content-based retrieval," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, pp. 3909-3912, 2016.
[46] Xintao Wang, Ke Yu, Chao Dong, Chen Change Loy, Recovering Realistic Texture in Image Super-resolution by Deep Spatial Feature Transform

26. Poster

Student: Ran Mishael
Advisor: DR. Alexander Apartsin

Joint Super-Resolution and Semantic Segmentation.

The objective of this project is, to explore various neural network architectures for combining semantic segmentation and super-resolution. Ultimately, we'd like to construct a joint network that is trained for performing both tasks, as an augmentation network that produces enhancement and labeled image for a next high-level vision task

Potential Applications

Improving image quality and image segmentation might assist in a number of video analytic tasks including object detection, recognition, and tracking. Super Resolution could extend video analytic capabilities, to objects of smaller dimensions (e.g. extending operational distance for face recognition) or alternatively, could allow saving in camera sensors size or network bandwidth, due to the reduction in required image resolution

Block Diagram

Tested two possibilities regarding the construction of the system. In the 1st approach we will apply a Semantic Segmentation on the LR image. After we have the SS map we will apply SR on the LR image. This approach is illustrated by the diagram 1

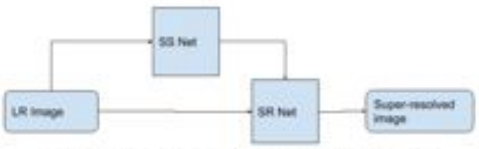


Diagram 1: *The serial combination of super-resolution and segmentation models*

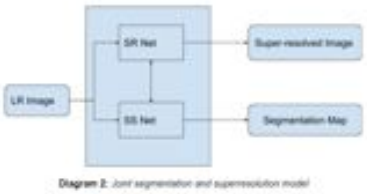


Diagram 2: *Joint segmentation and superresolution model*

Image. 2. *Super Resolution [39] Result on x4 scaled image. LR image on the left, SR output on the right.*

Image. 1. *Semantic Segmentation [38] results on x2 scaled image. The zoom-in segment of the image for other scale factors are displayed in the* **table 4** *below*

Electrical Engineering

AFEKA TEL · AVIV ACADEMIC COLLEGE OF ENGINEERING

Semantic Segmentation on Super-Resolved Images

We have evaluated pixel accuracy of the selected model on the original dataset and on downscaled versions of the same images. The resulting PACC evaluations serves as a baseline to our joint model testing if both SS and SR can benefit from sharing information. The pixel accuracy evaluated while testing the pretrained weights on unscaled images is 0.90235%. We wanted to see the different accuracies with different scale factors going from non scaled images, all the way to x4 scaled images. For that cause, 4 scale factor datasets were made. After testing the sets on unresolved images we used Super Resolution on each set of the scaled images, creating three super resolved datasets (x2,x3,x4) of the same set we used for semantic segmentation. We tested the super resolved sets with the same pre-trained weights and found that the super resolution had a positive effect on the semantic segmentation overall PACC performance.

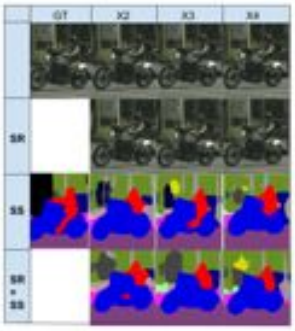
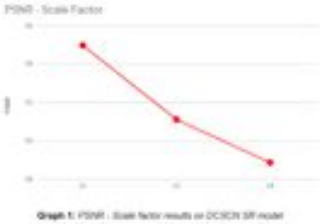


Table 4. SS and SR performed on Cityscape dataset with different scale factors

Image. 4. *This image is an example on how the super resolution helps the semantic segmentation to identify the legs of the bike rider. On the left, a x4 scaled image and on the right the x4 scaled, super resolved image.*

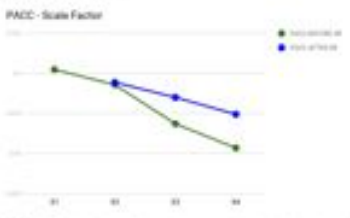
Super Resolution Results:



Graph 1: PSNR - Scale factor results on DCSCN SR model

PNSR - Scale factor is descending coordinated with the scale factor (Lower image resolution).

