Carlos Marzal Romon A08922106

# DLCV Homework 2

## Problem 1: Segmentation

### 1. Baseline Model

**1.1. Describe how you pre-process the data:**

The only change to the data is normalizing it using the values for MEAN and STD that are suggested by torch for the normalization of the images to be used in image segmentation:

MEAN=[0.485, 0.456, 0.406]

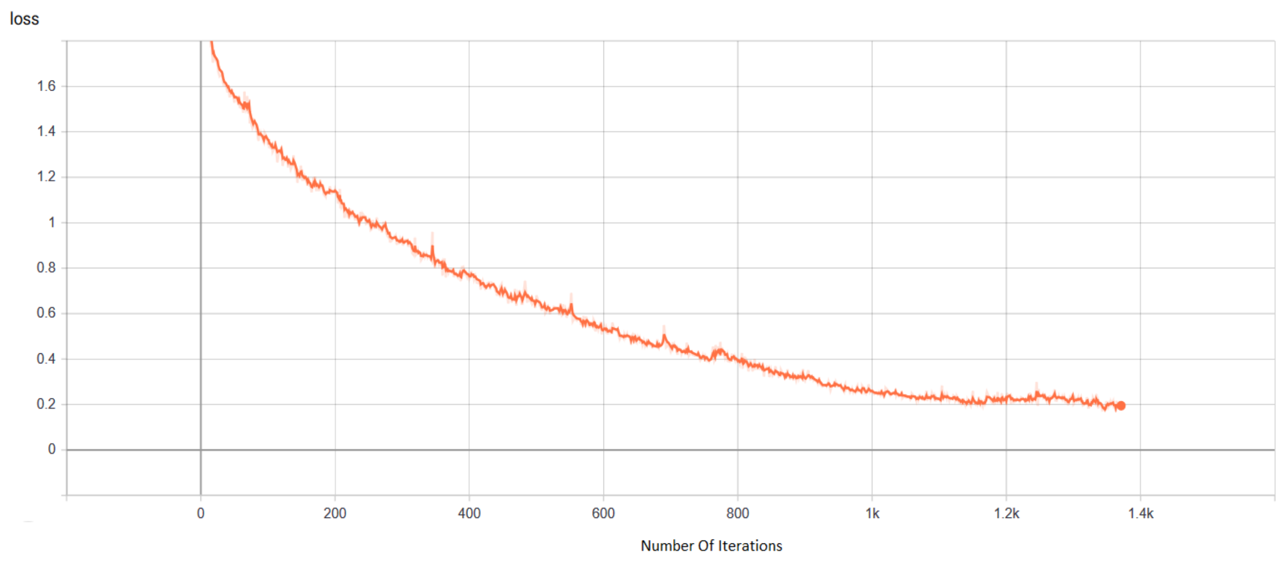
STD=[0.229, 0.224, 0.225]

These values mostly help reduce the training epochs needed to reach convergence.

