ELEC 475 Lab 5

Luka Gobovic 20215231

Ryan Anderson 20151125

Model (Step 2)

To begin the lab, we started out by using ResNet18, with pretrained weights from PyTorch. We decided to use ResNet18 because we found that it performed relatively well for us in the last lab, and it is a well-known model with good performance [1]. Also, we believe that, since our dataset is not particularly large, we did not want to use an overly complex model for fears of overfitting to the train dataset. However, since this was a regression task, we needed to modify it from the start for it to output what we wanted. ResNet18 was trained on ImageNet, meaning its last layer was designed to have 1000 neurons to output the class probabilities, of which ImageNet has 1000 of. Since here, we are technically dealing with 2 outputs, the x and y coordinates, we needed to modify the 2 layers, which are an average pool and fully connected layer. We removed these 2 layers and simply added a fully connected layer that brought the output size to 2. From here, we began to test our model. We found that our results were overall not ideal, with a mean Euclidean distance of approximately 36 pixels. Through plotting and monitoring the validation loss, we also found that the model began to overfit after a certain number of epochs (depended on other parameters). To try to combat this, we added dropout after the fully connected layer, with a probability of 0.5. The dropout probability was a parameter that was adjusted a few times until we found the optimal value. We also readded the global average pooling layer after the last convolutional layer. This, along with adjusted hyperparameters, more of which will be talked about in the next section, helped improve our results. Our final mean distance was 5.7982, with a standard deviation of 9.395, both values being in pixels. A bullet point list of modifications can be found below:

- Removed Last 2 layers (average pool and fully connected layer)
- Readded the global average pooling layer
- Flattened the output of the pooling layer
- Applied dropout with probability of 0.5
- Fully connected layer down to output size of 2 (one for x coordinate, one for y)

Training (Step 3)

As for the hyperparameters used in the training, we experimented with a wide variety throughout the training, with certain changes causing improvements, and some causing a downturn in results. To begin, since the dataset is relatively small, we began with relatively small batch sizes, of 16 and 32. We found that while larger batch sizes did slightly improve the training time, it often lead to overfitting, so through more testing, we found that a batch size of 32 produced the best results. As for the optimizer, scheduler, and the loss function, we found a large amount of variation between each method tested. We began with a similar approach to last lab, using SGD, ReduceLROnPlateau and MSELoss. Within the optimizer, we used a learning rate of 0.001, and a momentum of 0.9. With these hyperparameters, we found suboptimal results, with mean distances of approximately 35 pixels, and overfitting behaviour. We decided to maintain the same scheduler throughout the testing as with the previous labs we did not find much difference with alternatives. Using MSE loss appeared to produce very high loss values, starting at around 11000. While the loss still decrease, it did not appear to converge well. From doing research, we found that SmoothL1 loss was a better alternative to MSE, and from our testing, we found that it improved our results significantly, dropping the mean distance to approximately 20 pixels. From here, we decided to test various optimizers, eventually settling on ADAM. We found that Adam provided better model convergence and less overfitting, if we also added a weight decay factor. With the scheduler, optimizer and loss function finalized, we still made changes to other hyperparameters. As

mentioned previously, we started out with a learning rate of 0.001, but found that this converged too slowly. We then tried other rates including 0.0001, 0.00005 and 0.00001, but found that 0.0001 produced the overall best results, both in terms of the model's convergence and the distance values.

From here, we proceeded with testing the number of epochs. We initially started out with 40 but found that this resulted in overfitting. From here, while we also reduced the number of epochs, we also implemented early stopping, which monitors the validation loss for increases. We used a patience of 5, meaning that if the validation loss did not decrease after 5 epochs, we would stop early and save the model. This helped reduce overfitting and often resulted in 28-32 epochs. From this, we were able to settle on a value of 30 epochs. The final loss plot can be seen below in Figure 1**.**

