Prototyping a Privacy-Conscious Pet Monitoring System

Exploring the Adaptive Optimization of Deep Learning-Driven Semantic Segmentation Techniques

Allison Aprile

Background

Project Description

Pet monitoring systems, such as Furbo and Companion are becoming a necessity for pet owners. Some can even provide owners with real-time updates about their pets, or even allows owners to interact with them. However, these can be quite invasive of privacy!

Taking inspiration from Zoom's 'Blurred Background' feature, **I am aiming to build a prototype of a Privacy-Conscious Pet Monitoring System.** In this iteration of the project, I focused on better optimizing the Deep Learning model back-ending the blur effect.

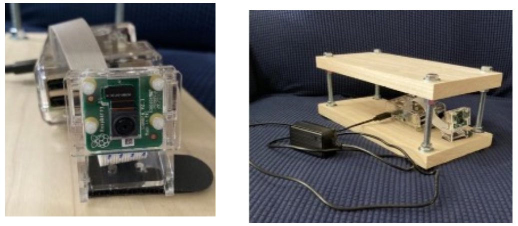
Using the **Oxford III-T Pet** dataset, I trained two variations of the **U-Net Convolutional Neural Network architecture for the task of Semantic Segmentation**: one with Attention Mechanisms and one without. Given an image, the networks output a pixel-wise classification (i.e. a mask) into the categories of *pet*, *boundary*, and *not pet*.

This project focuses on understanding and evaluating the performance of three popular optimizers for Deep Learning: **RMSProp, Adam, and Adagrad**.

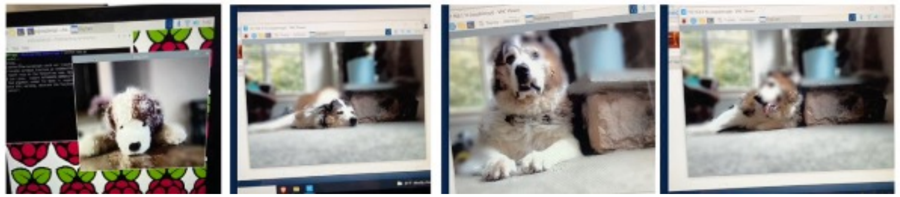
Previous Work

I completed the first iteration of this project in August 2021. I focused on hardware setup (using a Raspberry Pi 4 Model B and Arducam Raspberry Pi Official Camera Module V2), creating a rig out of wood, and exploring unsupervised techniques for Semantic Segmentation. Those methods (including edge detection and clustering) failed, so I quickly trained a U-Net CNN, which had mediocre results. I also reduced the dataset to the dogs.

I am revisiting the project given the recent popularity of Attention Mechanisms and Vision Transformers, and moreso to understand and explore the different optimization techniques. The data is now inclusive of both dogs and cats, increasing the generalizability of the model.



Camera Rig



Real-Time Results

Modeling

Optimization Model

Find the model parameters (i.e. the weights of the deep neural network) such that the **Categorical Cross-Entropy Loss** (cost function $\boldsymbol{J}(\boldsymbol{w})$ between the True ($\boldsymbol{y_i}$) and Predicted ($\boldsymbol{\hat{y}_i}$) pixel-wise probabilities is minimized:

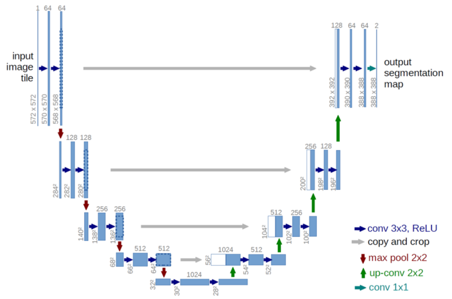
$$\min_{w} J(w) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where probabilities are configured with the **Softmax Activation Function**: $S(y_i) = \frac{e^{y_i}}{\sum e^{y_i}}$

- *N* is the number of categories.
- **Softmax** converts scores to a normalized probability distribution, which is differentiable and well-behaved; it is used to 'activate' nodes.
- In the models, this is specified as **Sparse Categorical Cross-Entropy Loss** due to the discretized format of the labels, i.e. [0], [1], or [2] instead of probability distributions – but the optimization processes itself considers it as probability distributions.
- The network weights are optimized by **backpropagating** the gradient of the loss.
- **Unconstrained problem**; many local minima and parameter configurations!