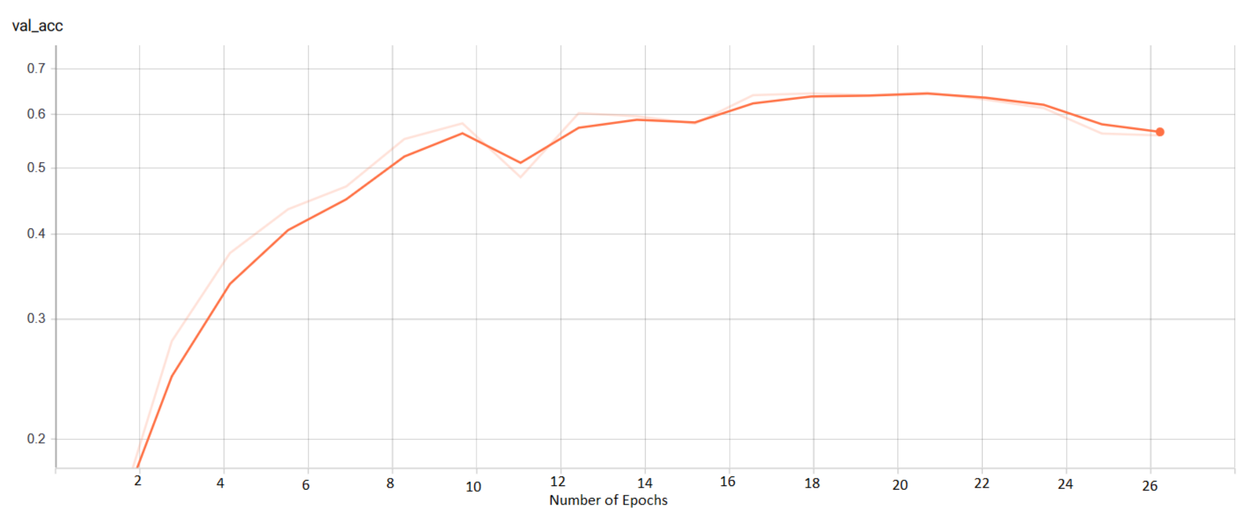
As for data augmentation, I tried implementing it by having the original images and the same image flipped horizontally, but overall, we already have a decently big sample size and the data augmentation ended up greatly increasing the training time for a not visible increase in the mIoU.