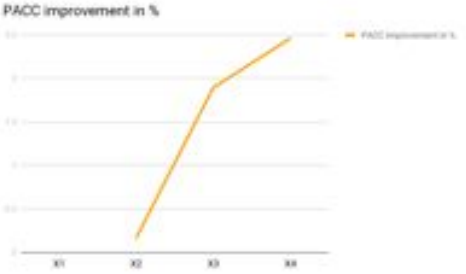
Semantic Segmentation Results:



Graph 2: PACC - Scale Factor before and after applying super resolution to the dataset, results using the ITSPNet in tensorflow

Data Set

The dataset chosen for the Semantic Segmentation training is the Cityscape Dataset. CityScape is a large-scale dataset that contains a diverse set of stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5 000 frames.



Graph 3: PACC - Scale factor improvement in % of the super resolved images prior to SS results in improving the overall class pixel accuracy

27. Code

Training SR:

```
import logging
import sys
import tensorflow as tf
import DCSCN
from helper import args, utilty as util
FLAGS = args.get()


def main(not_parsed_args):
    if len(not_parsed_args) > 1:
        print("Unknown args:%s" % not_parsed_args)
        exit()


    model = DCSCN.SuperResolution(FLAGS, model_name=FLAGS.model_name)


    if FLAGS.build_batch:
        model.load_datasets(FLAGS.data_dir + "/" + FLAGS.dataset, FLAGS.batch_dir + "/"
+ FLAGS.dataset,
                            FLAGS.batch_image_size, FLAGS.stride_size)
    else:
        model.load_dynamic_datasets(FLAGS.data_dir + "/" + FLAGS.dataset,
FLAGS.batch_image_size)
    model.build_graph()
    model.build_optimizer()
    model.build_summary_saver()


    logging.info("\n" + str(sys.argv))
    logging.info("Test Data:" + FLAGS.test_dataset + " Training Data:" + FLAGS.dataset)
    util.print_num_of_total_parameters(output_to_logging=True)
```

```
    total_psnr = total_ssim = 0

    for i in range(FLAGS.tests):
        psnr, ssim = train(model, FLAGS, i)
        total_psnr += psnr
        total_ssim += ssim

        logging.info("\nTrial(%d) %s" % (i, util.get_now_date()))
        model.print_steps_completed(output_to_logging=True)
        logging.info("PSNR:%f, SSIM:%f\n" % (psnr, ssim))

    if FLAGS.tests > 1:
        logging.info("\n=== Final Average [%s] PSNR:%f, SSIM:%f ===" % (
            FLAGS.test_dataset, total_psnr / FLAGS.tests, total_ssim / FLAGS.tests))

    model.copy_log_to_archive("archive")



def train(model, flags, trial):
    test_filenames = util.get_files_in_directory(flags.data_dir + "/" +
flags.test_dataset)
    if len(test_filenames) <= 0:
        print("Can't load images from [%s]" % (flags.data_dir + "/" +
flags.test_dataset))
        exit()

    model.init_all_variables()
    if flags.load_model_name != "":
        model.load_model(flags.load_model_name, output_log=True)

    model.init_train_step()
    model.init_epoch_index()
    model_updated = True
```

```
    psnr, ssim = model.evaluate(test_filenames)
    model.print_status(psnr, ssim, log=True)
    model.log_to_tensorboard(test_filenames[0], psnr, save_meta_data=True)

    while model.lr > flags.end_lr:

        model.build_input_batch()
        model.train_batch()

        if model.training_step * model.batch_num >= model.training_images:

            # one training epoch finished
            model.epochs_completed += 1
            psnr, ssim = model.evaluate(test_filenames)
            model.print_status(psnr, ssim, log=model_updated)
            model.log_to_tensorboard(test_filenames[0], psnr,
save_meta_data=model_updated)
            model.save_model(trial=trial, output_log=False)

            model_updated = model.update_epoch_and_lr()
            model.init_epoch_index()

    model.end_train_step()

    # save last generation anyway
    model.save_model(trial=trial, output_log=True)

    # outputs result
    evaluate_model(model, flags.test_dataset)

    if FLAGS.do_benchmark:
        for test_data in ['set5', 'set14', 'bsd100']:
            if test_data != flags.test_dataset:
```

```
                evaluate_model(model, test_data)

    return psnr, ssim



def evaluate_model(model, test_data):
    test_filenames = util.get_files_in_directory(FLAGS.data_dir + "/" + test_data)
    total_psnr = total_ssim = 0

    for filename in test_filenames:
        psnr, ssim = model.do_for_evaluate_with_output(filename,
output_directory=FLAGS.output_dir, print_console=False)
        total_psnr += psnr
        total_ssim += ssim

    logging.info("Model Average [%s] PSNR:%f, SSIM:%f" % (
        test_data, total_psnr / len(test_filenames), total_ssim / len(test_filenames)))



if __name__ == '__main__':
    tf.app.run()
```

