Stephen J. R. Smith Faculty of Engineering & Applied Science – Mechatronics & Robotics

ELEC 475: Computer Vision with Deep Learning

Lab 2 – Pet Nose Localization with SnoutNet

Aria Maz | 20278460

September 24, 2024

1. Network Architecture

The network architecture consists of a combination of convolutional layers for feature extraction and fully connected (FC) layers for regression. Information about the convolutional and FC layers are summarized in **Table 1** and **Table 2** respectively. The Weighting initialization of the layers are summarized in **Table 3**.

		Conv Layer 1	**Conv Layer 2**	**Conv Layer 3**
Input	Type	3-channel RGB image	Feature map	Feature map
	Size	227 x 227 x 3	57 x 57 x 64	15 x 15 x 128
Convolution / Kernel Size	Number of filters	64	128	256
	Size of filter	3 x 3	3 x 3	3 x 3
	Stride	1	1	1
	Padding	1	1	1
Max Pooling	Kernal Size	4 x 4	4 x 4	4 x 4
	Stride	4	4	4
	Padding	1	1	1
Output	Type	Feature map	Feature map	Feature map
	Size	57 x 57 x 64	15 x 15 x 128	4 x 4 x 256
Activation		ReLU	ReLU	ReLU

Table 1: Convolutional layers with specifications on each layer.

		FC1	**FC2**	**FC3**
Input	Type	Flattened feature vector	Neurons	Neurons
	Size / Number	4 x 4 x 256 = 4096	1024	1024
Output (neurons)		1024	1024	(u,v) = 2
Activation		ReLU	ReLU	None

Table 2: FC layers with specifications on each layer.

	Convolutional Layers	**FC Layers**
Weight Initializations	Kaiming normal	Xavier uniform initialization.
Initialized Biases	0	0.01

Table 3: Weighting initialization of convolutional and full connected layers and their respective biases.

2. Custom Dataset Implementation

The custom dataset, (PetNoseDataset), is designed to handle the oxford-iiit-pet-noses dataset.

2.1 Loading Data

Initialization (__init__ method)

Parameters

- images_dir: Relative path to the images (./Data/oxford-iiit-pet-noses/images-original/images \).
- labels_file: Relative path to the GT labels (./Data/oxford-iiit-pet-noses/<**train or test**>_noses.txt \).

Loading Labels

Reads the labels file line by line. Then parses the image (strips quotation marks and parentheses, then splits by the comma) filename and extracts the (u, v) coordinates, converting them to floats and stores the image filename and coordinates as tuples in a list self.labels.

Dataset Length (__len__ method)

Returns the total number of samples in the dataset, which is the length of self.labels.

Getting Items (__getitem__ method)

Parameters

- idx: Index of the sample to retrieve.

Process

Retrieves the image filename and coordinates from self.labels using the index. Constructs the full image path and opens the image using PIL, converting it to RGB. It then applies augmentations (if any) to both the image and coordinates. Records the original image dimensions (width and height) and resizes the image to 227 x 227 to match the input size expected by SnoutNet. It Calculates scaling factors based on the original and resized image dimensions and adjusts the (u, v) coordinates according to the scaling factors. It Applies transformations (e.g., converting the image to a tensor), prepares the target tensor containing the adjusted (u, v) coordinates, and returns the processed image and its corresponding target coordinates.

2.2 Checking Dataset and DataLoader

To ensure the correctness of the dataset and data loading process, a data loader check script (data_loader_check.py) was implemented.

Visualization of Samples

Selects a specified number of samples (5) from the dataset and prints the sample number, image shape, and target coordinates and displays the image with a red dot indicating the GT nose position for each sample.

Verification Steps

Ensures that the images are correctly resized to 227 x 227 and confirms that the scaling of coordinates is accurate after resizing. It checks that any augmentations applied adjust the coordinates appropriately and validates that the transformations (e.g., ToTensor) are correctly applied.

