ELEC 475 Lab 5

Lucas Coster

20223016

2023 – 12 – 05

Pre-Processing

The first problem was turning the provided data into usable training data that could be easily fed into a model. The dataset included two text files, one labeled test_nose.txt and one named train_nose.txt. It also came with a fill image folder which included many pet images. Each line in the text files referenced one of the specific images and had XY ground truth coordinates for the location of that pet's nose. A custom data loader was built which took each line in the text file and attached the labeled XY values to the images in the folder. The problem was further complicated by the fact each image had to be resized to the same number for use within the model so all XY coordinates also had to be rescaled. To solve this a function was created that normalized and rescaled all XY ground truth coordinates based on a specified image size, in this case, 224x224 was the decided upon dimensions. After that was done two new text files were saved with the rescaled ground truth coordinates to be used with the resized images. The custom data loader along with a class for rescaling and a class for file input-output allowed for the data to be properly prepared for training.

Model Description

For this project, the model decided upon was a custom CNN regressor which aimed to locate the XY coordinates in the image. The model was built using torch.nn and was called XYLocationImageRegressor(). The model was inspired by classic CNN structures which are commonly used and effective in image classification. It contains a sequential pattern of convolutional layers followed by max-pooling layers and fully connected layers. CNNs are highly effective at capturing hierarchical features from images and max-pooling layers effectively reduce spatial dimension while retaining important features. The fully connected layers at the end allow the model to map the features extracted by the CNN layers to the final output space which is the coordinates of the nose.

The exact model architecture includes the three main sections outlined above, CNN layers, max-pooling Layers, and fully connected layers. The first convolution layer is applied to the input image and has 64 filters with a kernel size of 3x3 with Relu activation and 1-pixel padding. Following that is a max pooling layer with a kernel size of 2x2 and a stride of 2. The model then feeds into a second CNN layer with 128 filters and a kernel size of 3x3. It also uses 1-pixel padding and Relu activation feeding. It is followed by another max-pooling layer which has the same features as the last one. The third and final CNN layer has 256 filters and a 3x3 kernel size along with the same 1-pixel padding and Relu activation as the previous layers. It too feeds into an identical max-pooling layer. The Relu activation layers after each CNN layer introduce non-linearity. After the final max-pooling layer the model is flattened and fed into a fully connected layer with 512 output features. The final layer is another fully connected layer that takes the 512 features as input and outputs a final 2 features which represent the x and y coordinates of the nose.

The model is a purpose-built CNN for predicting nose positions in images. It leverages convolutional and pooling layers for feature extraction and fully connected layers for regression. The model's simplicity makes it suitable for the specific task at hand without unnecessary complexity.