**1.2. Figures**

**1.2.1. Training loss versus number of training iterations:**

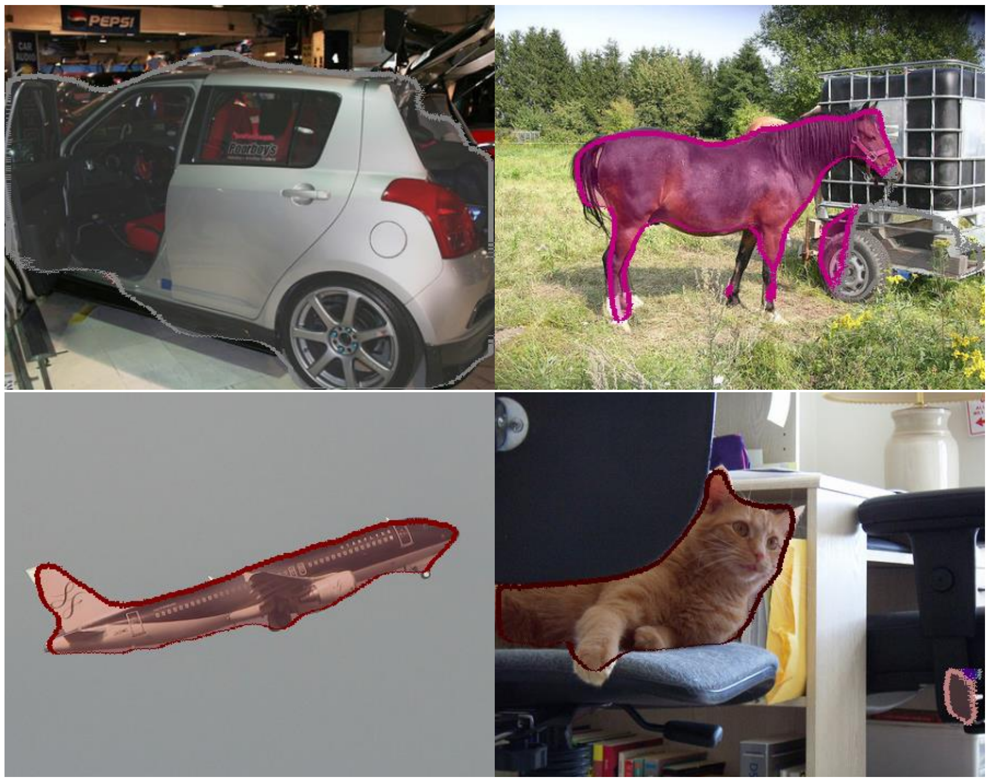


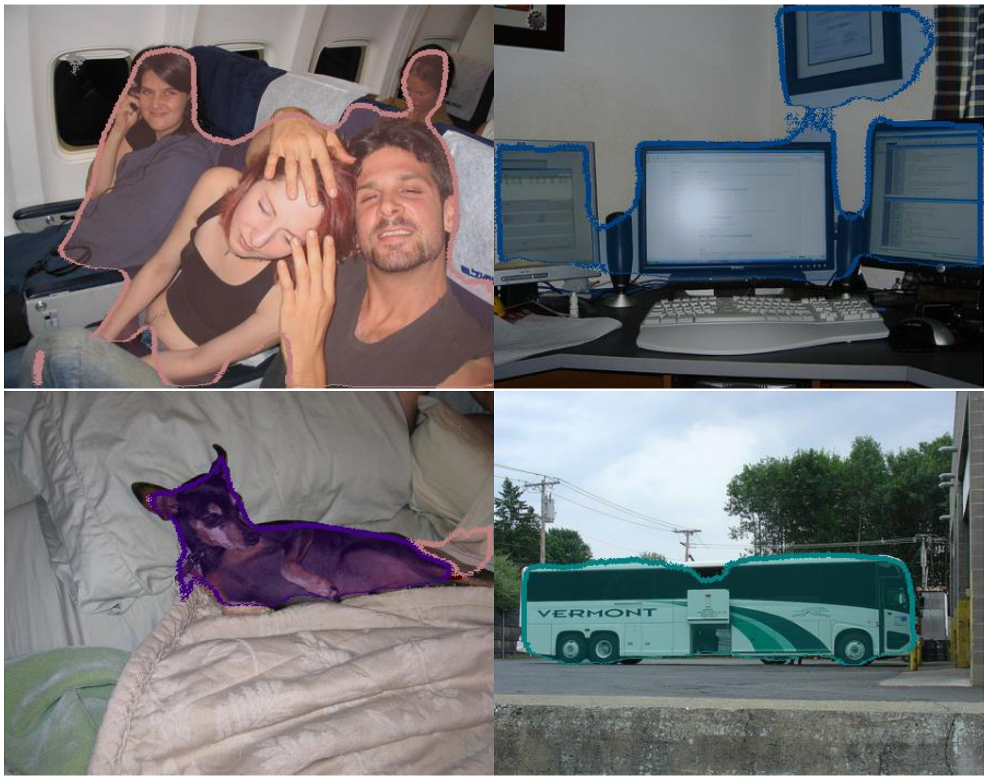
**1.2.2 IoU score on validation set versus number of training iterations:**



Comparing both images, you can clearly see there is a direct relationship between these two graphs, showing how when one of the peaked the other one did two (obviously inversely related).

**1.3. Visualize at least one semantic segmentation result for each class:**





These 8 images show the segmentation obtained in the respective images. The biggest segmented part in each image according to its color are the following (from left to right, top down):

Car, horse, aeroplane, cat, person, tv/monitor, dog, bus.

As you can observe, images are correctly segmented except for some small zones where it's obtaining things that are wrong.

### 1.4. mIoU score

The mean IoU score obtained in the validation set of images with the best model is **0.651434**, which is higher than the simple baseline required in the problem.

This IoU is also obtained per class (there are 9 different classes), the results are the following:

class #0 : 0.89999 (Background)

class #1 : 0.74100 (Person)

class #2 : 0.67189 (Aeroplane)

class #3 : 0.68022 (Bus)

class #4 : 0.37065 (tv/monitor)

class #5 : 0.52418 (Horse)

class #6 : 0.60950 (Dog)