3. Training

3.1 Hyperparameters

- **Batch Size (-b):** 16 (number of samples processed before the model is updated)
- **Number of Epochs (-e):** 50 (number of complete passes through the training dataset)
- **Learning Rate (-l):** 1e-3 (controls how much to adjust the model's weights with respect to the loss gradient)
- **Weight Decay (-w):** 0 (L2 regularization parameter to prevent overfitting)
- **Optimizer:** Adam (adaptive optimizer that adjusts the learning rate for each parameter)
- **Training DataLoader:** Shuffled (shuffles training data)
- **Validation DataLoader:** Non-shuffled (keeps validation data in the original order)
- **Loss Function:** Euclidean Distance Loss,

$$Loss = mean(\sqrt{\left(u_{predicted} - u_{target}\right)^2 + \left(v_{predicted} - v_{target}\right)^2}.$$

- **Early Stopping Patience:** 5 (stops training if the validation loss does not improve for consecutive epochs)
- **Learning Rate Scheduler:** ReduceLROnPlateau (reduces the learning rate when the validation loss plateaus)
  - **Mode:** 'min'
  - **Patience:** 3
  - **Factor:** 0.5
- **Augmentation:** hlfip (applies a horizontal flip to the image)
  - **hflip probability:** 0.5
- **Augmentation:** shift (applies a random shift to the image**)**
  - **Shifts maximum range:** $\pm$10 pixels
- **Weight initialization (convolutional layers):** Kaiming normal (Sets the weights of convolutional layers)
  - **Biases:** 0
- **Weight initialization (FC layers):** Xavier uniform initialization (Sets the weights of FC layers)
  - **Biases:** 0.01

3.2 Hardware

The device for the training environment runs on a GPU if available, otherwise CPU. For this specific environment, it ran on a **2.3 GHz Quad-Core Intel Core i7 processor** with **32 GB of RAM** and **Intel Iris Plus Graphics (1536 MB) integrated GPU**

3.3 Training time

The training time and corresponding loss plot for each combination of data augmentations are shown in **Figure 1**, **Figure 2**, **Figure 3**, and **Figure 4**.

Training Time: 2 h, 32 min, and 22 s

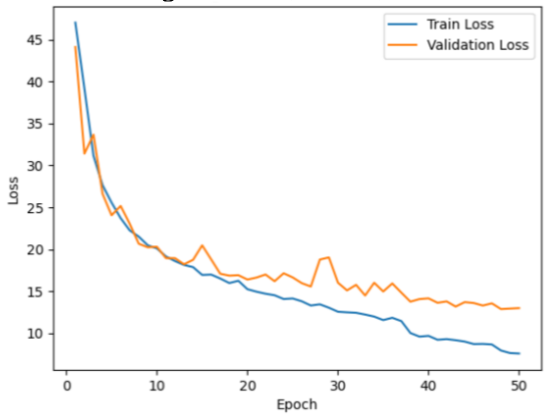

Figure 1: Training and validation loss curve plot for both hflip and shift data augmentations over 50 epochs during training.

Training Time: 1 h, 23 min, and 39.34 s

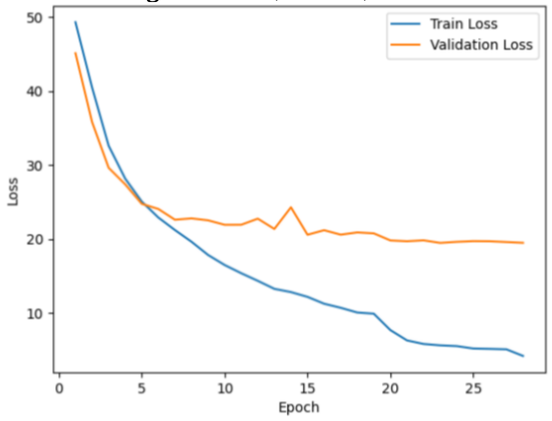

Figure 2: Training and validation loss curve plot for neither data augmentations over 28 epochs during training.

Training Time: 2 h, 36 min, and 37.23 s

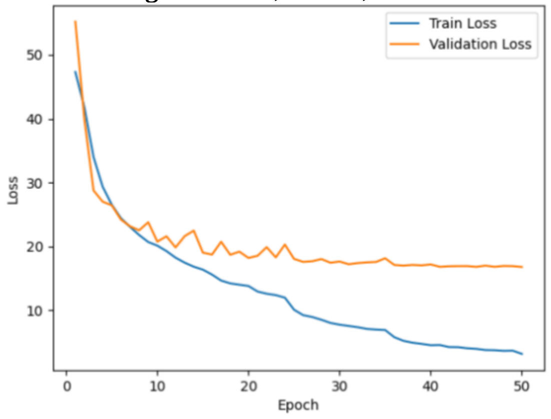

Figure 3: Training and validation loss curve plot for the hflip data augmentation over 50 epochs during training.

