APS360 - *ExtractNet* Final Report

Mohsin Hasan
Nahian Khan
Sagar Patel

April 11, 2019

Word Count: 1780

1 Goal and Motivation

ExtractNet is a software tool which removes backgrounds from images that have prominent objects belonging to the following classes: people and vehicles. This is an application of image segmentation, which is the clustering of pixels into salient regions [1]. The segmentation of images into separate regions allows for the extraction of prominent objects.

(a) With background. (b) Without background.

Figure 1: Example of background removal. Sample image from [2].

There are several motives for background removal. For example, the media industry can use background removal to refine publication images, and the autonomous vehicle industry can use it to enhance machine vision. The rapid growth [3][4] of both of these industries highlights the importance of the task.

Normally, one would have to use tools such as Adobe Photoshop to extract backgrounds manually [5]. However, this process can be time consuming, requires expertise, and cannot be done in real-time. Machine learning is therefore an excellent tool for automating the task. Given enough data, machine learning approaches have been shown to outperform traditional image processing algorithms for similar segmentation tasks [6].

2 Overall Software Structure

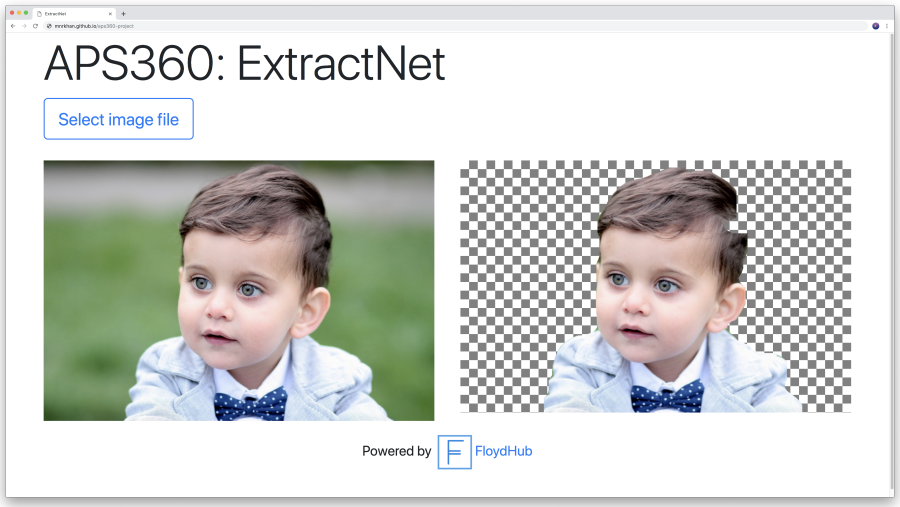


Figure 2: Example of the ExtractNet user interface. Sample image from [7].

The software allows users to upload arbitrarily sized images onto a Flask server hosted on FloydHub. The image dimensions are checked and the image is possibly scaled (so that dimensions divide evenly for downsampling steps) in a lightweight pre-processing stage before being passed into the model for inference. A forward pass through the trained model generates a mask indicating pixels of the image belonging to the prominent object.

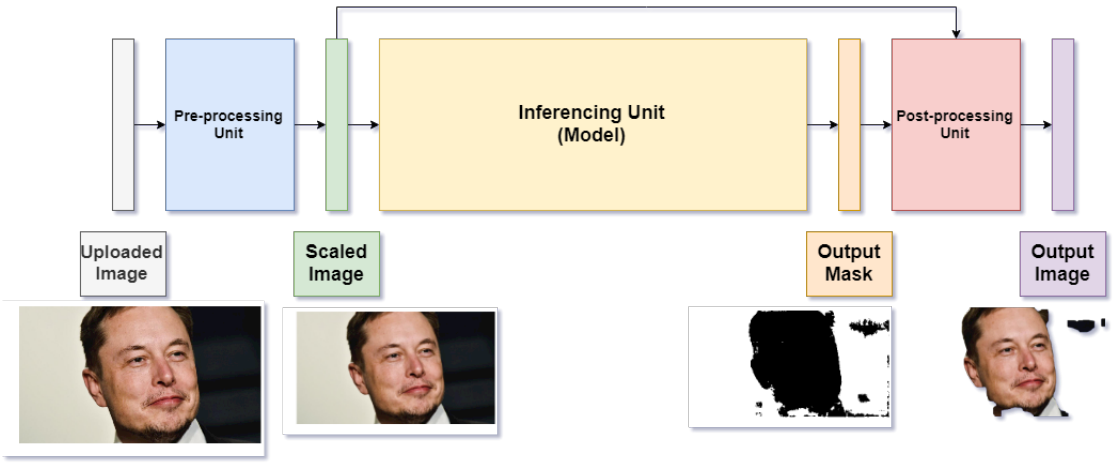


Figure 3: ExtractNet software pipeline. Sample image from [8].

