UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

SCS_3546_007 Deep Learning - Prof. Sina Jamshidi - April 2019

Exploration of Semantic Segmentation Applications for Packaging and E-Commerce Content

Bruno Schrappe

Problem Definition

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Consumer packaged products have image retouching requirements such as background removal that are usually delivered by human operators, as seen on examples on the right.

- Imagery meant to be distributed to e-commerce sites usually have their background removed or set to white, whereas packaging artwork files require images with a blended background and may be difficult to properly mask.

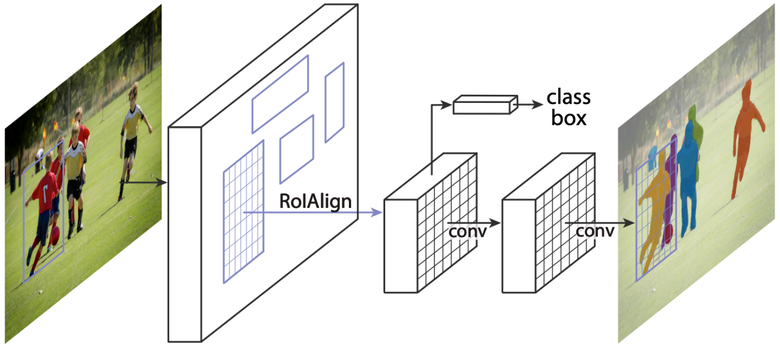
- This project investigates Deep Learning options using semantic and instance segmentation models to automate the process at comparable levels of quality.

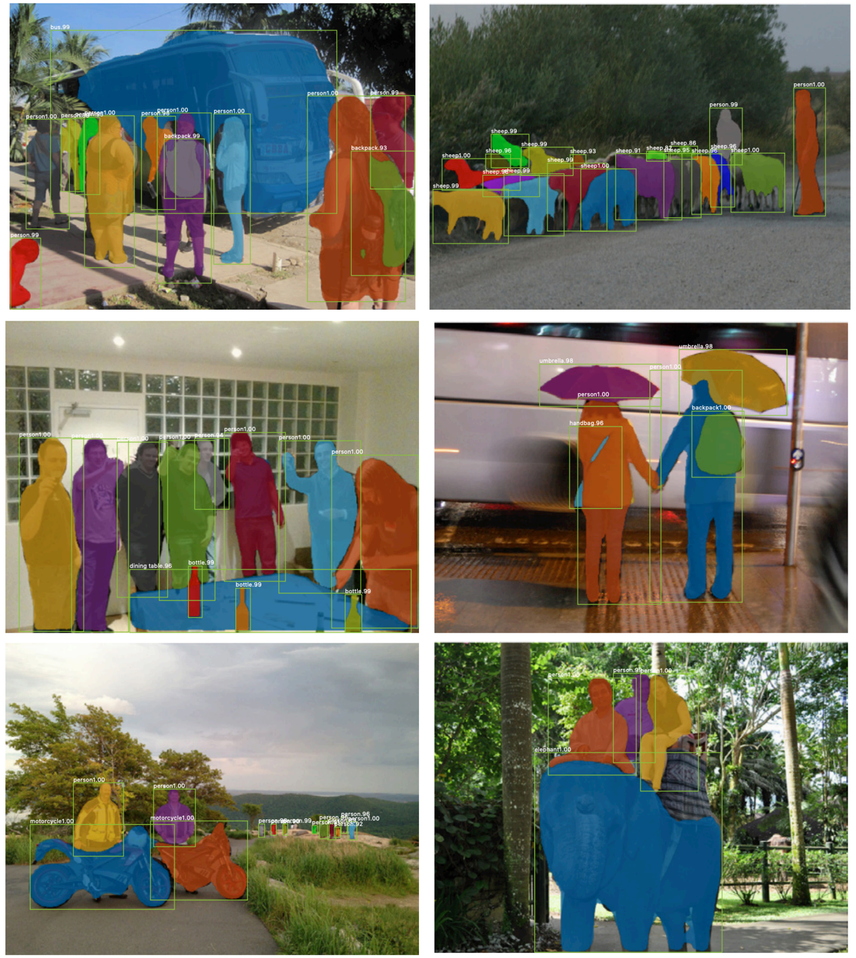
Semantic Image Segmentation

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Faster R-CNN predicts object classes and bounding boxes for object recognition. Mask R-CNN is an instance segmentation extension to Faster R-CNN that predicts object masks for every instance.

- We used a Mask R-CNN model previously trained on the COCO image dataset as an initial reference for this project.





