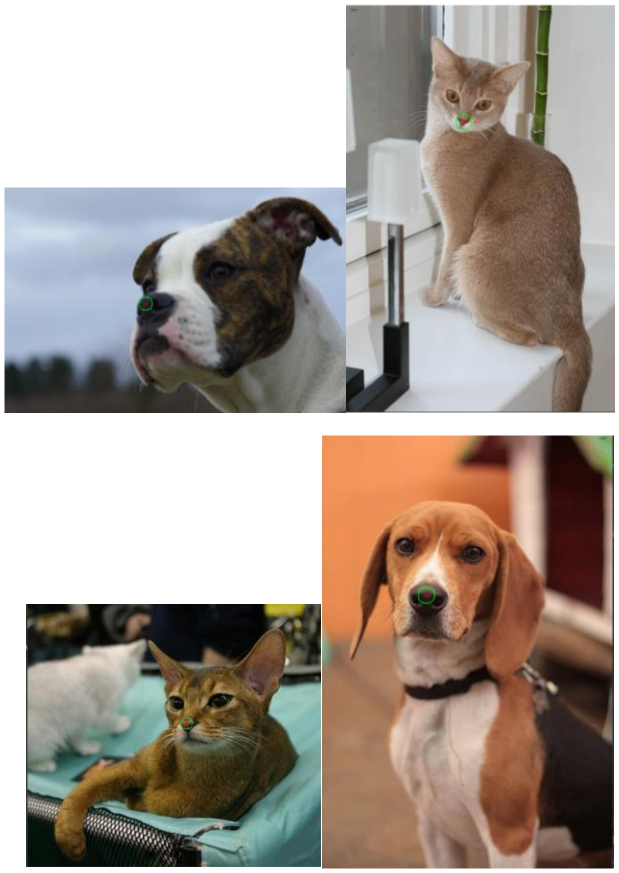
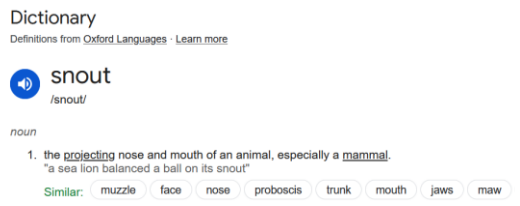


# ELEC 475

Lab 2 – Pet Nose Localization

# Lab 2 – Pet Nose Localization with SnoutNet

Yangzheng Wu and Michael Greenspan



Dictionary

Definitions from Oxford Languages · Learn more

snout

/snout/

*noun*

1. the projecting nose and mouth of an animal, especially a mammal.
   "a sea lion balanced a ball on its snout"

   Similar: muzzle face nose proboscis trunk mouth jaws maw

Copyright 2024, Kingston, Ontario, Canada

## Contents

1. Introduction ..... 1
2. Task ..... 1
2.1 Step One – Model ..... 2
2.2 Step Two – Data ..... 2
2.3 Step Three – Training ..... 3
2.4 Step Four – Testing ..... 3
2.5 Step Five – Data Augmentation Training and Testing ..... 3
3. Deliverables ..... 3
3.1 Completed Code ..... 3
3.2 Report ..... 4
4. Submission ..... 4

## 1. Introduction

The objective of this lab is to implement a *SnoutNet* model that can localize pet noses in images, and augment data to improve the performance. The dataset is ***oxford-iiit-pet-noses***, which is our reannotation of the ***oxford-iiit-pets*** dataset. You should be able to reuse elements of the Lab1 code base in pursuit of this goal.