The generated mask is fine-tuned using the following image processing techniques (Figure 4)(Figure 5 shows an example).

- Gaussian Blur ($kernelsize = 27$): This smooths rough edges and fills in small holes within the prominent object. The large kernel size allows for larger areas to be considered at once for improved hole-filling.
- Simple Thresholding ($threshold = 0.5$): The Gaussian blur above results in pixel values between 0 and 1 (non-binary). This applies a threshold of 0.5 to re-obtain a binary mask (each decimal is rounded to 0 or 1).
- Morphological Opening ($kernelsize = 39$): This reduces noise in the mask [9].

Finally, the mask is applied to the scaled image to generate the final output.

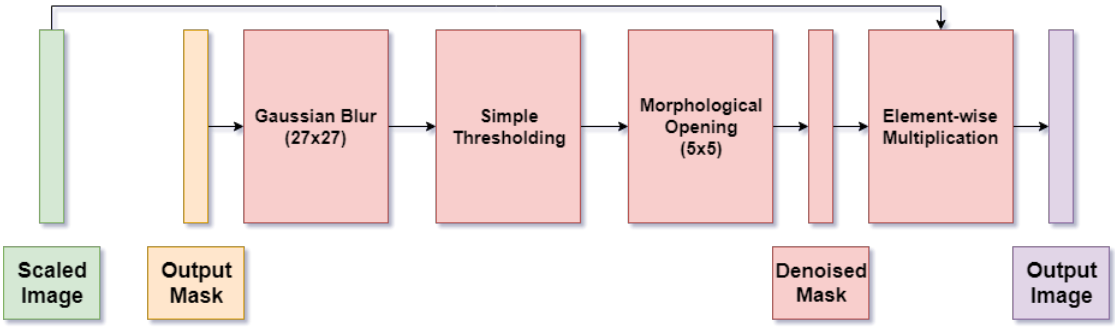


Figure 4: Post-processing unit denoiser.

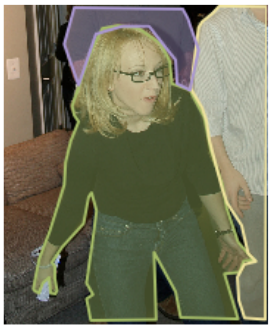
(a) Before post-processing. (b) After post-processing.

Figure 5: Output mask before and after post-processing.

3 Sources of Data

The "COCO Train 2014" dataset was used to train the model [10][11]. Many images in the dataset were not directly usable for the task as they had objects which were too small, or too large (leaving little to no background). The default annotation format also was not appropriate for the task as each object instance had its own mask. These issues were resolved by the following data cleaning steps:

1. Filter images to those containing the classes: 'people' and 'vehicle'.
2. Extract masks corresponding to "prominent objects" (masks belonging to the classes above which occupy at least 10% of the image):
    - We only consider images with at least one "major" prominent object: a mask occupying at least 30% of the image. Most of such images are more centralized and focused.
    - For such images, we include the masks of up to 3 more prominent objects.
3. Combine the individual masks above to form a single binary mask for the image.
4. If the mask occupies more than 90% of the image, we reject it: since images with a sizable background are preferred for training.
5. Resize images to 224×224 (which may change the aspect ratio): this aids in training the model, by allowing for batching, and reducing the computational time needed for forward passes.



(a) Original Annotations.

(b) Generated Mask.

Figure 6: Example of generating the final (resized) mask from the base COCO annotations.

This process resulted in a set of around 10,000 images and corresponding masks. A custom `DataSet` class was implemented to handle the new data format, which can interface with PyTorch's `DataLoader`.