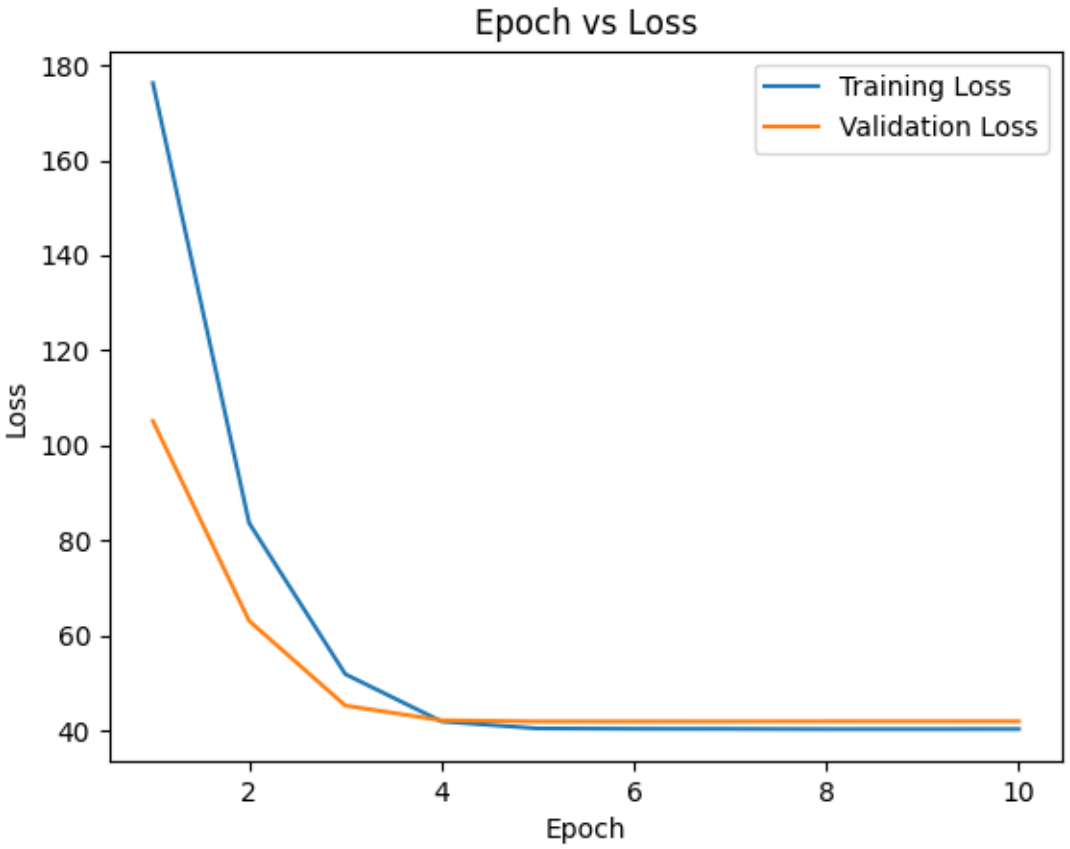
Training Process

The training process took the data loader-created model described above and ran through training loops using both the training data and the testing data for validation. The goal of the training process was to reduce the loss value and improve the accuracy of the model on the data so it could learn how to

properly identify the pet noses from the images. The training process relied on hyperparameters to properly guide training and give the best results. The hyperparameters were chosen based on several trial-and-error tests along with some background research and qualitative findings. The number of epochs chosen was 10 as it was noticed the loss curve tapered out around epoch ten and there was no significant change after that. The learning rate was chosen to be 0.01 which is higher than a typical learning rate but was raised as in earlier tests the convergence was extremely slow. The chosen optimizer was SGD as compared with Adam it gave better loss results for this specific problem, it used a weight decay of 1e-4 and a momentum of 0.9. A StepLR scheduler was used with a gamma of 0.8 and a step size of 15 to assist with model convergence. The loss function used was a Euclidian loss function which was calculated in a separate class using the following code:

```
torch.sqrt(torch.sum((pred - target) ** 2, dim=1)).mean()
```

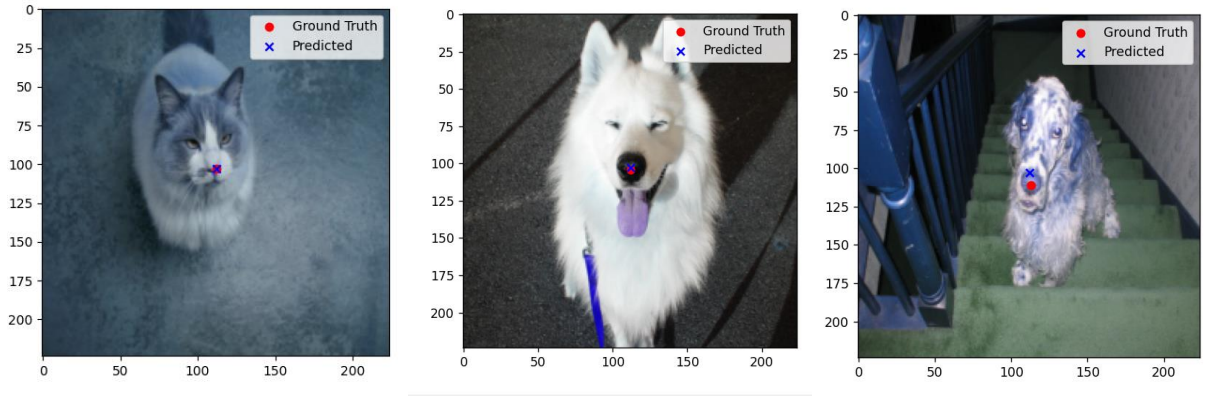
This loss function gave better results than other common loss functions due to its direct application in the coordinate calculation field. All these hyperparameters and functions together guided the training process producing the best possible results. The model was trained on a laptop GPU, specifically the GeForce RTX 3050, and each epoch took around 90 seconds to complete causing the whole model to train in around 900 seconds or 15 minutes.