## 2. Task

Your task is as follows:

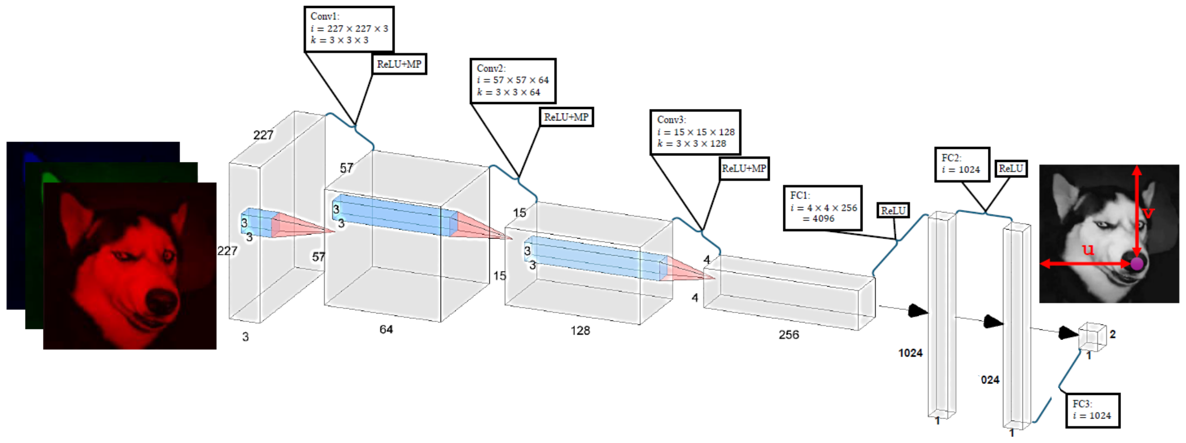


*Figure 1 – SnoutNet Architecture Diagram*

## 2.1 Step One – Model

Develop a model to regress (i.e. estimate) the pet nose locations in each image in the dataset, based on the diagram of SnoutNet in Figure 1. Code this into a separate 'model.py' source file. Keep in mind that you will need to follow the exact configuration of the architecture, as shown in Figure 1. The PyTorch official documentation contains all of the information for you to build SnoutNet, e.g. Conv2d — PyTorch 2.4 documentation, MaxPool2d — PyTorch 2.4 documentation . You will also need to reshape the feature map after convolution layers to feed it into fully connected (linear) layers (torch.Tensor.view — PyTorch 2.4 documentation).

After building your model, write a simple script and prepare a simple random dummy input tensor, with the shape of BxCxWxH = 1 x 3 x 227 x 227 (= batch_size x rgb_channels x img_width x img_height), and forward it into the model you created. If your network architecture has been coded correctly, then an output tensor of shape 1 x 2 (= batch_size x uv_coor_dim) should be expected.

## 2.2 Step Two – Data

**Download the data:** Download the ***oxford-iiit-pets*** dataset with reannotation from the ELEC 475 Google Drive into your local drive. (You should ***download only,*** please do not modify anything within the Google Drive). Alternatively, you can copy the dataset into your own Google Drive, and make it visible in your Google Colab (Snippets: Accessing files - Colab (google.com)). The dataset file structure is as follows:

```
- images-original
        -- images # actual  images
                --- Abyssinian_1.jpg
                --- Abyssinian_2.jpg
                ……..
        -- images.tar.gz # original oxford-iiit-pets data
- test-noses.txt #annotation of our test set
- train-noses.txt #annotation of our train set
```

**Develop a Custom Dataset:** Your custom Dataset class (Datasets & DataLoaders — PyTorch Tutorials 2.4.0+cu121 documentation) should be able to accommodate different types of data augmentation (Transforming and augmenting images — Torchvision 0.19 documentation (pytorch.org)). Your Dataset should initialize the path to images (images-original/images/), and groundtruth (GT) labels (train-noses.txt and test-noses.txt). The GT labels look like the following:

```
beagle_145.jpg,"(198, 304)"
shiba_inu_136.jpg,"(182, 203)"
english_cocker_spaniel_181.jpg,"(145, 293)"
english_cocker_spaniel_17.jpg,"(126, 206)"
```

```
chihuahua_165.jpg,"(122, 122)"
…
```

On each line, the string before the comma is the image's file name, and the quoted ordered pair after the comma is the uv (i.e. (x,y)) pixel coordinates of the nose. Be mindful with these coordinates, as different libraries handle these images differently, and the sequence of u and v coordinates can sometimes be flipped. You will also need to reshape the images as the input images to SnoutNet's should always be of size 3 x 227 x 227.