4 Models

The neural network in the software is responsible for producing pixel-wise predictions to generate the prominent object mask given the image. The general model architecture used to achieve this task consists of a fully convolutional, encoder-decoder pair, as presented in Figure 7.

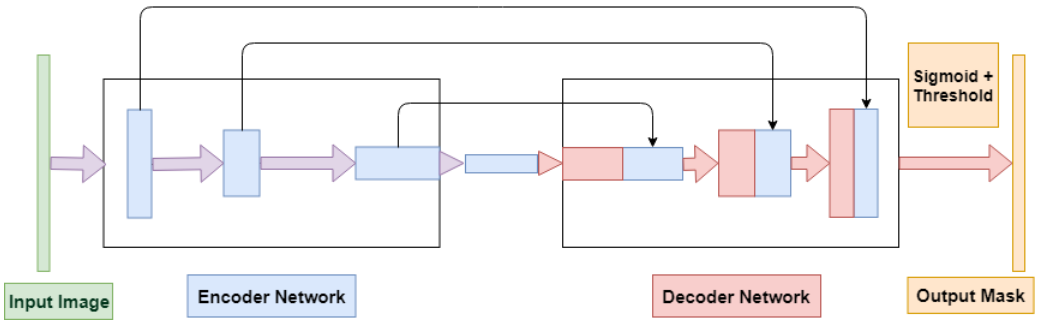


Figure 7: General Model Architecture.

The encoder uses convolution and downsampling layers to distill the image into lower dimensional features, while the decoder uses deconvolution (also referred

to as transposed convolution) layers to upsample these in a mirrored fashion. The result is a single channel output of the same size as the input image. The output is generated using a sigmoid activation, so that the value at each pixel can be interpreted as the probability that the pixel is part of a prominent object. The mask is generated by thresholding these probabilities.

There are links connecting the encoder and decoder, which are inspired from U-net, a popular architecture for segmentation [12]. A link directly forwards and concatenates the activations of an intermediate encoder layer to its mirrored decoder layer. The concatenated activations are fed together to the next decoder layer. These interconnections allow for the preservation of high-resolution details, allowing for a sharper mask.

4.1 Model Variations

The general architecture above was used as the base for the following implemented models:

4.1.1 Simple Autoencoder

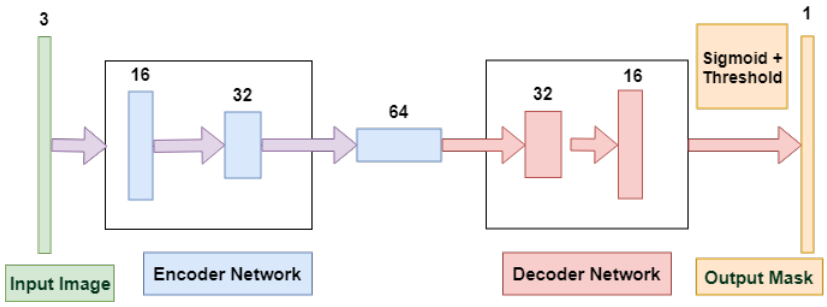


Figure 8: Simple autoencoder model. The channel depths are denoted on top of each input and output.

The first implemented model was a simpler version of the general architecture: lacking the interconnections. It consisted of an encoder and decoder (3 layers each), with ReLU activations. Each layer has a kernel size of 3 and stride of 2 (so that the encoder halves the dimensions, while the decoder doubles them).

4.1.2 Interconnected Autoencoder

The second model is identical to the first, except that it includes the aforementioned interconnects between the encoder and decoder.

4.1.3 Pretrained Encoder Models

The pretrained encoder models benefit from transfer learning by using an encoder with frozen weights pretrained on ImageNet. The encoders used include VGG-11, VGG-19, ResNet-50, and ResNet-152.

These encoders primarily consist of functional blocks, each of which contain consecutive padded convolutions (such that the dimensions are unchanged), followed by a single downsampling layer. Since these encoders are fairly deep, a fully mirrored decoder is not tractable. Instead the implemented decoder contains only upsampling layers for each downsampling layer at the encoder. The upsampling at the decoder is performed by a deconvolution layer with a kernel size of 3.

This lead to decoders which were 6 layers deep for VGG encoders, and 7 layers deep for ResNet encoders.