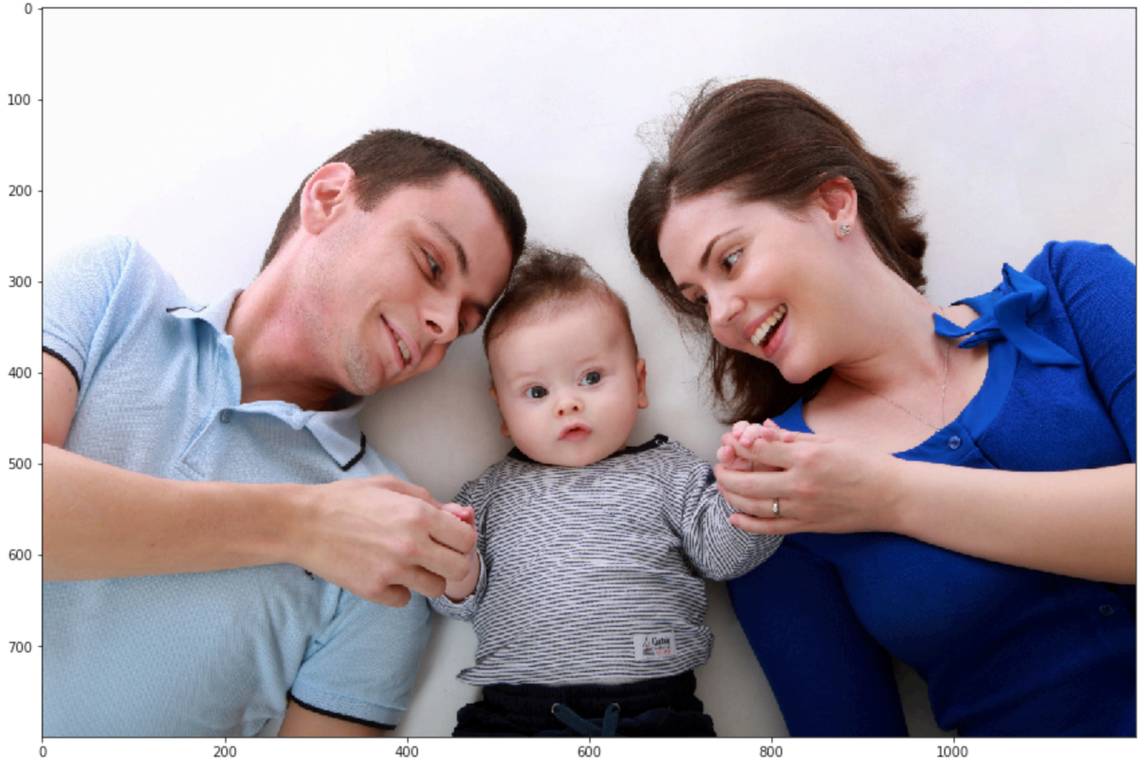
Source: Mask R-CNN
Kaiming He Georgia Gkioxari Piotr Dollar Ross Girshick
Facebook AI Research (FAIR)