Training Time: 2 h, 37 min, and 55.53 s

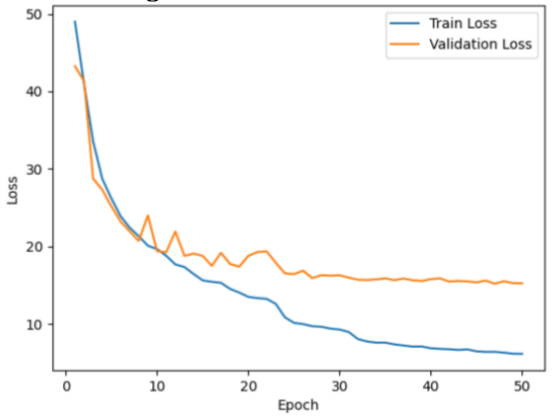

Figure 4: Training and validation loss curve plot for the shift data augmentation over 50 epochs during training.

4. Data Augmentation

To enhance the model's ability to generalize and to mitigate overfitting, the diversity of training data was increased through the implementation of data augmentation techniques during the data loading phase in the custom dataset.

The apply_augmentations function iterates over the list of augmentations specified in the command-line arguments. Augmentations are only applied to the training dataset, not to the validation or test datasets.

4.1 Horizontal Flip (hflip)

Description: Uses torchvision.transforms.functional.hflip to flip the image horizontally (mirror the image along the vertical axis) with a probability of 0.5.
Coordinate Adjustment: The v coordinate remains unchanged, and the u coordinate is adjusted to reflect the flip,

$$u_{new} = Image\ Width - u_{original}.$$

4.2 Random Shift

Description: Uses torchvision.transforms.functional.affine (with translation parameters) to shift the image randomly in both horizontal and vertical directions within a maximum range of ± 10 pixels.
Coordinate Adjustment: Both (u, v) coordinates are adjusted according to the random integer shift values (dx, dy),

$$u_{new} = u_{original} + dx,$$
$$v_{new} = v_{original} + dy.$$

The new coordinates are clamped to ensure they remain within the image bounds.

5. Experiments

Several experiments were conducted to evaluate the performance of SnoutNet and the impact of data augmentation on nose localization accuracy.

5.1 Quantitative Results

An ablation study of the different combination of augmentation methods, and how they impacted the quantitative results is shown in **Table 1** and **Table 2**. Note, all the training was conducted on the hardware specified in section 3.2 (Hardware).

Augmentation		Localization Error				Average Time performance	
Horizontal Flip	Random Shift	Min Distance (px)	Max Distance (px)	Mean Distance (px)	Stdev (px)	Training (s/epoch)	Testing (ms/image)
✓	✓	0.27	117.11	12.90	14.05	182.83	9.24
✓	✗	2.29	135.24	45.85	24.15	187.96	11.74
✗	✓	0.13	134.35	15.17	16.15	189.51	14.52
✗	✗	0.37	136.38	19.47	17.52	179.26	9.40

Table 4: Augmentation method combinations, and how they impacted the localization error and average time performance.

Augmentation		Accuracy Metrics			
Horizontal Flip	Random Shift	Within 5px (%)	Within 10px (%)	Within 20px (%)	Within 20px (%)
✓	✓	24.68	55.81	83.50	97.42
✓	✗	1.15	4.59	13.63	59.97
✗	✓	19.08	49.50	80.63	95.84
✗	✗	9.47	32.28	66.14	93.83

Table 5: Augmentation method combinations, and how they impacted the accuracy metrics within different pixel ranges.

5.2 Qualitative Results

Some of the good, average, and bad outcomes of the SnoutNet model regressing/localizing pet nose locations in images by predicting the (u, v) pixel coordinates for different combination of augmentation methods are shown in **Figure 5**, **Figure 6**, **Figure 7**, and **Figure 8**.