Evaluate SR:

```
import logging
import tensorflow as tf
import DCSCN
from helper import args, utilty as util

args.flags.DEFINE_boolean("save_results", True, "Save result, bicubic and loss
images.")
args.flags.DEFINE_boolean("compute_bicubic", False, "Compute bicubic performance.")

FLAGS = args.get()



def main(not_parsed_args):
   if len(not_parsed_args) > 1:
       print("Unknown args:%s" % not_parsed_args)
       exit()

   model = DCSCN.SuperResolution(FLAGS, model_name=FLAGS.model_name)
   model.build_graph()
   model.build_summary_saver()
   model.init_all_variables()

   if FLAGS.test_dataset == "all":
       test_list = ['bochum', 'bremen', 'cologne', 'darmstadt', 'dusseldorf',
'erfurt', 'hamburg', 'hanover', 'jena', 'krefeld', 'monchengladbach', 'strasbourg',
'stuttgart', 'tubingen', 'ulm', 'weimar', 'zurich']
   else:
       test_list = [FLAGS.test_dataset]

   for i in range(FLAGS.tests):
       model.load_model(FLAGS.load_model_name, trial=i, output_log=True if FLAGS.tests
> 1 else False)
```

```
        if FLAGS.compute_bicubic:
            for test_data in test_list:
                evaluate_bicubic(model, test_data)

        for test_data in test_list:
            evaluate_model(model, test_data)


def evaluate_bicubic(model, test_data):
    test_filenames = util.get_files_in_directory(FLAGS.data_dir + "/" + test_data)
    total_psnr = total_ssim = 0

    for filename in test_filenames:
        psnr, ssim = model.evaluate_bicubic(filename, print_console=False)
        total_psnr += psnr
        total_ssim += ssim

    logging.info("Bicubic Average [%s] PSNR:%f, SSIM:%f" % (
        test_data, total_psnr / len(test_filenames), total_ssim / len(test_filenames)))


def evaluate_model(model, test_data):
    test_filenames = util.get_files_in_directory(FLAGS.data_dir + "/" + test_data)
    total_psnr = total_ssim = 0

    for filename in test_filenames:
        if FLAGS.save_results:
            psnr, ssim = model.do_for_evaluate_with_output(filename,
output_directory=FLAGS.output_dir,
                                                           print_console=False)
        else:
            psnr, ssim = model.do_for_evaluate(filename, print_console=False)
```

```
        total_psnr += psnr
        total_ssim += ssim

    logging.info("Model Average [%s] PSNR:%f, SSIM:%f" % (
        test_data, total_psnr / len(test_filenames), total_ssim / len(test_filenames)))

if __name__ == '__main__':
    tf.app.run()
```