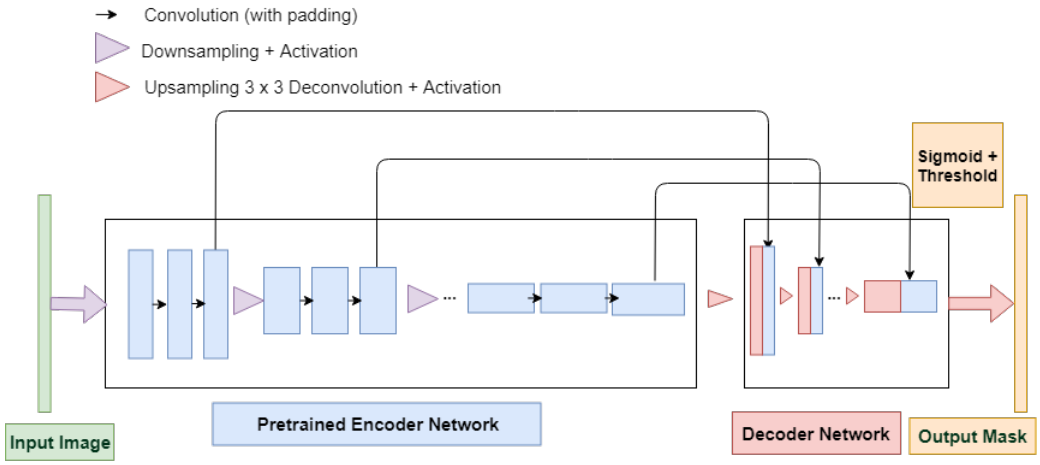


Figure 9: Model with pretrained encoder

4.2 Baseline Model

To judge model performance, a simplistic baseline was implemented which relied on traditional computer vision methods for segmentation.

Inspired by [13], the model used traditional image processing algorithms to segment the original image:

- Convert original RGB image to grayscale
- Apply Otsu thresholding to binarize between foreground and background
- Identify "connected" (clustered) regions in the image

- Apply watershed segmentation

Next, each of the segment masks were multiplied by a center-prioritized weight matrix in order to identify the central segment as the most "prominent" segment in the image (and extract a single mask).

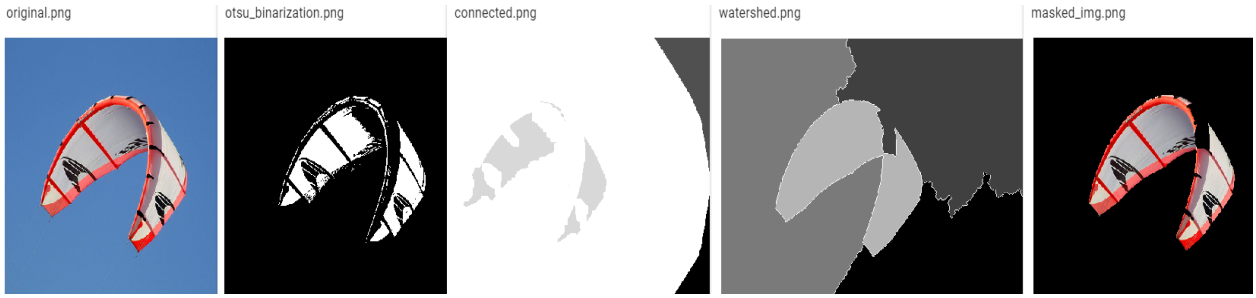


Figure 10: The steps used in the baseline model. Sample image from [11].

5 Training, Validation and Test

The primary hyperparameters decided via training included:

- Which model variant to use
- The inclusion of batch normalization layers (inspired by their potential to speed up training)
- The choice of activation function at the decoder, between: ReLU, and parametric ReLU (PReLU). (Inspired by the added flexibility of PReLU with its trainable parameter).
- The learning rate
- The batch size

5.1 Training Setup

Training was done using the loss function of binary cross entropy loss, since the task involves the binary classification of each pixel (to determine whether or not it is part of a prominent object). This loss was minimized using the Adam optimizer.