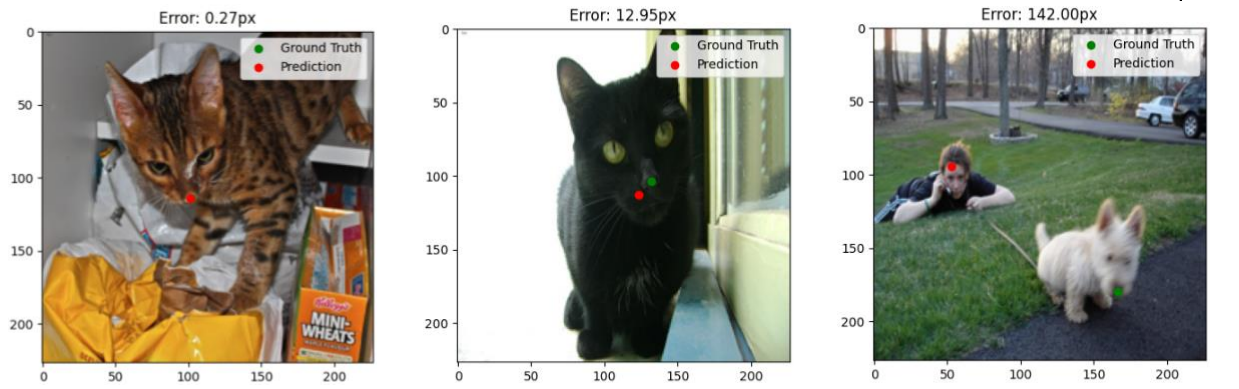

Figure 1: The best, mean, and worst (from left to right) pet nose location predictions for both hflip and shift data augmentations.

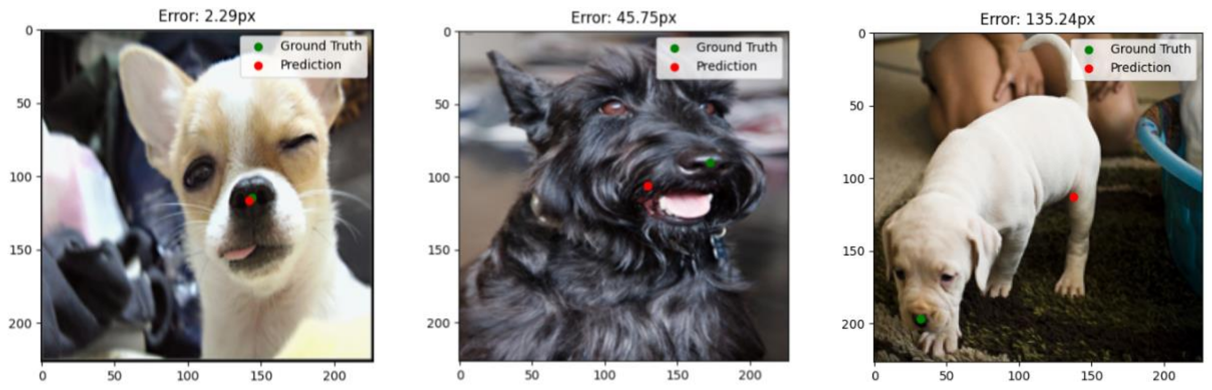

Figure 2: The best, mean, and worst (from left to right) pet nose location predictions for the hflip data augmentation.

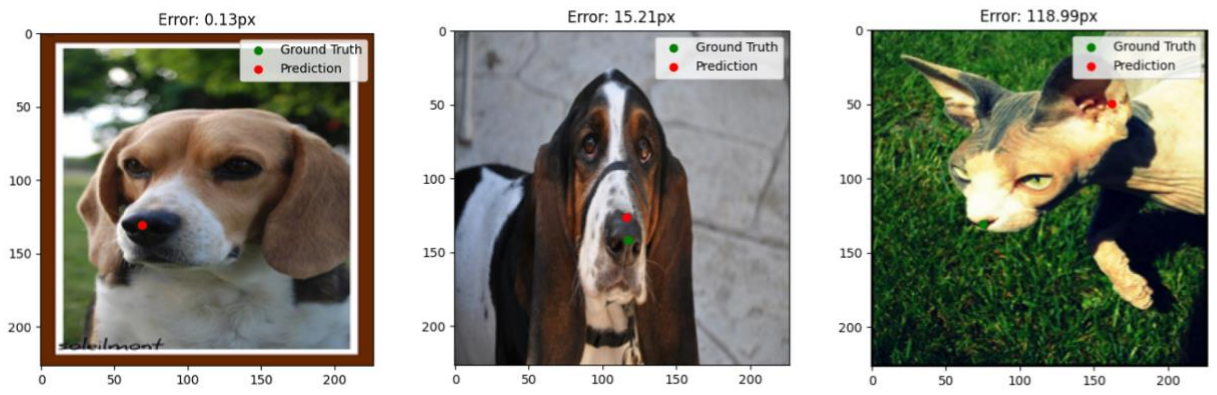

Figure 3: The best, mean, and worst (from left to right) pet nose location predictions for the shift data augmentation.