Proposed Solution (I)

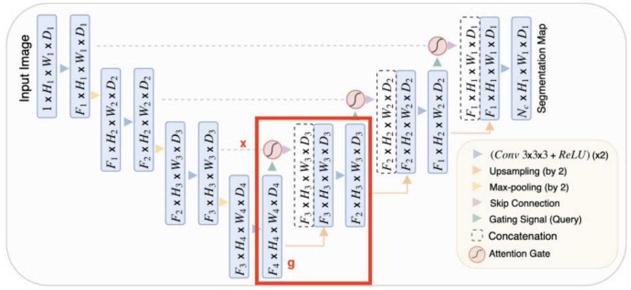
Based on its success in biomedical Semantic Segmentation tasks, I implemented the standard **U-Net Convolutional Neural Network** and a simplified **Attention U-Net Convolutional Neural Network**.



U-NET CNN

https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

- Builds on standard CNN architecture
- Contracting (downsampling) path to capture context
- Expanding (upsampling) path to localize precisely
- Propagates context of feature channels to higher-resolution layers (creates the U-shaped architecture)
- No fully-connected layers
- Effectively trained on smaller datasets



ATTENTION U-NET CNN

https://arxiv.org/pdf/1804.03999.pdf

- Uses U-Net CNN as backbone
- Contracting path maintained
- Attention mechanisms are added into the skip connections in Expanding path to put more weight on relevant features
- Attention highlights relevant activations during training
- Here, Soft Attention is used because it is differentiable and thus can be trained with backpropagation

Proposed Solution (II)

Optimizers are used to tune the parameters ***w*** of the networks to minimize the cost ***J(w)***. Algorithms such as Gradient Descent are standard but require manually tuning the learning rate, which can be tedious for training deep networks. Adaptive Optimizers build on Stochastic Gradient Descent to alleviate this burden. These help to get a stable numerical solution and are conveniently provided in the machine learning frameworks.

This problem is hypothetically **non-convex** given the weights, but could be convex.

I trained both architectures, compiling using three different Adaptive Optimizers:

RMSPROP	**ADAM**	**ADAGRAD**
$v_t^w = \beta * v_{t-1}^w + (1-\beta)(\nabla w_t)^2$ $w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$ $v_t^b = \beta * v_{t-1}^b + (1-\beta)(\nabla b_t)^2$ $b_{t+1} = b_t - \frac{\eta}{\sqrt{v_t^b + \epsilon}} * \nabla b_t$	$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t$ $v_t = \beta_2 * v_{t-1} + (1-\beta_2) * (\nabla w_t)^2$ $\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$ $w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$	$v_t^w = v_{t-1}^w + (\nabla w_t)^2$ $w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$ $v_t^b = v_{t-1}^b + (\nabla b_t)^2$ $b_{t+1} = b_t - \frac{\eta}{\sqrt{v_t^b + \epsilon}} * \nabla b_t$
• ‘Root Mean Square Propagation’ • Decays the denominator in addition to the numerator the learning rate coefficient to avoid learning rate decay issue	• ‘Adaptive Moment Estimation’ • Combines RMSProp and Adagrad intuition • Uses a cumulative history of gradients	• ‘Adaptive Gradient Algorithm’ • Decays learning rate for parameters proportionally to their update frequency • Converges fast with sparse features

Results

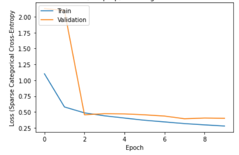
Optimization Behavior

U-Net CNN

RMSProp

```
Epoch 1/10
184/184 [==============================] - 35s 170ms/step - loss: 1.1013 - val_loss: 2.1294
Epoch 2/10
184/184 [==============================] - 29s 160ms/step - loss: 0.5787 - val_loss: 2.0969
Epoch 3/10
184/184 [==============================] - 29s 160ms/step - loss: 0.4858 - val_loss: 0.4554
Epoch 4/10
184/184 [==============================] - 29s 155ms/step - loss: 0.4395 - val_loss: 0.4728
Epoch 5/10
184/184 [==============================] - 29s 156ms/step - loss: 0.4038 - val_loss: 0.4693
Epoch 6/10
184/184 [==============================] - 29s 159ms/step - loss: 0.3693 - val_loss: 0.4549
Epoch 7/10
184/184 [==============================] - 29s 159ms/step - loss: 0.3432 - val_loss: 0.4355
Epoch 8/10
184/184 [==============================] - 29s 159ms/step - loss: 0.3168 - val_loss: 0.3929
Epoch 9/10
184/184 [==============================] - 29s 156ms/step - loss: 0.2966 - val_loss: 0.4041
Epoch 10/10
184/184 [==============================] - 29s 157ms/step - loss: 0.2786 - val_loss: 0.4003
```