The metric used to evaluate model performance was Intersection-over-Union (IoU). Given the model's generated mask, and the true mask, the IoU is calculated, as the area of the intersection divided by the area of the union for the two masks. Therefore, a higher IoU corresponds to more similar masks, with an

IoU of 1 corresponding to equivalent masks. For the purposes of training the model, none of the aforementioned post-processing is applied. This is to get a more accurate characterization of the performance of the network alone.

5.2 Training Results

In order to decide hyperparameter settings in an efficient way, a set of 1000 images were set aside from the overall dataset to use as the train and validation set for tuning.

On this smaller set, each model was trained for a total of 80 epochs. Some initial tuning showed that a learning rate of 0.001, and a batch size of 64 worked well over the tested models. The resulting IoU and loss for each trained model is tabulated below.

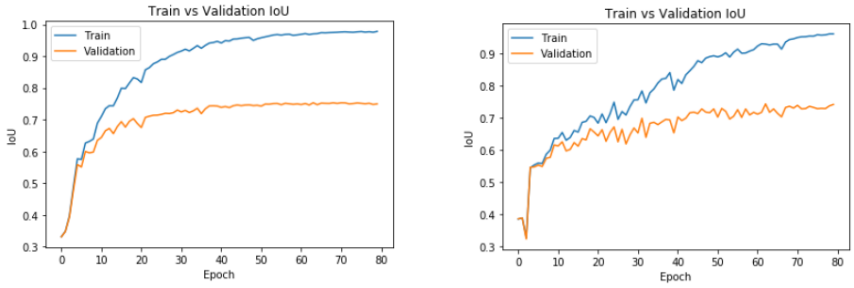
Model	**Training IoU**	**Validation IoU**
Baseline	-	0.379
Simple Autoencoder	0.431	0.449
Interconnected Autoencoder	0.424	0.430
VGG-11 Encoder (R)	0.978	0.683
VGG-11 Encoder (BN)	0.979	0.694
VGG-11 Encoder (BN), (P)	0.977	0.702
VGG-19 Encoder (R)	0.971	0.710
VGG-19 Encoder (BN)	0.973	0.710
VGG-19 Encoder (BN), (P)	0.968	0.724
ResNet-50 Encoder (R), (BN)	**0.978**	**0.750**
ResNet-50 Encoder (P), (BN)	0.959	0.741
ResNet-152 Encoder (R), (BN)	0.974	0.745
ResNet-152 Encoder (P), (BN)	0.0.971	0.742

Table 1: Performance of different models. (R) denotes ReLU activations, (P) denotes PReLU, and (BN) denotes the inclusion of batch normalization.

A few key observations from this are that:

- All neural network models outperformed the baseline.
- Batch normalization slightly boosted performance for some models, however, its primary effect was to speed up the training: convergence was reached up to 40 epochs earlier in some cases. This made batch normalization highly preferable for training.

- PReLU had the effect of making the training less stable, as can be seen in Figure 11. Specifically, the training curve tended to oscillate more, and converged to a slightly lower IoU than with ReLU. This made it less preferable compared to the regular ReLU activation.