Train SS:

```
from __future__ import print_function
import argparse
import os
import sys
import time
import tensorflow as tf
import numpy as np
from model import PSPNet101
from tools import prepare_label
from image_reader import ImageReader

IMG_MEAN = np.array((103.939, 116.779, 123.68), dtype=np.float32)

BATCH_SIZE = 2
DATA_DIRECTORY = './data/cityscapes_dataset/cityscape/'
DATA_LIST_PATH = './list/cityscapes_train_list.txt'
IGNORE_LABEL = 255
INPUT_SIZE = '713,713'
LEARNING_RATE = 1e-3
MOMENTUM = 0.9
NUM_CLASSES = 19
NUM_STEPS = 60001
```

```
POWER = 0.9
RANDOM_SEED = 1234
WEIGHT_DECAY = 0.0001
RESTORE_FROM = './'
SNAPSHOT_DIR = './model/'
SAVE_NUM_IMAGES = 4
SAVE_PRED_EVERY = 20



def get_arguments():
   parser = argparse.ArgumentParser(description="DeepLab-ResNet Network")
   parser.add_argument("--batch-size", type=int, default=BATCH_SIZE,
                       help="Number of images sent to the network in one step.")
   parser.add_argument("--data-dir", type=str, default=DATA_DIRECTORY,
                       help="Path to the directory containing the PASCAL VOC
dataset.")
   parser.add_argument("--data-list", type=str, default=DATA_LIST_PATH,
                       help="Path to the file listing the images in the dataset.")
   parser.add_argument("--ignore-label", type=int, default=IGNORE_LABEL,
                       help="The index of the label to ignore during the training.")
   parser.add_argument("--input-size", type=str, default=INPUT_SIZE,
                       help="Comma-separated string with height and width of images.")
   parser.add_argument("--is-training", action="store_true",
                       help="Whether to updates the running means and variances during
the training.")
   parser.add_argument("--learning-rate", type=float, default=LEARNING_RATE,
                       help="Base learning rate for training with polynomial decay.")
   parser.add_argument("--momentum", type=float, default=MOMENTUM,
                       help="Momentum component of the optimiser.")
   parser.add_argument("--not-restore-last", action="store_true",
                       help="Whether to not restore last (FC) layers.")
   parser.add_argument("--num-classes", type=int, default=NUM_CLASSES,
                       help="Number of classes to predict (including background).")
   parser.add_argument("--num-steps", type=int, default=NUM_STEPS,
```

```
                        help="Number of training steps.")
    parser.add_argument("--power", type=float, default=POWER,
                        help="Decay parameter to compute the learning rate.")
    parser.add_argument("--random-mirror", action="store_true",
                        help="Whether to randomly mirror the inputs during the
training.")
    parser.add_argument("--random-scale", action="store_true",
                        help="Whether to randomly scale the inputs during the
training.")
    parser.add_argument("--random-seed", type=int, default=RANDOM_SEED,
                        help="Random seed to have reproducible results.")
    parser.add_argument("--restore-from", type=str, default=RESTORE_FROM,
                        help="Where restore model parameters from.")
    parser.add_argument("--save-num-images", type=int, default=SAVE_NUM_IMAGES,
                        help="How many images to save.")
    parser.add_argument("--save-pred-every", type=int, default=SAVE_PRED_EVERY,
                        help="Save summaries and checkpoint every often.")
    parser.add_argument("--snapshot-dir", type=str, default=SNAPSHOT_DIR,
                        help="Where to save snapshots of the model.")
    parser.add_argument("--weight-decay", type=float, default=WEIGHT_DECAY,
                        help="Regularisation parameter for L2-loss.")
    parser.add_argument("--update-mean-var", action="store_true",
                        help="whether to get update_op from tf.Graphic_Keys")
    parser.add_argument("--train-beta-gamma", action="store_true",
                        help="whether to train beta & gamma in bn layer")
    return parser.parse_args()


def save(saver, sess, logdir, step):
  model_name = 'model.ckpt'
  checkpoint_path = os.path.join(logdir, model_name)


  if not os.path.exists(logdir):
     os.makedirs(logdir)
  saver.save(sess, checkpoint_path, global_step=step)
```

```
  print('The checkpoint has been created.')


def load(saver, sess, ckpt_path):
    saver.restore(sess, ckpt_path)
    print("Restored model parameters from {}".format(ckpt_path))


def main():
    """Create the model and start the training."""
    args = get_arguments()

    h, w = map(int, args.input_size.split(','))
    input_size = (h, w)

    tf.set_random_seed(args.random_seed)

    coord = tf.train.Coordinator()

    with tf.name_scope("create_inputs"):
        reader = ImageReader(
            args.data_dir,
            args.data_list,
            input_size,
            args.random_scale,