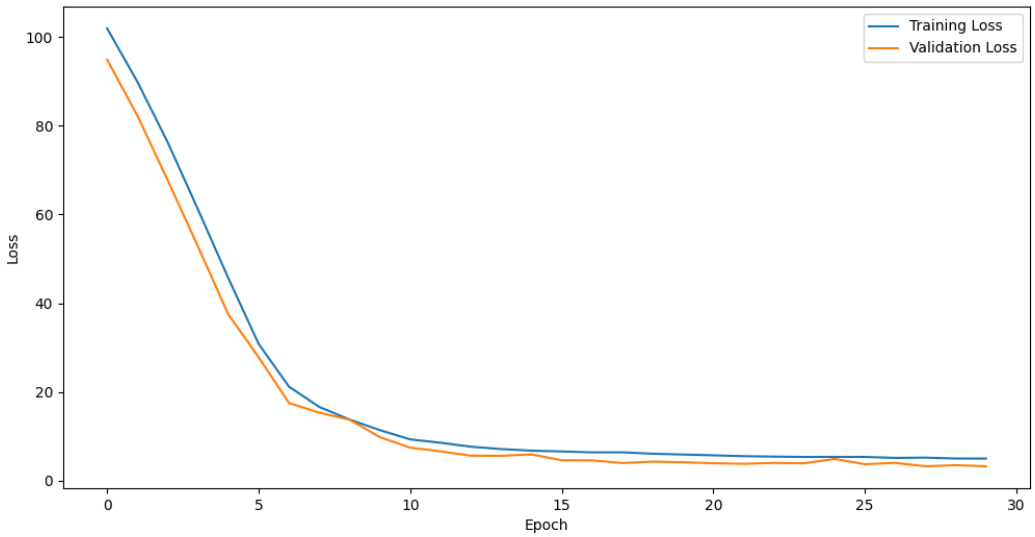

Figure 1: Loss plot for model, showing training and validation losses after 30 epochs

As for training time, this varied with batch size and the number of epochs, but the final figure we achieved on a desktop PC with an RTX 2060 Super was approximately 25 minutes. Sometimes, certain epochs would take longer than others. We presume this could be attributed to running another task on the computer at that time, potentially using more resources and slowing down the training time. We did not feel the need to test training time on Google Colab with more powerful hardware as training time was already relatively short.

Results (Step 4)

After completing the training, we began to test the performance of the model, in terms of its ability to localize the pet noses on the test dataset. After many model iterations, we finally reached results that we were satisfied with. We kept track of 4 metrics, (all in pixels) the minimum, maximum and mean Euclidean distances, as well as the standard deviation. We achieved values of 0.00, 134.6291, 5.7982 and 9.395 respectively. Although the maximum distance is 134, we assume that this is an outlier due to the low mean and relatively low standard deviation, indicating good model performance. The minimum distance of 0 indicates that there was at least one instance of the model guessing the nose coordinates exactly. Some of the results of the localized pet images can be seen below, with the ground truth nose being circled in green, and the predictions in red.

It is evident that the pets with fairly discernable noses performed better than those with obscure ones. Pets with noses that have similar colours or features to other parts of their faces are where the model seemed to struggle. One that it really appeared to struggle on was the first image in the last row, where the blurry pet nose was not recoginzed, rather the humans hairline. Additionally, from visualizing a large number of images, it seemed to be able to recognize cat noses better, but that could have been just from the images we looked at. From both the qualitiative results as well as the statistics, we believe that the model perfomed relatively well overall, with room for improvement in certain aspects. As for the inference itself, the same hardware used to train the model was used for evaluation aswell. The average processing time per image was approximately 4-5 milliseconds, showing fast overall inference times. One other point is to be noted. During training, we resized all of the images to 224 by 224, this however, meant that we needed to scale each of the labels to match the newly sized images. This was simply done

by scaling the original coorindate by a factor related to the original size and the new size. This ensured that the ground truth labels matched the new images.

Discussion

As for the overall performance of the system, we believe that it produced great results, considering the relatively small dataset size of approximately 8000 total images. We believe that given a larger dataset, we could have further reduced our mean and maximum distances. Given that we also used a well-known model in ResNet18, that was trained on a massive dataset, we believe that the results are as expected. At one stage of our training, we tried to implement more complex models, however, we believed that they became too complex for the size of the dataset, and therefore began to overfit. This is why we decided to proceed with ResNet18, as we knew that it would produce great results. Throughout the lab, there were no real challenges that hindered our progress; however, we did find that the dataset portion was one of the more complex aspects of this lab. Since we were dealing with a custom labels file, we needed to ensure that the data was parsed correctly, and that the x,y coordinates remained as a tuple that could be used in the model. This, along with the need to properly scale the images and the coordinates did take up a large portion of the hours put into this lab. Overall, the lab was a great opportunity to look at a regression task, which involved slightly different methodologies than the previous labs. However, it was a fun task, resulting in good results overall.

Bibliography

[1] K. He, "Deep Residual Learning for Image Recognition," Microsoft Research, 2015.