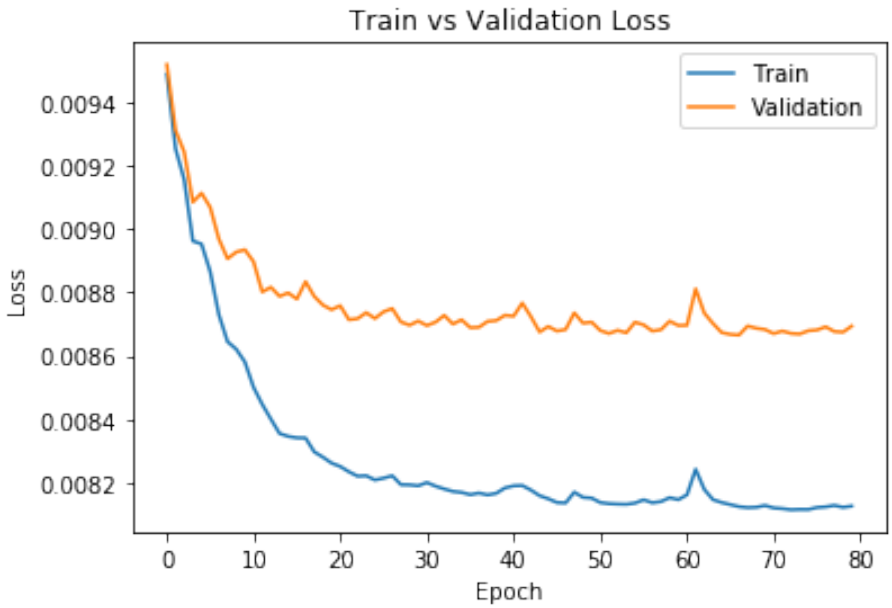


(a) ResNet-50 with ReLU

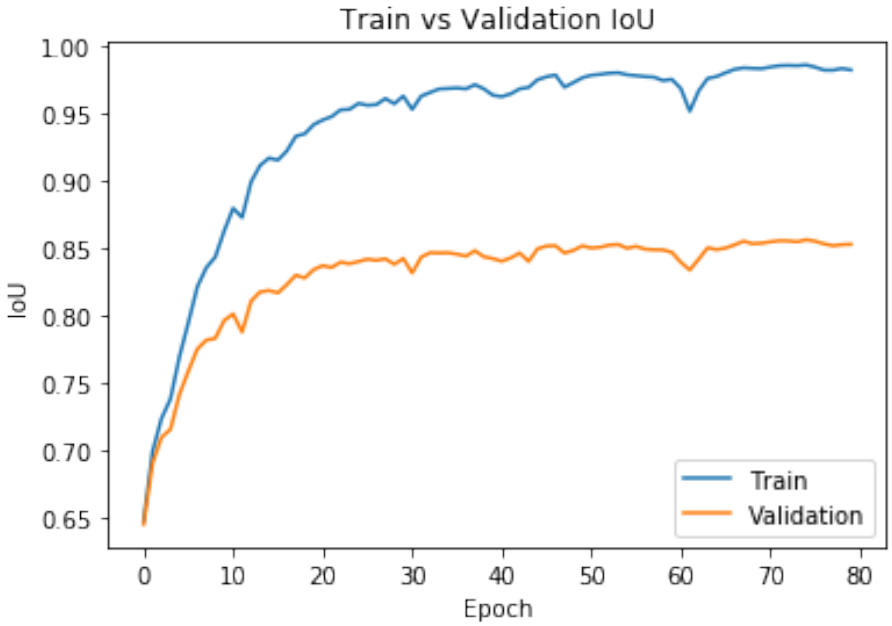
(b) ResNet-50 with PReLU

Figure 11: The effect of PReLU, on the ResNet-50 pretrained network

The best model among these is the one using a pretrained ResNet-50 encoder. This model was trained on the full dataset (using a 60/20/20 split for training, validation and test). The training curves for this model are shown below:



(a) ResNet model loss curves.



(b) ResNet model IoU curves.

Figure 12: Training curves for the final model.

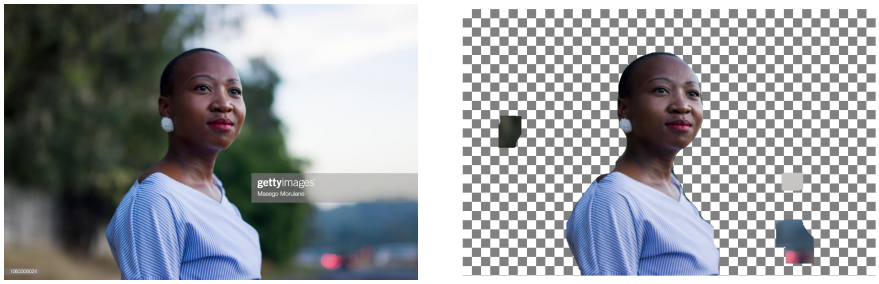
The final performance of this model is listed in the table below:

Model	Training IoU	Validation IoU	Test IoU
ResNet-50 Encoder Model	0.982	0.856	0.842

For comparison, many similar segmentation tasks have a state of the art of around 0.89 IoU [14].

6 Ethical Issues

The primary ethical concern of our project is intellectual property. Several images contain logos, watermarks and credits pertaining to the image owner. Our software has shown to remove these items, and ultimately violate fair usage of the image. Malicious users may even misuse the software to automate the removal of these items using our software.



(a) With background and watermark. (b) Without background and watermark.