            args.random_mirror,
            args.ignore_label,
            IMG_MEAN,
            coord)
        image_batch, label_batch = reader.dequeue(args.batch_size)

    net = PSPNet101({'data': image_batch}, is_training=True,
num_classes=args.num_classes)

    raw_output = net.layers['conv6']
```

```
    # According from the prototxt in Caffe implement, learning rate must multiply by
10.0 in pyramid module
    fc_list = ['conv5_3_pool1_conv', 'conv5_3_pool2_conv', 'conv5_3_pool3_conv',
'conv5_3_pool6_conv', 'conv6', 'conv5_4']
    restore_var = [v for v in tf.global_variables()]
    all_trainable = [v for v in tf.trainable_variables() if ('beta' not in v.name and
'gamma' not in v.name) or args.train_beta_gamma]
    fc_trainable = [v for v in all_trainable if v.name.split('/')[0] in fc_list]
    conv_trainable = [v for v in all_trainable if v.name.split('/')[0] not in fc_list]
# lr * 1.0
    fc_w_trainable = [v for v in fc_trainable if 'weights' in v.name] # lr * 10.0
    fc_b_trainable = [v for v in fc_trainable if 'biases' in v.name] # lr * 20.0
    assert(len(all_trainable) == len(fc_trainable) + len(conv_trainable))
    assert(len(fc_trainable) == len(fc_w_trainable) + len(fc_b_trainable))


    # Predictions: ignoring all predictions with labels greater or equal than n_classes
    raw_prediction = tf.reshape(raw_output, [-1, args.num_classes])
    label_proc = prepare_label(label_batch, tf.stack(raw_output.get_shape()[1:3]),
num_classes=args.num_classes, one_hot=False) # [batch_size, h, w]
    raw_gt = tf.reshape(label_proc, [-1,])
    indices = tf.squeeze(tf.where(tf.less_equal(raw_gt, args.num_classes - 1)), 1)
    gt = tf.cast(tf.gather(raw_gt, indices), tf.int32)
    prediction = tf.gather(raw_prediction, indices)


    # Pixel-wise softmax loss.
    loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=prediction, labels=gt)
    l2_losses = [args.weight_decay * tf.nn.l2_loss(v) for v in tf.trainable_variables()
if 'weights' in v.name]
    reduced_loss = tf.reduce_mean(loss) + tf.add_n(l2_losses)


    # Using Poly learning rate policy
    base_lr = tf.constant(args.learning_rate)
    step_ph = tf.placeholder(dtype=tf.float32, shape=())
```

```
    learning_rate = tf.scalar_mul(base_lr, tf.pow((1 - step_ph / args.num_steps),
args.power))


    # Gets moving_mean and moving_variance update operations from
tf.GraphKeys.UPDATE_OPS
    if args.update_mean_var == False:
        update_ops = None
    else:
        update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)


    with tf.control_dependencies(update_ops):
        opt_conv = tf.train.MomentumOptimizer(learning_rate, args.momentum)
        opt_fc_w = tf.train.MomentumOptimizer(learning_rate * 10.0, args.momentum)
        opt_fc_b = tf.train.MomentumOptimizer(learning_rate * 20.0, args.momentum)


        grads = tf.gradients(reduced_loss, conv_trainable + fc_w_trainable +
fc_b_trainable)
        grads_conv = grads[:len(conv_trainable)]
        grads_fc_w = grads[len(conv_trainable) : (len(conv_trainable) +
len(fc_w_trainable))]
        grads_fc_b = grads[(len(conv_trainable) + len(fc_w_trainable)):]


        train_op_conv = opt_conv.apply_gradients(zip(grads_conv, conv_trainable))
        train_op_fc_w = opt_fc_w.apply_gradients(zip(grads_fc_w, fc_w_trainable))
        train_op_fc_b = opt_fc_b.apply_gradients(zip(grads_fc_b, fc_b_trainable))


        train_op = tf.group(train_op_conv, train_op_fc_w, train_op_fc_b)


    # Set up tf session and initialize variables.
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    sess = tf.Session(config=config)
    init = tf.global_variables_initializer()
```

```
    sess.run(init)

    # Saver for storing checkpoints of the model.
    saver = tf.train.Saver(var_list=tf.global_variables(), max_to_keep=10)

    ckpt = tf.train.get_checkpoint_state(SNAPSHOT_DIR)
    if ckpt and ckpt.model_checkpoint_path:
        loader = tf.train.Saver(var_list=restore_var)
        load_step = int(os.path.basename(ckpt.model_checkpoint_path).split('-')[1])
        load(loader, sess, ckpt.model_checkpoint_path)
    else:
        print('No checkpoint file found.')
        load_step = 0

    # Start queue threads.
    threads = tf.train.start_queue_runners(coord=coord, sess=sess)

    # Iterate over training steps.
    for step in range(args.num_steps):
        start_time = time.time()

        feed_dict = {step_ph: step}
        if step % args.save_pred_every == 0:
            loss_value, _ = sess.run([reduced_loss, train_op], feed_dict=feed_dict)
            save(saver, sess, args.snapshot_dir, step)
        else:
            loss_value, _ = sess.run([reduced_loss, train_op], feed_dict=feed_dict)
        duration = time.time() - start_time
        print('step {:d} \t loss = {:.3f}, ({:.3f} sec/step)'.format(step, loss_value,
duration))

    coord.request_stop()
    coord.join(threads)
```

```
if __name__ == '__main__':
   main()
```