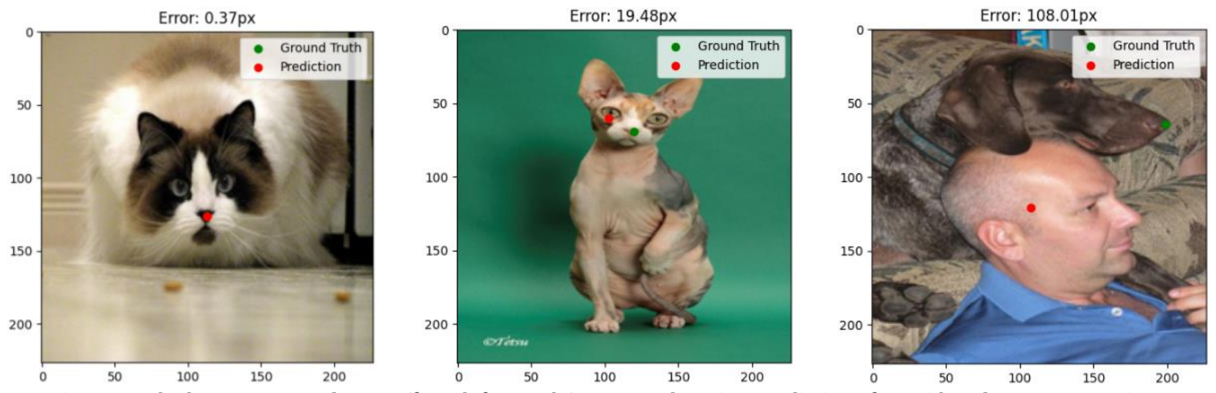

Figure 4: The best, mean, and worst (from left to right) pet nose location predictions for neither data augmentations.

6. Performance

The model trained with 'hflip and shift augmentations' achieved the best overall performance. This indicates that combining both augmentations significantly improves the model's ability to localize pet noses accurately. Its performance reflects its validation and train loss curve, decreasing steadily over epochs with the best convergence and minimal signs of overfitting.

The model trained with a shift augmentation was the second best, and the model trained with neither augmentation was a close third; it had an early stopping at the 28th epoch due to the overfitting from the validation loss curve flattening out while the train loss curve continued to decrease.

The model trained with only a hflip augmentation performed the worst. The model had a mean deviation of about 46 px, and the high standard deviation of 24.15 px suggests inconsistency in the model's predictions. This suggests that the model is close to blindly guessing where the pet's noses are and getting lucky or unlucky sometimes.

Even though the hflip augmentation had the worst performance by far and degraded the model (more than double as much as neither augmentation) and the shift augmentations performance was very close to neither augmentation, somehow, when combining the 'hflip and shift augmentations', they performed the best out of all combinations.

Another interesting result was that the 'hflip and shift augmentations' validation loss curve had the lowest value at the final epoch compared to every other combination of augmentations. However, the other combinations of augmentations had a lower train loss curve at the final epoch, including the neither augmentations method, which had an early stopping at just over half the epochs.

These two results differed from what was expected initially and seemed unrelated. However, there is a correlation between the symmetric and predictable nature of the augmentation and how the model performed. The hflip augmentation alone confused the model due to the symmetry of the augmentation; however, when combined with the randomized shift, it now has positional and orientation variations. The combined augmentation becomes more diverse/varied, complex and performs best. With more varied data, the model must learn to generalize over broader examples. This increased complexity, resulting in less overfitting and, explains the second result related to the loss plots.

The main challenge experienced was the slow computing time of 2 hours and 30 minutes for one model to train over 50 epochs. This becomes a challenge because when tweaking parameters, it is best to tweak one parameter at a time to observe/pinpoint its effects since tweaking multiple parameters at a time can lead to confusion on which parameter did what. This process of increasing or decreasing one variable and waiting two and a half hours to analyze the result was overcome by proper time management and setting alarms at predicted times of when the model would be done based on previous iterations. This challenge was overcome since the lab was conducted over a long period of time.