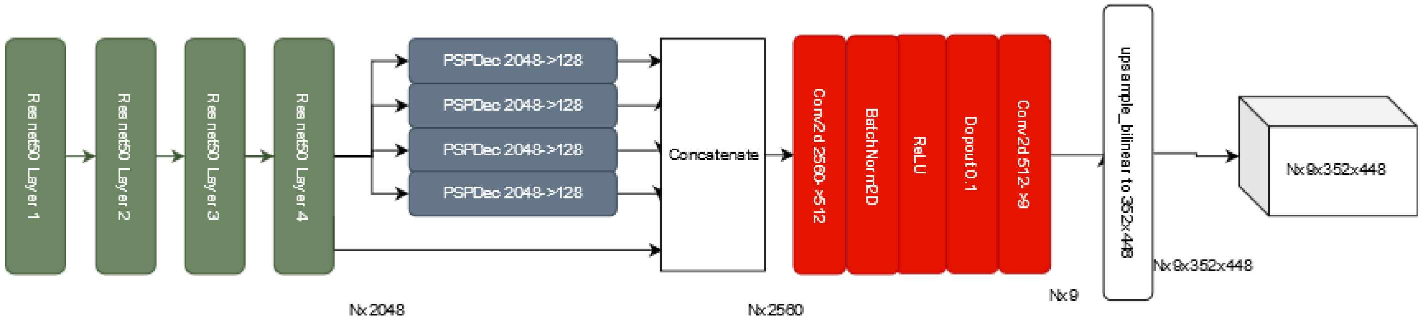
class #7 : 0.71981 (Cat)

class #8 : 0.64566 (Car)

The class with the lowest IoU is the class #4 which corresponds to the tv/monitor. This could be because it's a really simple object which can be easily confused with other objects, like painting or windows. Also, if the monitor screen in on, the inside of the monitor can contain different objects and things, so its hard to train knowing all the options that it can have.

## 2. Improved Model

### 2.1. Draw the model architecture of your improved model



### 2.2. Why does the improved model work better than the baseline one?

So, first of all this model is based in using Resnet50, which is a heavier version of Resnet than the Resnet18 used in the baseline model.

Also, on the architecture we can see how this model used a Pyramid Scene Parsing Network (PSP), which basically uses maxpoling of different sizes to obtain more information on the group of pixels rather than only and individual pixel. This information is then concatenated with the output of the resnet into a 2560-dimension array and resized using convolutions and upsamples until out desired output.

Both the different Resnet and the PSP makes this model slower to run overall compared to the original one, but it trains after much less epochs and at a higher limit.

Just to do some quick comparisons, this model took 12 epochs to reach the desired MiOu in the validation set, while the initial one needed 20 (although it took around 10 less minutes to run.

Also, using this same code with Resnet18 instead of 50 (and making the appropriate changes in the dimensioning of the code) we got a really fast training, but it capped at 66% after 8 epochs. This shows how Resnet50 works better without set dimensions.

### 2.3. Prove of improved model

The best MiOu obtained in the improved model after was **0.705206** (which is slightly above the minimum required in the problem)

And per class:

class #0 : 0.90635

class #1 : 0.73926

class #2 : 0.70348

class #3 : 0.76546

class #4 : 0.44183

class #5 : 0.62275

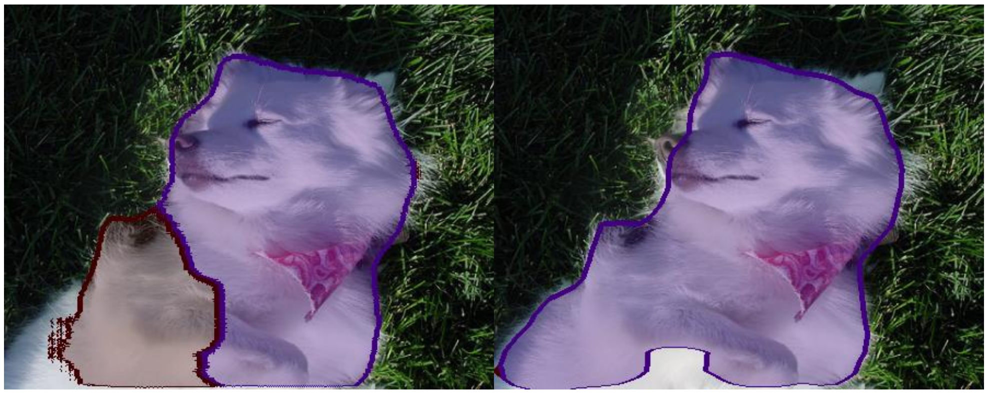
class #6 : 0.63557

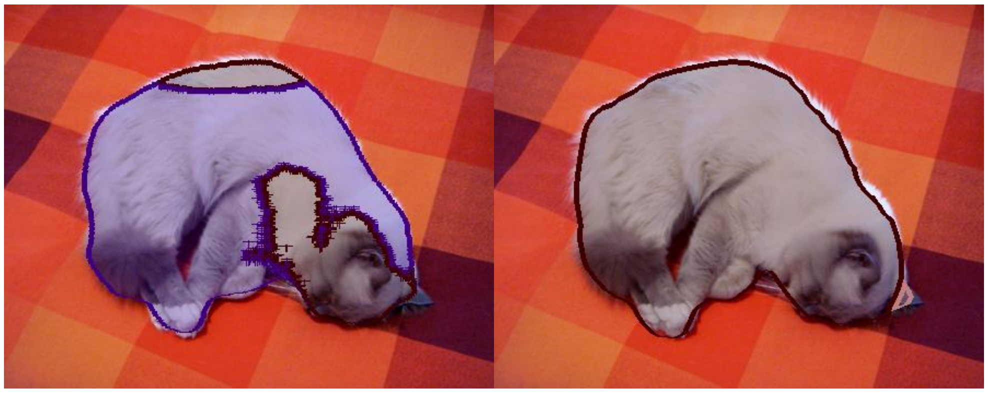
class #7 : 0.75403

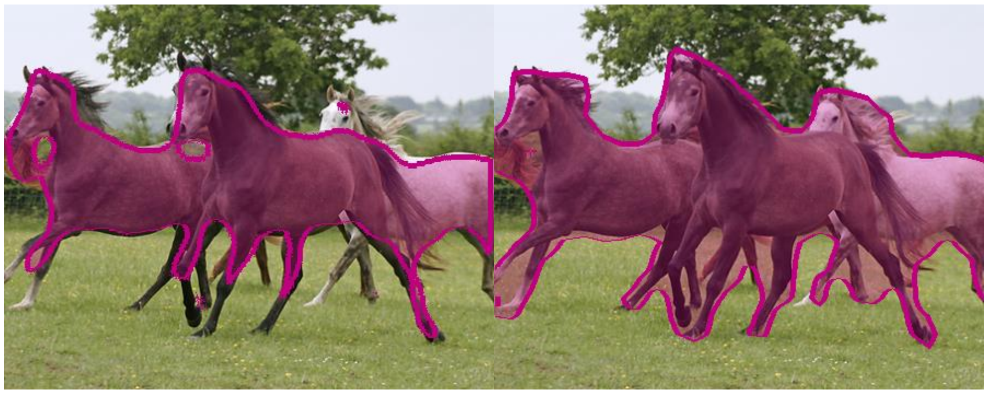
class #8 : 0.77812

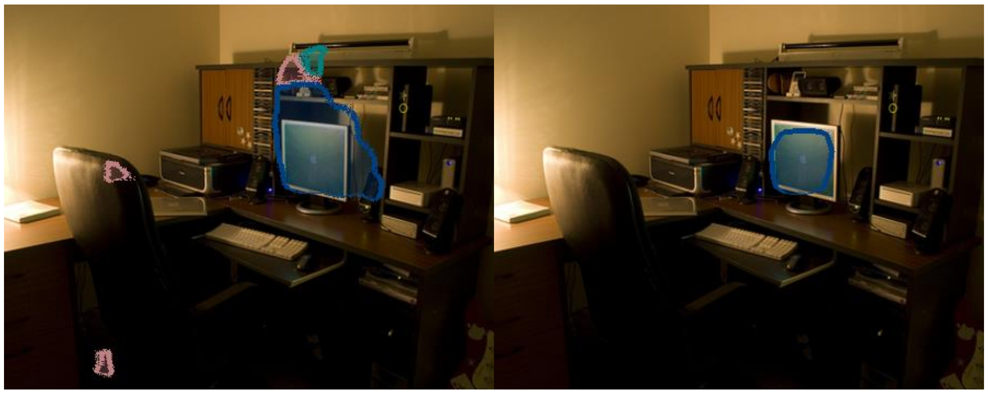
This is basically and increase in "only" a 5% compared to the initial one, but improving a model at such a high percentage gets hard really fast (to run some better models using Resnet101 I would have needed a much more powerful GPU than the one I was using if I wanted to finish on time).