Evaluate SS:

```
from __future__ import print_function
import argparse
import os
import sys
import time
from PIL import Image
import tensorflow as tf
import numpy as np
from tqdm import trange
from model import PSPNet101, PSPNet50
from tools import *

SNAPSHOT_DIR = './model/pspnet101-cityscapes'

ADE20k_param = {'crop_size': [473, 473],
                'num_classes': 150, # predict: [0~149] corresponding to label [1~150],
ignore class 0 (background)
                'ignore_label': 0,
                'num_steps': 2000,
                'model': PSPNet50,
                'data_dir': '../ADEChallengeData2016/', #### Change this line
                'val_list': './list/ade20k_val_list.txt'}

cityscapes_param = {'crop_size': [720, 720],
                    'num_classes': 19,
```

```
                    'ignore_label': 255,
                    'num_steps': 500,
                    'model': PSPNet101,
                    'data_dir': './data/cityscapes_dataset/cityscape/', #### Change
this line
                    'val_list': './list/cityscapes_val_list.txt'}

def get_arguments():
    parser = argparse.ArgumentParser(description="Reproduced PSPNet")

    parser.add_argument("--checkpoints", type=str, default=SNAPSHOT_DIR,
                        help="Path to restore weights.")
    parser.add_argument("--flipped-eval", action="store_true",
                        help="whether to evaluate with flipped img.")
    parser.add_argument("--dataset", type=str, default='cityscapes',
                        choices=['ade20k', 'cityscapes'],
                        required=False)

    return parser.parse_args()

def load(saver, sess, ckpt_path):
    saver.restore(sess, ckpt_path)
    print("Restored model parameters from {}".format(ckpt_path))

def main():
    args = get_arguments()

    # load parameters
    if args.dataset == 'ade20k':
        param = ADE20k_param
    elif args.dataset == 'cityscapes':
        param = cityscapes_param
```

```
    crop_size = param['crop_size']
    num_classes = param['num_classes']
    ignore_label = param['ignore_label']
    num_steps = param['num_steps']
    PSPNet = param['model']
    data_dir = param['data_dir']

    # Set placeholder
    image_filename = tf.placeholder(dtype=tf.string)
    anno_filename = tf.placeholder(dtype=tf.string)

    # Read & Decode image
    img = tf.image.decode_image(tf.read_file(image_filename), channels=3)
    anno = tf.image.decode_image(tf.read_file(anno_filename), channels=1)
    img.set_shape([None, None, 3])
    anno.set_shape([None, None, 1])

    shape = tf.shape(img)
    h, w = (tf.maximum(crop_size[0], shape[0]), tf.maximum(crop_size[1], shape[1]))
    img = preprocess(img, h, w)

     # Create network.
    net = PSPNet({'data': img}, is_training=False, num_classes=num_classes)
    with tf.variable_scope('', reuse=True):
        flipped_img = tf.image.flip_left_right(tf.squeeze(img))
        flipped_img = tf.expand_dims(flipped_img, dim=0)
        net2 = PSPNet({'data': flipped_img}, is_training=False,
num_classes=num_classes)

    raw_output = net.layers['conv6']

    # Do flipped eval or not
    if args.flipped_eval:
```

```
        flipped_output = tf.image.flip_left_right(tf.squeeze(net2.layers['conv6']))
        flipped_output = tf.expand_dims(flipped_output, dim=0)
        raw_output = tf.add_n([raw_output, flipped_output])


    # Scale feature map to image size, get prediction
    raw_output_up = tf.image.resize_bilinear(raw_output, size=[h, w],
align_corners=True)
    raw_output_up = tf.image.crop_to_bounding_box(raw_output_up, 0, 0, shape[0],
shape[1])
    raw_output_up = tf.argmax(raw_output_up, dimension=3)
    pred = tf.expand_dims(raw_output_up, dim=3)


    # Calculate mIoU
    pred_flatten = tf.reshape(pred, [-1,])
    raw_gt = tf.reshape(anno, [-1,])
    indices = tf.squeeze(tf.where(tf.not_equal(raw_gt, ignore_label)), 1)
    gt = tf.cast(tf.gather(raw_gt, indices), tf.int64)
    pred = tf.gather(pred_flatten, indices)


    if args.dataset == 'ade20k':
        pred = tf.add(pred, tf.constant(1, dtype=tf.int64))
        mIoU, update_op = tf.contrib.metrics.streaming_mean_iou(pred, gt,
num_classes=num_classes+1)
    elif args.dataset == 'cityscapes':
        #mIoU, update_op = tf.contrib.metrics.streaming_mean_iou(pred, gt,
num_classes=num_classes)
        #mIoU, update_op = tf.contrib.metrics.streaming_accuracy(pred, gt)
        mIoU, update_op =tf.metrics.mean_per_class_accuracy(pred, gt,
num_classes=num_classes)
    # Set up tf session and initialize variables.
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    sess = tf.Session(config=config)


    global_init = tf.global_variables_initializer()
```

```
    local_init = tf.local_variables_initializer()
    sess.run(global_init)
    sess.run(local_init)

    restore_var = tf.global_variables()

    ckpt = tf.train.get_checkpoint_state(args.checkpoints)
    if ckpt and ckpt.model_checkpoint_path:
        loader = tf.train.Saver(var_list=restore_var)
        load_step = int(os.path.basename(ckpt.model_checkpoint_path).split('-')[1])
        load(loader, sess, ckpt.model_checkpoint_path)
    else:
        print('No checkpoint file found.')


    file = open(param['val_list'], 'r')
    #num_steps = 5
    for step in trange(num_steps, desc='evaluation', leave=True):
        f1, f2 = file.readline().split('\n')[0].split(' ')
        f1 = os.path.join(data_dir, f1)
        f2 = os.path.join(data_dir, f2)

        _ = sess.run(update_op, feed_dict={image_filename: f1, anno_filename: f2})

    print('PACC: {:04f}'.format(sess.run(mIoU)))
    #print('Pixel accuracy is:'.format(sess.run(Pacc)))

if __name__ == '__main__':
    main()
```