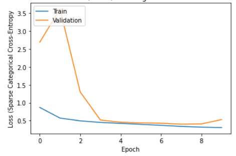
U-Net CNN (rmsprop): Training vs. Validation Loss



Adam

```
Epoch 1/10
184/184 [==============================] - 32s 161ms/step - loss: 0.8676 - val_loss: 2.6894
Epoch 2/10
184/184 [==============================] - 28s 154ms/step - loss: 0.5718 - val_loss: 3.6176
Epoch 3/10
184/184 [==============================] - 29s 159ms/step - loss: 0.4937 - val_loss: 1.3005
Epoch 4/10
184/184 [==============================] - 29s 157ms/step - loss: 0.4504 - val_loss: 0.5167
Epoch 5/10
184/184 [==============================] - 29s 157ms/step - loss: 0.4234 - val_loss: 0.4559
Epoch 6/10
184/184 [==============================] - 29s 158ms/step - loss: 0.3972 - val_loss: 0.4365
Epoch 7/10
184/184 [==============================] - 29s 157ms/step - loss: 0.3682 - val_loss: 0.4274
Epoch 8/10
184/184 [==============================] - 29s 157ms/step - loss: 0.3389 - val_loss: 0.4005
Epoch 9/10
184/184 [==============================] - 28s 153ms/step - loss: 0.3176 - val_loss: 0.4084
Epoch 10/10
184/184 [==============================] - 28s 153ms/step - loss: 0.3072 - val_loss: 0.5298
```

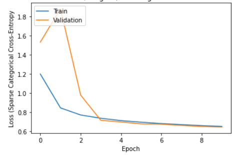
U-Net CNN (adam): Training vs. Validation Loss



Adagrad

```
Epoch 1/10
184/184 [==============================] - 52s 159ms/step - loss: 1.1999 - val_loss: 1.5317
Epoch 2/10
184/184 [==============================] - 28s 152ms/step - loss: 0.8463 - val_loss: 1.8795
Epoch 3/10
184/184 [==============================] - 29s 156ms/step - loss: 0.7722 - val_loss: 0.9800
Epoch 4/10
184/184 [==============================] - 29s 156ms/step - loss: 0.7380 - val_loss: 0.7170
Epoch 5/10
184/184 [==============================] - 29s 156ms/step - loss: 0.7134 - val_loss: 0.6992
Epoch 6/10
184/184 [==============================] - 29s 156ms/step - loss: 0.6971 - val_loss: 0.6797
Epoch 7/10
184/184 [==============================] - 29s 156ms/step - loss: 0.6824 - val_loss: 0.6762
Epoch 8/10
184/184 [==============================] - 29s 156ms/step - loss: 0.6720 - val_loss: 0.6632
Epoch 9/10
184/184 [==============================] - 29s 157ms/step - loss: 0.6610 - val_loss: 0.6526
Epoch 10/10
184/184 [==============================] - 29s 156ms/step - loss: 0.6533 - val_loss: 0.6464
```

U-Net CNN (adagrad): Training vs. Validation Loss

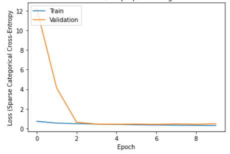


Attention U-Net CNN

RMSProp

```
Epoch 1/10
184/184 [==============================] - 62s 294ms/step - loss: 0.7198 - val_loss: 12.2151
Epoch 2/10
184/184 [==============================] - 54s 293ms/step - loss: 0.5365 - val_loss: 4.1177
Epoch 3/10
184/184 [==============================] - 54s 292ms/step - loss: 0.4793 - val_loss: 0.6292
Epoch 4/10
184/184 [==============================] - 54s 292ms/step - loss: 0.4376 - val_loss: 0.4347
Epoch 5/10
184/184 [==============================] - 54s 292ms/step - loss: 0.4044 - val_loss: 0.4230
Epoch 6/10
184/184 [==============================] - 52s 285ms/step - loss: 0.3723 - val_loss: 0.4364
Epoch 7/10
184/184 [==============================] - 54s 292ms/step - loss: 0.3485 - val_loss: 0.4077
Epoch 8/10
184/184 [==============================] - 52s 285ms/step - loss: 0.3286 - val_loss: 0.4572
Epoch 9/10
184/184 [==============================] - 52s 285ms/step - loss: 0.3129 - val_loss: 0.4262
Epoch 10/10
184/184 [==============================] - 53s 286ms/step - loss: 0.3003 - val_loss: 0.4797
```