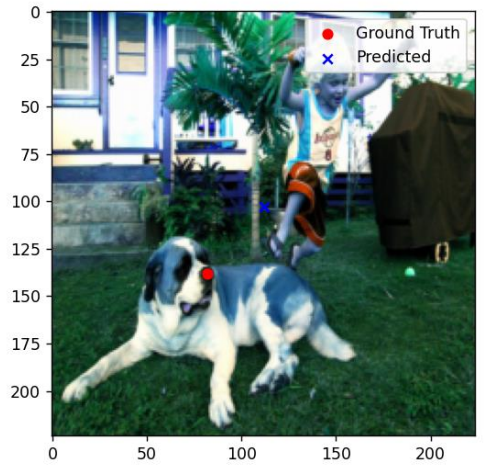
The loss plot depicted above shows both the training and validation loss values over time and clearly shows the model regressing down to its best accuracy.

Testing Results

After training the model was saved and tested on the test_noses.txt images. These images were separated from the training images to prove the model had been properly trained. Each test image was fed into the trained model to get the predicted nose location. This predicted nose location was then compared with the ground truth nose location using Euclidian distance to gain localization statistics. The statistics calculated were minimum distance, which was 0.46 pixels, mean distance which was 41.86 pixels, largest distance which was 123.06 pixels and standard deviation which was 23.13 pixels. Along with the localization results a general accuracy was calculated using a threshold of 0.5 or 50% of the radius. This accuracy came out to be 71.2% for the entire test set. The testing of each image was very quick at around 0.15s per image and the testing function was run on the same GPU as the training.



Pictured above are three example outputs from the testing loop with the ground truth red dot and the predicted blue X showing the model's prediction of the nose. The model was able to accurately predict the nose positions of pets in many images and in the cases shown above did it with extreme precision. The model did run into problems when the pet was not the focus of the image as shown in the example below:



It is expected images like this are what could have caused the largest pixel variation and reduced model accuracy. Moving forward the model could be improved by training more images like this to improve diverse condition accuracy. The general accuracy of 72% within a threshold of 0.5 radii proves the model still has room to improve and results were not perfect.

Moving forward to the discussion, difficulties, and improvements will be discussed and explained further, bringing together the final findings of the report.

Discussion

The entire system performed well being able to predict nose positions with decent accuracy. Performance was higher than expected due to the fact the classification model did not have a low loss value and there was worry the model had not been trained properly. This is most likely due to the way the loss function works and did not accurately capture the training of the model. The two main challenges came in the pre-processing stage and the model selection stage. The scaling and descaling of the images along with their ground truth coordinates was a complicated problem that caused a large initial delay in the project. The final solution involved including a separate class function that took the regular test and train text files and spat out two new text files containing the scaled ground truth numbers for each image.

Difficulties in the model selection were simply due to trial and error. Throughout the model selection process, an estimated ten different models were created, tested, and changed in multiple iterations to find the best ones. Pre-trained models such as Resnet-18 were used along with fully connected models but after multiple training iterations, the CNN model was finally decided upon. To further improve this project the CNN model would have to be further enhanced especially if we wanted to handle diverse cases such as the one outlined in the image above.

The project introduced many complex challenges that required further research as well as understanding class content to solve furthering my understanding of the process of making a full computer vision model from start to finish.