**Reality Check your DataLoader:** Once you have developed your Dataset and DataLoader, write a quick reality check routine that iterates through all images and labels, prints the values, and (optionally, selectively) visualizes the images and the GT labels. This is crucial, as incorrect data loading can lead to unpredictable behavior of the network.

### 2.3 Step Three – Training

Write a training script and train your SnoutNet model, using the train partition of the dataset. For this step, don't augment the data. Use the test partition for validation purposes (i.e. just to track the progress of the loss plot, and not as part of network parameter optimization). Keep in mind that only regression losses can be used as we are handling a regression task, letting the model estimate the location of pets' snouts (as uv-coordinates within the image frame).

### 2.4 Step Four – Testing

Test your trained model on the dataset test partition. Calculate the localization accuracy statistics, i.e. the minimum, mean, maximum, and standard deviation of the Euclidean distance from your estimated pet nose locations, to the ground truth pet nose locations.

### 2.5 Step Five – Data Augmentation Training and Testing

Repeat Steps Four and Five using at least two different types of data augmentation. Include these as options in your training and testing scripts.

## 3. Deliverables

The deliverables as described below comprise the following:

- Your completed code;
- Your report.

### 3.1 Completed Code

In addition to the code itself, provide the following two scripts to train and test your code (from the PyCharm terminal):

```
train.txt
test.txt
```

Each of these scripts can contain multiple command lines, e.g. for different augmentations. Make sure that each script contains a command line argument that points to the absolute path of the data. When the TAs evaluate your code, they will only execute statements within these scripts (in the PyCharm terminal), so make sure that they work correctly!

### 3.2 Report

You report should be relatively brief (4-6 pages), and include the following information:

1. What is your network architecture?
   You should describe the SnoutNet structure in sufficient detail so that it can be reimplemented solely from the textual description.
2. How did you implement your custom Dataset?
   You should describe how the images and labels are loaded, and how you conduct a quick fact check on the Dataset and DataLoader you developed.
3. Describe all hyperparameters used in training, and describe the hardware used. How long did the training take? Include the loss plot.
4. Describe the different methods of data augmentation that you implemented.
5. Describe your experiments. Include a description of the hardware that you used, and the time performance for training for each augmentation variation, and for testing (e.g. msec per image). Describe the localization accuracy statistics, as an error measure. Include a table that shows an ablation study of the different augmentation methods, and how they impacted the quantitative results, e.g.:

| Augmentation | | Localization Error | | | |
|---|---|---|---|---|---|
| 1 | 2 | min | max | mean | stdev |
| | | | | | |
| x | | | | | |
| | x | | | | |
| x | x | | | | |

   Also include some qualitative results (i.e. images of localized pet nose images) to support your claim. These can include both successful results, and some failure cases.
6. Discuss the performance of your system. How well did it perform? Was the performance as expected? Discuss which (if any) of the augmentation methods were beneficial. What challenges did you experience, and how did you overcome them?

## 4. Submission

The submission should all of your source code, and the report described in Section 3.

All deliverables should be compressed into a single **<zzz>`.zip file,`** where filename **<zzz>** is replaced with your student number. If you are in a team of 2, then concatenate both student numbers, separated by an underscore. The zipped directory should include your code and scripts, your trained parameter files, and all output plots and images.

Do not include the Google Drive data that you downloaded in Step 2.2 in your zip file.

The report should include a title (e.g. minimally ELEC 475 Lab 2), your name and student number. If you are working in a team of two, then include the information for both partners in the report (but only make one submission in OnQ).

Your code should execute by entering the syntax provided in your two included scripts (i.e. **`train.txt`** and **`test.txt`**) on the PyCharm terminal.

The marking rubric is as follows:

| Item | mark |
|---|---|
| Step 1 | 1 |
| Step 2 | 1 |
| Step 3 | 1 |
| Step 4 | 1 |
| Step 5 | 1 |
| Report | 2.5 |
| Correct submission format | 0.5 |
| **Total:** | **8** |