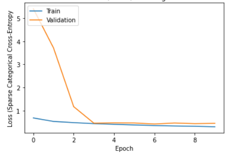
U-Net CNN with Attention (rmsprop): Training vs. Validation Loss



Adam

```
Epoch 1/10
184/184 [==============================] - 58s 291ms/step - loss: 0.6827 - val_loss: 5.4073
Epoch 2/10
184/184 [==============================] - 53s 290ms/step - loss: 0.5315 - val_loss: 3.7113
Epoch 3/10
184/184 [==============================] - 53s 290ms/step - loss: 0.4809 - val_loss: 1.1711
Epoch 4/10
184/184 [==============================] - 53s 287ms/step - loss: 0.4359 - val_loss: 0.4573
Epoch 5/10
184/184 [==============================] - 51s 278ms/step - loss: 0.4075 - val_loss: 0.4706
Epoch 6/10
184/184 [==============================] - 52s 280ms/step - loss: 0.3802 - val_loss: 0.4658
Epoch 7/10
184/184 [==============================] - 53s 290ms/step - loss: 0.3546 - val_loss: 0.4199
Epoch 8/10
184/184 [==============================] - 52s 283ms/step - loss: 0.3357 - val_loss: 0.4640
Epoch 9/10
184/184 [==============================] - 52s 282ms/step - loss: 0.3221 - val_loss: 0.4364
Epoch 10/10
184/184 [==============================] - 52s 281ms/step - loss: 0.3005 - val_loss: 0.4489
```

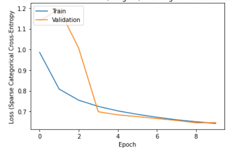
U-Net CNN with Attention (adam): Training vs. Validation Loss



Adagrad

```
Epoch 1/10
184/184 [==============================] - 55s 275ms/step - loss: 0.9861 - val_loss: 1.1370
Epoch 2/10
184/184 [==============================] - 50s 273ms/step - loss: 0.8084 - val_loss: 1.1961
Epoch 3/10
184/184 [==============================] - 52s 284ms/step - loss: 0.7547 - val_loss: 1.0056
Epoch 4/10
184/184 [==============================] - 53s 286ms/step - loss: 0.7248 - val_loss: 0.6989
Epoch 5/10
184/184 [==============================] - 53s 288ms/step - loss: 0.7031 - val_loss: 0.6838
Epoch 6/10
184/184 [==============================] - 53s 287ms/step - loss: 0.6864 - val_loss: 0.6755
Epoch 7/10
184/184 [==============================] - 53s 288ms/step - loss: 0.6723 - val_loss: 0.6663
Epoch 8/10
184/184 [==============================] - 53s 289ms/step - loss: 0.6602 - val_loss: 0.6570
Epoch 9/10
184/184 [==============================] - 53s 288ms/step - loss: 0.6507 - val_loss: 0.6455
Epoch 10/10
184/184 [==============================] - 52s 281ms/step - loss: 0.6423 - val_loss: 0.6463
```

U-Net CNN with Attention (adagrad): Training vs. Validation Loss

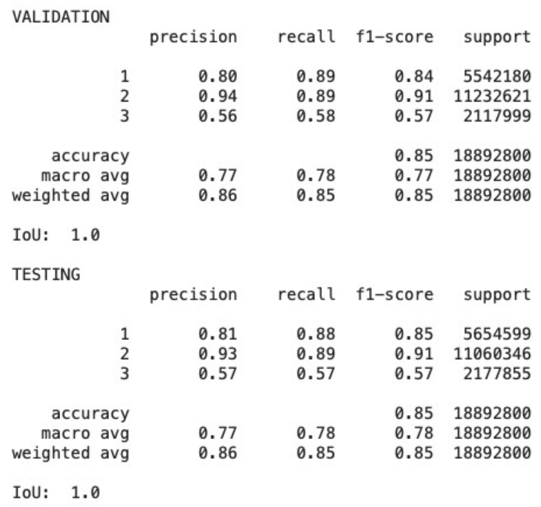


Validation (I)

