# ELEC 475 Lab 2

Ethan Parliament

Student number: 20220776

# Network Architecture

The model (SnoutNet) is constructed using multiple convolutional layers and fully connected layers. Each convolutional layer has a 3x3 kernel size, and includes a 2x2 max pooling function (stride of 2, padding of 0), and a ReLU activation function.

Input layer: Input shape of 2x227x227.

Conv layer 1: 64 filters, stride of 2, padding of 1. Output has shape 64x57x57.

Conv layer 2:  128 filters, stride of 2, padding of 2. Output has shape 128x15x15.

Conv layer 3: 256 filters, stride of 2, padding of 1. Output has shape 256x4x4.

Flattening layer: Flattens the tensor to shape 1x4096.

FC 1: 4096 input units, 1024 output units with ReLU activation function.

FC 2: 1024 input units, 1024 output units with ReLU activation function.

FC 3: 1024 input units, 2 output units.

# Custom Dataset

A custom dataset class was created for the model that could read images from the specified directory using the python PIL module, and would read in the corresponding label from the provided file path. The custom dataset implementation also provided support for applying transformations to the images, which was used to resize each image into the same size (227x227), as well as update the target coordinate to match the new image size. Other transformations could be applied to augment that dataset with additional images. Before the model was trained, a sanity check was conducted on the dataset and data loader to ensure the images and labels were being retrieved correctly. This was done by looping through the images in the dataset and displaying them with a plot of the label coordinate on the image. Once it was determined that the dataset and data loader were properly configured, training the model could begin.

# Hyperparameters

Many different hyperparameters were tested while training the model in order to achieve the best results. This included changing the batch size, learning rate, weight decay, optimizer, scheduler, and different loss functions. The final hyperparameters used are shown below.

| Hyperparameter | Value used |
|---|---|
| Batch size | 64 |
| Learning rate | 1e-3 |
| Weight decay | 1e-5 |
| Optimizer | Adam |

| | |
|---|---|
| Scheduler | ReduleLROnPlateau |
| Loss function | Average Euclidean distance from target label |

The final hyperparameters were decided through trial and error, and the highest achieving set of hyperparameters was chosen based on validation loss scores. The learning rate and weight decay values tested varied from 1e-2 to 1e-5. Different optimizers were tested, including Adam, AdamW, and SGD. Adam had the best performance, but the difference between them was negligible. The loss function used was a custom implementation that took the average Euclidean distance from the predicted coordinate to the target coordinate in each batch. Different loss functions were tested as well, but the chosen loss function provided much better results than any other method considered.

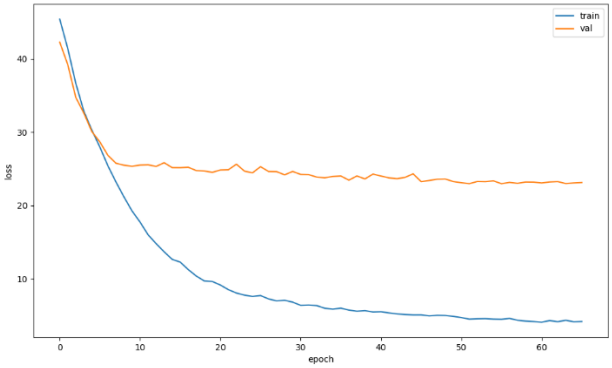
# Data Augmentation

In hopes to increase the diversity of the training dataset, which can improve the generalization of the model, the images of the dataset had different operations applied to them. The 2 operations that were tested include a Gaussian blur of the image, and flipping the image horizontally. These augmentations were used alongside the original dataset to double, and even triple the number of images that were used in the training process. An example of the augmentations with the original image can be seen below.


# Performance

The model was trained using a NVIDIA GeForce MX250 GPU, which took about 12.7 ms per image while training the dataset with no augmentations. While testing the model, it took about 7 ms per image to make a predication.

The loss plot for each of the models trained look very similar, with the train set converging at a value around 5, and the loss of the validation set at a much higher value. There seems to be some overfitting in each of these models, as the training loss continues to drop while the validation loss stops decreasing around the 10$^{th}$ epoch. A sample loss plot of the training set including both augmentations can be seen in the figure below.




Overfitting was one of the biggest challenges with this model. Different hyperparameters such as weight decay and optimizers were used to help the model generalize better, with little to no success. It seems that a possible solution to help the model generalize better is to simplify the model by reducing the number of layers (or trainable parameters) that it has. However, given the strict instructions to use the model described, this was not possible for this problem.