Figure 13: The watermark is removed along with the background. Sample image from [15].

7 Key Learnings

Throughout the development of the software, several key learnings revolved around improving computational time since efficient training was critical to effectively train on large datasets. Initially, models were trained on Google Colab using free GPUs. However, as the network size increased, Colab was insufficient, and training required the use of Google Cloud with more powerful dedicated GPUs.

In addition to hardware acceleration, batch normalization was used in larger networks to reduce overall computation time. Batch normalization allowed the models to converge to similar losses faster, which allowed for quicker tuning

decisions. Moreover, transfer learning proved to drastically improve the training time, since it allowed for the use of pretrained, frozen encoder weights, rather than having to train the encoder weights.

Future improvements include using even better hardware to train on a larger dataset and/or additional classes of objects.

In addition, after analyzing the results of the final model, it became evident that results which under-crop the prominent object are preferred over results with over-cropping. In the former case, additional software can be used to re-crop and salvage the masked image, whereas this is not possible in the latter case (as in Figure 14). A future improvement would take this into account by penalizing false negative pixel classifications more than false positives.

(a) With background. (b) Without background and parts of body.

Figure 14: Although the background has been removed, parts of the prominent object (body) have also been removed. Sample image from [16].

This can be achieved by integrating an additional penalty term for false negatives into the loss function directly when training, or simply reducing the hole-filling threshold in the post-processing stage.

References

[1] A. Jepson and D. Fleet. (2007). Image segmentation, [Online]. Available: http://www.cs.toronto.edu/~jepson/csc2503/segmentation.pdf. (accessed: 02.18.2019).

[2] (2018). Model x — tesla, [Online]. Available: https://www.tesla.com/modelx. (accessed: 04.11.2019).

[3] A. Smith and M. Anderson. (Mar. 2018). Social media use in 2018, [Online]. Available: http://www.pewinternet.org/2018/03/01/social-media-use-in-2018/. (accessed: 02.23.2019).

[4] (Dec. 2018). The global autonomous vehicle market is expected to reach $30 billion by 2023: Key analyses forecasts, [Online]. Available: https://www.businesswire.com/news/home/20181227005139/en/Global-Autonomous-Vehicle-Market-Expected-Reach-30. (accessed: 02.23.2019).

[5] (2013). Removing a background from an image in photoshop, [Online]. Available: http://www.microknowledge.com/remove-background-photoshop/. (accessed: 02.24.2019).

[6] J. C. Caicedo, J. Roth, A. Goodman, T. Becker, K. W. Karhohs, M. Broisin, M. Csaba, C. McQuin, S. Singh, F. Theis, and A. E. Carpenter, "Evaluation of deep learning strategies for nucleus segmentation in fluorescence images," *bioRxiv*, 2019. DOI: 10.1101/335216. eprint: https://www.biorxiv.org/content/early/2019/02/06/335216.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2019/02/06/335216.

[7] (Feb. 2017), [Online]. Available: https://c.pxhere.com/photos/15/ab/boy_toddler_green_eyes_portrait_nice-617351.jpg!d. (accessed: 04.11.2019).

[8] (Mar. 2019). Biography.com, [Online]. Available: https://www.biography.com/people/elon-musk-20837159. (accessed: 03.21.2019).

[9] (). Morphological transformations, [Online]. Available: https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html. (accessed: 03.12.2019).

[10] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[11] (2018). Coco common objects in context, [Online]. Available: http://cocodataset.org/#home.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. arXiv: 1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597.

[13] (). Image segmentation with watershed algorithm, [Online]. Available: https://docs.opencv.org/3.4.3/d3/db4/tutorial_py_watershed.html. (accessed: 03.10.2019).

[14] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *CoRR*, vol. abs/1802.02611, 2018. arXiv: 1802.02611. [Online]. Available: http://arxiv.org/abs/1802.02611.

[15] M. Morulane. (). Woman waiting for a taxi, [Online]. Available: https://media.gettyimages.com/photos/woman-waiting-for-a-taxi-picture-id1063308024. (accessed: 04.11.2019).

[16] (Apr. 2017), [Online]. Available: https://c.pxhere.com/photos/e4/4b/kid_happy_girl_childhood_smile_portrait_joy-1375820.jpg!d. (accessed: 04.11.2019).