I evaluated each optimizer's loss trajectory over the course of the epochs, both for training and validation data. Based on final loss and stability, I selected the **RMSProp-optimized model for the U-Net CNN** and the **Adam-optimized model for the Attention U-Net CNN.**

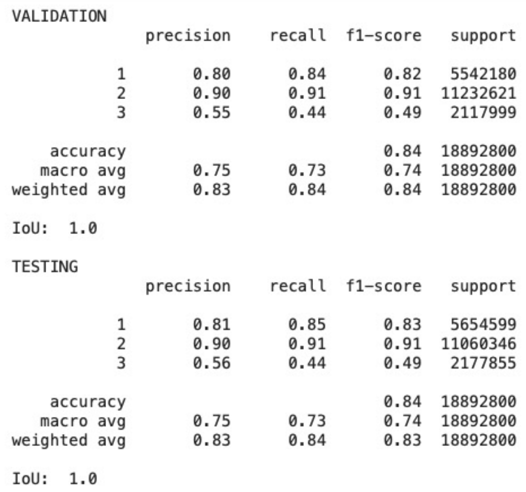
For each model, I computed the class-wise **Precision, Recall, and F1-Scores**, as well as **Intersection over Union** (which was not very informative). These metrics are standard for classification tasks, and the latter especially for evaluating segmentation. I focused on F1-Score, as it is case-dependent whether to prioritize Precision or Recall.

U-Net CNN



```
VALIDATION
              precision    recall  f1-score   support

           1       0.80      0.89      0.84   5542180
           2       0.94      0.89      0.91  11232621
           3       0.56      0.58      0.57   2117999

    accuracy                           0.85  18892800
   macro avg       0.77      0.78      0.77  18892800
weighted avg       0.86      0.85      0.85  18892800

IoU:  1.0

TESTING
              precision    recall  f1-score   support

           1       0.81      0.88      0.85   5654599
           2       0.93      0.89      0.91  11060346
           3       0.57      0.57      0.57   2177855

    accuracy                           0.85  18892800
   macro avg       0.77      0.78      0.78  18892800
weighted avg       0.86      0.85      0.85  18892800

IoU:  1.0
```

Attention U-Net CNN

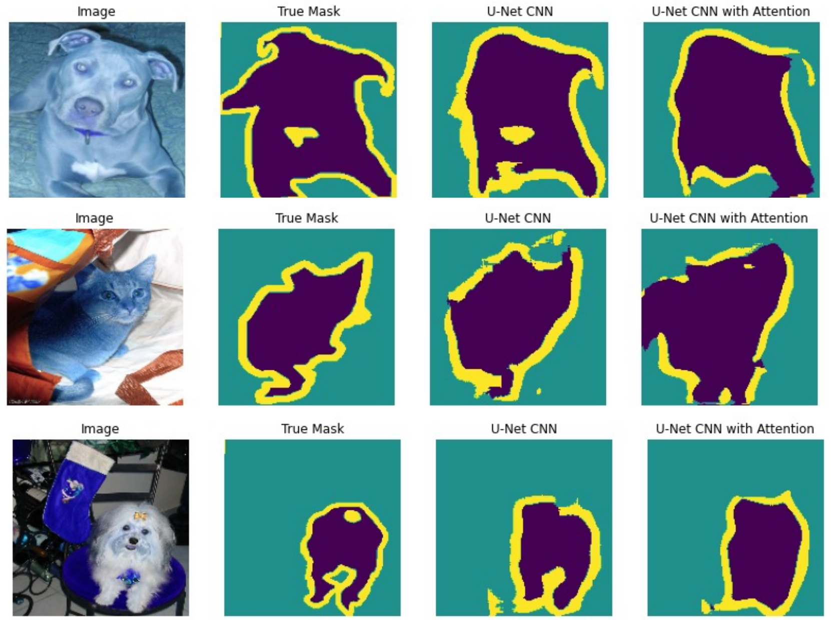


```
VALIDATION
              precision    recall  f1-score   support

           1       0.80      0.84      0.82   5542180
           2       0.90      0.91      0.91  11232621
           3       0.55      0.44      0.49   2117999

    accuracy                           0.84  18892800
   macro avg       0.75      0.73      0.74  18892800
weighted avg       0.83      0.84      0.84  18892800

IoU:  1.0

TESTING
              precision    recall  f1-score   support

           1       0.81      0.85      0.83   5654599
           2       0.90      0.91      0.91  11060346
           3       0.56      0.44      0.49   2177855

    accuracy                           0.84  18892800
   macro avg       0.75      0.73      0.74  18892800
weighted avg       0.83      0.84      0.83  18892800

IoU:  1.0
```

Validation (II)

Because this project is a prototype for a product, perhaps the most important test is **human-evaluation**. In this step, I selected a few images from the training and validation sets and compared their predicted masks to the ground truth.