Use SS on single image:

```
from __future__ import print_function

import argparse

import os

import sys

import time

import tensorflow as tf

import numpy as np

from scipy import misc

from model import PSPNet101, PSPNet50

from tools import *


ADE20k_param = {'crop_size': [473, 473],

                'num_classes': 150,

                'model': PSPNet50}

cityscapes_param = {'crop_size': [720, 720],

                    'num_classes': 19,

                    'model': PSPNet101}


SAVE_DIR = './output/'

SNAPSHOT_DIR = './model/pspnet101-cityscapes'


def get_arguments(current_file):

   parser = argparse.ArgumentParser(description="Reproduced PSPNet")

   parser.add_argument("--img-path", type=str, default='./input/'+current_file,

                       help="Path to the RGB image file.")

   parser.add_argument("--checkpoints", type=str, default=SNAPSHOT_DIR,

                       help="Path to restore weights.")

   parser.add_argument("--save-dir", type=str, default=SAVE_DIR,

                       help="Path to save output.")

   parser.add_argument("--flipped-eval", action="store_true",

                       help="whether to evaluate with flipped img.")
```

```
    parser.add_argument("--dataset", type=str, default='cityscapes',
                        choices=['ade20k', 'cityscapes'],
                        required=False)

    return parser.parse_args()

def save(saver, sess, logdir, step):
  model_name = 'model.ckpt'
  checkpoint_path = os.path.join(logdir, model_name)

  if not os.path.exists(logdir):
     os.makedirs(logdir)
  saver.save(sess, checkpoint_path, global_step=step)
  print('The checkpoint has been created.')

def load(saver, sess, ckpt_path):
    saver.restore(sess, ckpt_path)
    print("Restored model parameters from {}".format(ckpt_path))


def main(current_file):

    args = get_arguments(current_file)
    # load parameters
    if args.dataset == 'ade20k':
        param = ADE20k_param
    elif args.dataset == 'cityscapes':
        param = cityscapes_param

    crop_size = param['crop_size']
    num_classes = param['num_classes']
    PSPNet = param['model']
```

```
    # preprocess images
    img, filename = load_img(args.img_path)
    img_shape = tf.shape(img)
    h, w = (tf.maximum(crop_size[0], img_shape[0]), tf.maximum(crop_size[1],
img_shape[1]))
    img = preprocess(img, h, w)


    #if current_file == "leftImg8bit/val/lindau/lindau_000000_000019_leftImg8bit.png":
    with tf.variable_scope('', reuse=tf.AUTO_REUSE):
        net = PSPNet({'data': img}, is_training=False, num_classes=num_classes)
    with tf.variable_scope('', reuse=True):
        flipped_img = tf.image.flip_left_right(tf.squeeze(img))
        flipped_img = tf.expand_dims(flipped_img, dim=0)
        net2 = PSPNet({'data': flipped_img}, is_training=False,
num_classes=num_classes)
    raw_output = net.layers['conv6']


    # Do flipped eval or not
    if args.flipped_eval:
        flipped_output = tf.image.flip_left_right(tf.squeeze(net2.layers['conv6']))
        flipped_output = tf.expand_dims(flipped_output, dim=0)
        raw_output = tf.add_n([raw_output, flipped_output])


    # Predictions.
    raw_output_up = tf.image.resize_bilinear(raw_output, size=[h, w],
align_corners=True)
    raw_output_up = tf.image.crop_to_bounding_box(raw_output_up, 0, 0, img_shape[0],
img_shape[1])
    raw_output_up = tf.argmax(raw_output_up, axis=3)
    pred = decode_labels(raw_output_up, img_shape, num_classes)


    # Init tf Session
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
```

```
    sess = tf.Session(config=config)
    init = tf.global_variables_initializer()

    sess.run(init)

    restore_var = tf.global_variables()

    ckpt = tf.train.get_checkpoint_state(args.checkpoints)
    if ckpt and ckpt.model_checkpoint_path:
        loader = tf.train.Saver(var_list=restore_var)
        load_step = int(os.path.basename(ckpt.model_checkpoint_path).split('-')[1])
        load(loader, sess, ckpt.model_checkpoint_path)
    else:
        print('No checkpoint file found.')

    preds = sess.run(pred)

    if not os.path.exists(args.save_dir):
        os.makedirs(args.save_dir)
    misc.imsave(args.save_dir + filename, preds[0])


if __name__ == '__main__':
    x = 0
    file = open('./list/cityscapes_val_list.txt', 'r')
    for index in range(22):
        f1, f2 = file.readline().split('\n')[0].split(' ')
        current_file = f1
        main(current_file)
```