The performance of the model around what was expected. The mean localization error of 27.6 is nothing incredible, but much better than some of the other models that were tested in this experiment. The results from training with the original dataset provide a baseline to compare the augmented datasets to. Data on the localization error and training time for each augmentation added is found in the table below.

| Augmentation | | Localization Error | | | | Average Training Time (per image) |
|---|---|---|---|---|---|---|
| Blur | Horizontal flip | Min | Max | Mean | stdev | |
| | | 0.9864 | 167.0851 | 27.5758 | 20.7017 | 12.7 ms |
| X | | 0.6924 | 147.53 | 28.9892 | 20.7884 | 25.5 ms |
| | X | 0.6638 | 141.1894 | 23.5466 | 19.5524 | 25.5 ms |
| X | X | 0.2604 | 178.693 | 23.0975 | 20.9253 | 38.2 ms |

## Blur

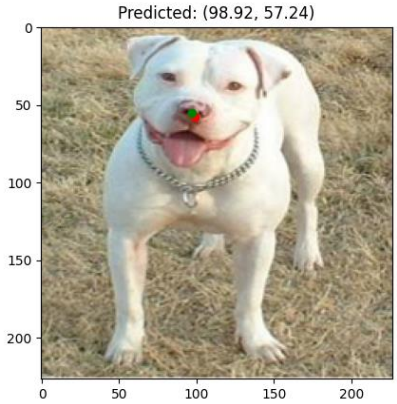
When the blur augmentation was added to the training set, the mean error surprisingly increased. However, this augmentation lowered the minimum and maximum distance compared to the original dataset, showing that it can provide some sort of improvement to the model, although a small one. This method of augmentation possibly could have been improved on through experimenting with different strengths of blur used. Perhaps a stronger blur on the images would have helped the model generalize further.

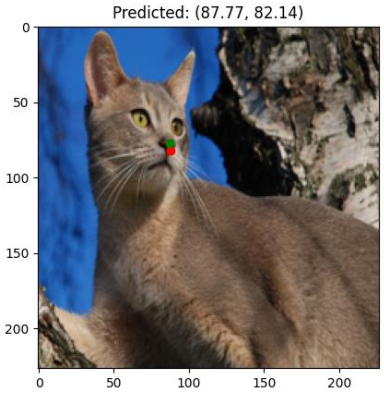
## Horizontal Flip

The horizontal flip augmentation provided great improvements compared to both the original dataset, and the blur-augmented dataset. It decreased the mean error from 27.6 to 23.5 as well as decreasing the minimum and maximum errors. This method of augmentation worked well, and shows that similar augmentations such as rotation, or resizing the image may provide similar improvements to the model.

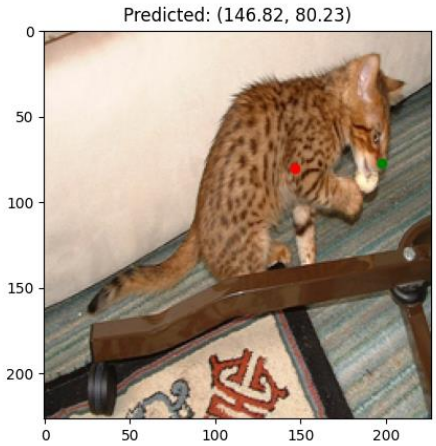
# Sample Predictions

Judging the predictions qualitatively, most of them are close to the actual label for the image, landing on the snout of the animal, or near it. Some examples of this behaviour are shown in the images below. The actual label is shown with a green dot, and the predicted coordinate is shown with a red dot.



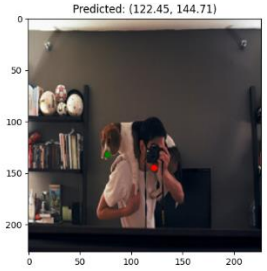

There are also quite a few predictions that were not great, with the predicted location not landing on the snout, or even the head of the pictured animal.

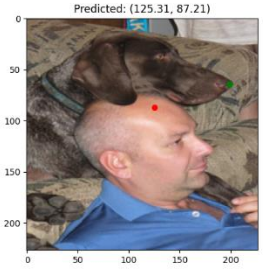


## Outliers

Within the dataset, some outliers were identified that are worth discussing. It is worth noting that not every image from the dataset was viewed, so it is unknown how often these types of images show up – an estimate of this would be about 1 in 30 images.

The following images have particularly bad predictions associated with them. The images include a dog, as well as a person – both in the center of the images.





After seeing this, issues with other images were noticed. There were images where the snout was obstructed by an object, or the image quality was poor so even the labeled coordinate was in the wrong spot. It is worth considering if these types of images should have been included in the dataset for this problem, as having outlier images can make predicting the location of the snout much more difficult. Removing these types of images before training the model could have a drastic impact on the success of the model.