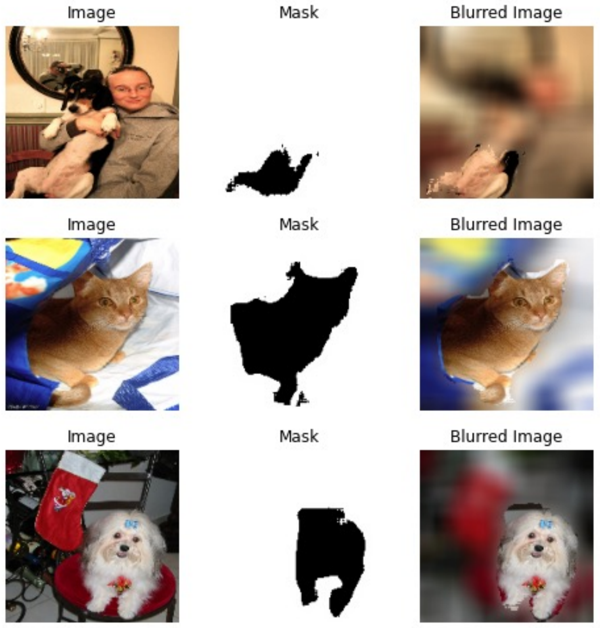
Here, I prefer the output of the Attention U-Net CNN. Because the *pet* (purple) and *boundary* (yellow) pixels are combined in the post-processing steps, I especially feel that the Attention U-Net has a cleaner mask.



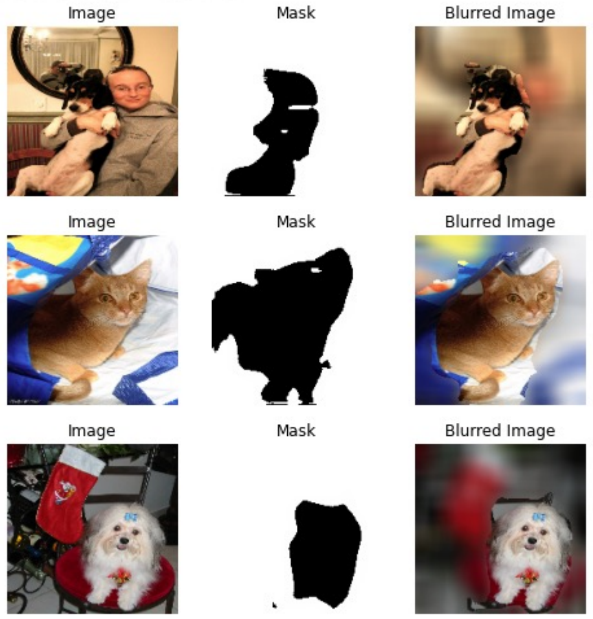
Simulation (I)

In order to evaluate the blurring effect, I incorporated the mask prediction step into a quick image-processing pipeline. The blurring is done using a large-Sigma Gaussian Filter convolved over a copy of the image, and the corresponding pixels classified as *not-pet* (white) from the blurred copy replace those in the original image. I first applied the pipeline to the previous dataset images:

U-Net CNN



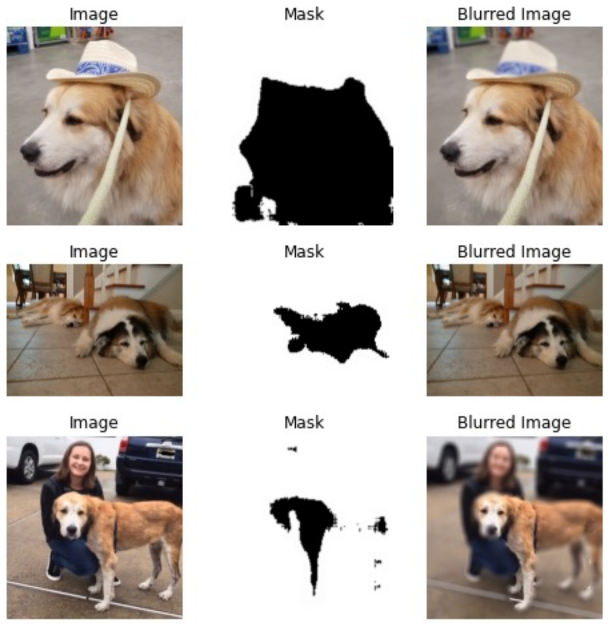
Attention U-Net CNN



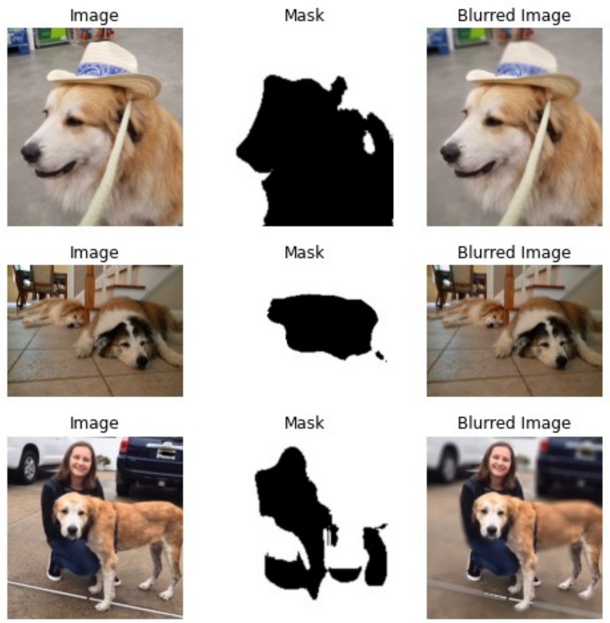
Simulation (II)

Then, I applied the pipeline to images of my own dogs. Here, it appears that the U-Net CNN performs better in the context of privatizing the images, although the Attention U-Net CNN has a much smoother result, which I feel is important for output aesthetic.

U-Net CNN



Attention U-Net CNN



Conclusions

Key Insights

- When comparing results to the previous iteration of this project, **focusing on the backend optimizer and network architecture was beneficial.**

- **Adam appears to be the 'go-to' optimizer** for deep neural networks, as supported in many papers.

- **Attention Mechanisms, although complicated to incorporate, are very informative to the network.** In this case, they seemed to improve the smoothness of the results, although did cause some accuracy concerns in terms of detecting humans if near the pet. This is expected behavior based on the nature of Attention Mechanisms.

Major Conclusions

- **My approach was viable**, mostly due to the capabilities of the Keras machine learning framework. However, because this is very black-boxed, later adaptions of the network may require TensorFlow.

- Because I was training deep networks, I was forced to use a GPU, but even still **I faced RAM issues with the Attention U-Net**. This causes me some concern if I were to deploy this to the Raspberry Pi; I may require a better GPU.

- I expected most of the results, and was **especially satisfied with the Attention U-Net**. The previous iteration of this project had issues with not completely capturing the pet, so I can see how incorporating Attention has helped to at least smooth the target region.

Improvements

- Once I can alleviate the runtime issues, I plan to deploy the Attention U-Net CNN to Raspberry Pi and do real-time tests. That will allow me to evaluate the prediction latency and other behavior of the model.

- A lot of the issues with the output may be resolved with more image-processing. For instance, I feel that applying some morphological closing may help to get rid of the sharp regions of the masks.

- I have been researching a lot about the optimization of Vision Transformers. Although they are currently better-suited for Instance-Based Segmentation, I hope to find a way to incorporate them, at least for research purposes.

References

https://neptune.ai/blog/cross-entropy-loss-and-its-applications-in-deep-learning

https://towardsdatascience.com/learning-parameters-part-5-65a2f3583f7d

https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

https://www.v7labs.com/blog/semantic-segmentation-guide

https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

https://sh-tsang.medium.com/review-attention-u-net-learning-where-to-look-for-the-pancreas-biomedical-image-segmentation-e5f4699daf9f

https://towardsdatascience.com/using-attention-for-medical-image-segmentation-dd78825eaac6