Here are some main difference in the semantic segmenations between the baseline model and the improved one. The baseline are the images on the left and the improved model ones on the right:









These examples show how the improved model is better at distinguishing different objects and being more specific at what they are.

The first two show how it now distinguishes much better between cats and dogs. The third shows that it's more precise at figuring the actual shape of the object, since it correctly gets the legs of the horses and the face of the one on the back. The last images show how also the improved model is better at not setting random segmentation in areas where there shouldn't be (the only thing recognized in that room should be the monitor) although it also demonstrates that the monitor/tv class still has some troubles at recognizing the shape.

## Problem 2: Image Filtering

1. Gaussian Filters

We can do the Fourier transform of the 1D kernel for x and multiply it in the Fourier domain by itself (but using y instead of x)

This gives us the following:

$$G(X) \times G(Y) = \sqrt{\pi}\sqrt{\pi}\frac{1}{\sqrt{2\pi\sigma^2}}\exp-\frac{\pi^2x^2}{2\sigma^2}\times\frac{1}{\sqrt{2\pi\sigma^2}}\exp-\frac{\pi^2y^2}{2\sigma^2}=\frac{1}{2\sigma^2}\exp-\frac{\pi^2(x^2+y^2)}{2\sigma^2}$$

And this Fourier transform is exactly the same as the direct transform of the 2D kernel.

Also, If the transform is the same it means that when convolving both will act the same, since convolving is multiplying in the Fourier domain.

2. **Image after the Gaussian filter:**

This Gaussian filter simply slightly blurs the original image so it doesn't have such sharp edges.

3. **The kernels used to detect the edges of the image are the following:**

$K_y$=[-1/2, 0, 1/2]

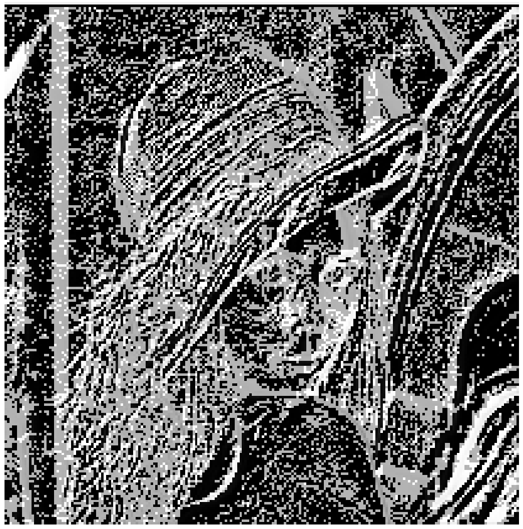
$K_x$=[-1/2, 0, 1/2] (transposed)

These kernels give us the following images:

**$I_x$** **$I_y$**

4. **With the Orinigal lena image:**

**With the Lena after the Gaussian filter:**



We can see how the one after the Gaussian filter has a much darker output, meaning it has stonger edges in it. This is because the added blur to the image makes the zones where the images are constant less smooth and therefore have more edges to it.

## Bibliography

[1] Improved Model Code base, https://github.com/FredHuangBia/PytorchDL/blob/master/models/PSPNet.py
[2] Resnet 101 example, https://www.programcreek.com/python/example/108008/torchvision.models.resnet101
[3] Image classification using pytorch, https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5
[4] Tensorflow tutorial, https://www.tensorflow.org/tutorials
[5] PSP Resnet, https://towardsdatascience.com/review-pspnet-winner-in-ilsvrc-2016-semantic-segmentation-scene-parsing-e089e5df177d
[6] Resnet guide, https://cv-tricks.com/keras/understand-implement-resnets/
[7] Relevant Stackflow to solve errors.

Students I collaborated with:

Ricardo Manzanedo-R08942139

Javier Sanguino-T08901105

Celine Nauer-A08922116

Julia Maricalva-A08922107