Creating the Dataset:

```
import configparser
import logging
import os
import random
import numpy as np
from scipy import misc
import utilty as util
from PIL import Image
from resizeimage import resizeimage


INPUT_IMAGE_DIR = "input"
INTERPOLATED_IMAGE_DIR = "interpolated"
TRUE_IMAGE_DIR = "output"


def make_directory(directory):
    if not os.path.exists(directory):
        os.makedirs(directory)


def resize_func(scale):
    file_path = "imageset/"
    scale_string = str(scale)
    make_directory(INPUT_IMAGE_DIR)
    make_directory(TRUE_IMAGE_DIR)
    if not os.path.exists(file_path):
        print(filepath + "wasnt found breaking script")
        exit()
    fp = open(file_path + "cityscapes_val_list.txt", 'r')
    for line in range(500):
        line_num = fp.readline()
        print(line)
        print("reading image from path " + file_path + line_num)
```

```
        line_num = line_num.replace(" \n", "")
        line_num = line_num.replace(".png\n", ".png")
        im = Image.open(file_path+line_num)
        width, height = im.size
        width_new = width / scale
        height_new = height / scale

        cover = im.resize((int(width_new), int(height_new)), Image.BICUBIC)
        new_image = cover.resize((int(width), int(height)), Image.BICUBIC)

        #cover = resizeimage.resize_cover(im, [width_new, height_new],
validate=False,resample=im.BICUBIC)
        #new_image = resizeimage.resize_cover(cover, [width, height],
validate=False,resample=cover.BICUBIC)
        directoryx = line_num.rsplit('/', 1)[0]
        new = TRUE_IMAGE_DIR+"/"+scale_string+directoryx
        print(new)
        make_directory(new)
        new_image.save(TRUE_IMAGE_DIR+"/"+scale_string+line_num)



print("Resizing Image set to path /imageset/val/")
scale = 2
resize_func(scale)
scale = 3
resize_func(scale)
scale = 4
resize_func(scale)
print("test finished")
```