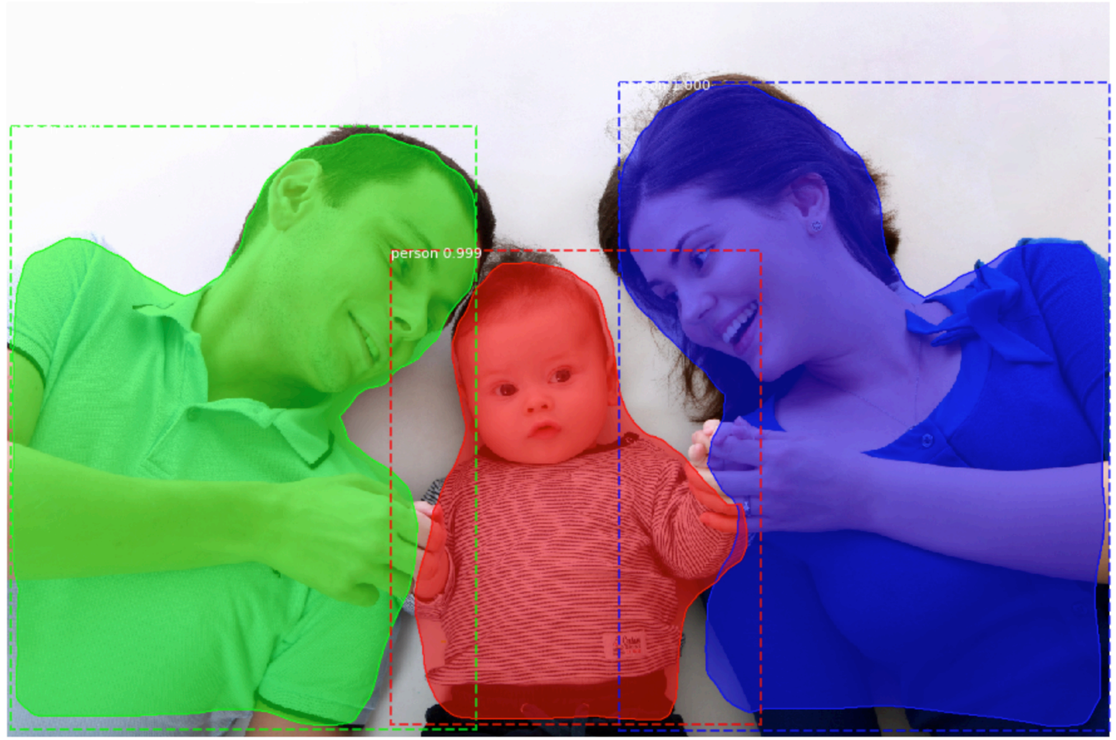
Instance Segmentation With Mask R-CNN

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Mask R-CNN performs object detection and instance segmentation, but segmentation mask accuracy is not acceptable for image retouching purposes.

```
image                    shape: (800, 1200, 3)        min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 93)               min:    0.00000  max: 1200.00000  float64
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:    1.29134  float32
```



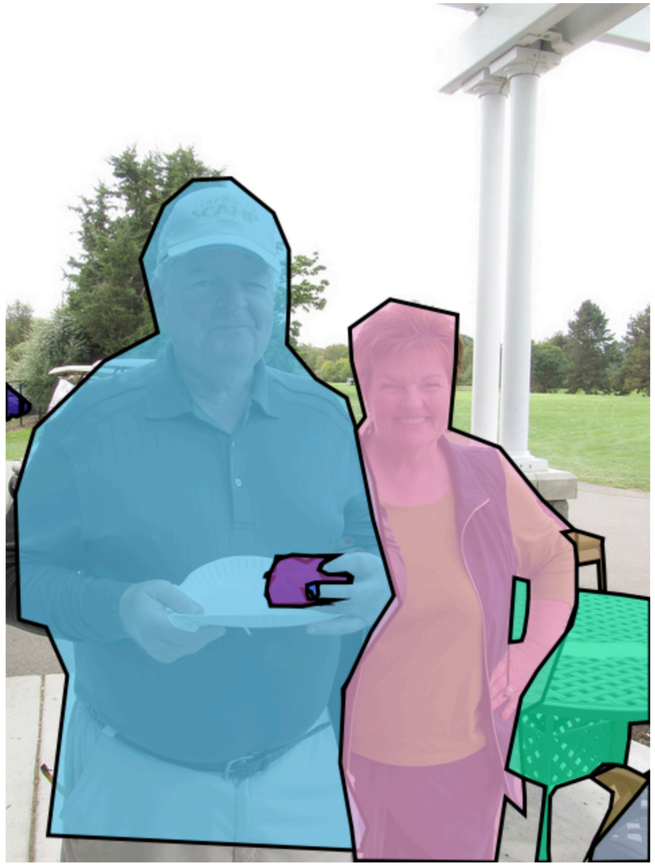


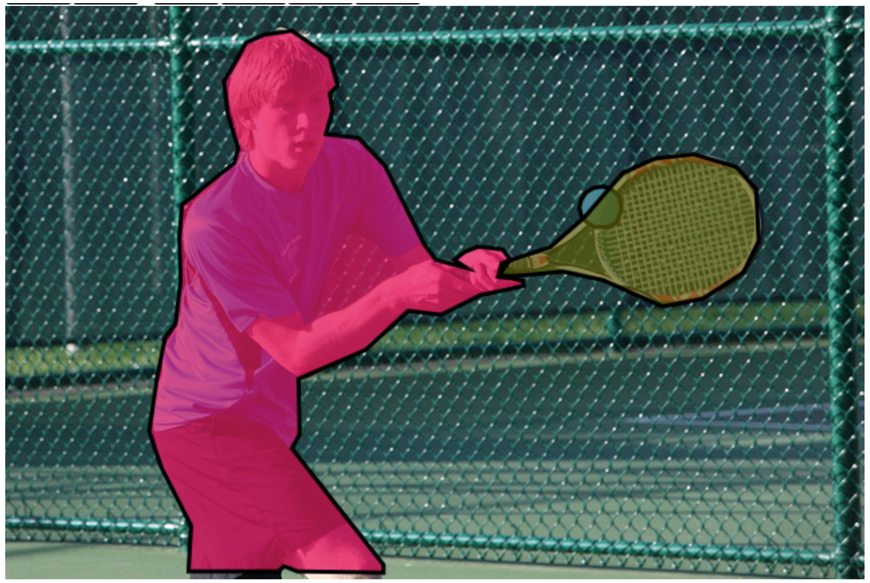
Instance Segmentation With Mask R-CNN

- Pixel-level classification is not accurate close to object borders
- Masks can be used for coarse object identification but not for image processing purposes (as seen below)

Insights on the COCO Image Dataset

- Common Objects in COntext (COCO) is a large dataset of 123,287 images and 80 different classes. However, human image masks are defined in a very coarse way, as shown on the examples below, so they are not suitable for accurate predictions. A better dataset with accurate contour masks is thus required.



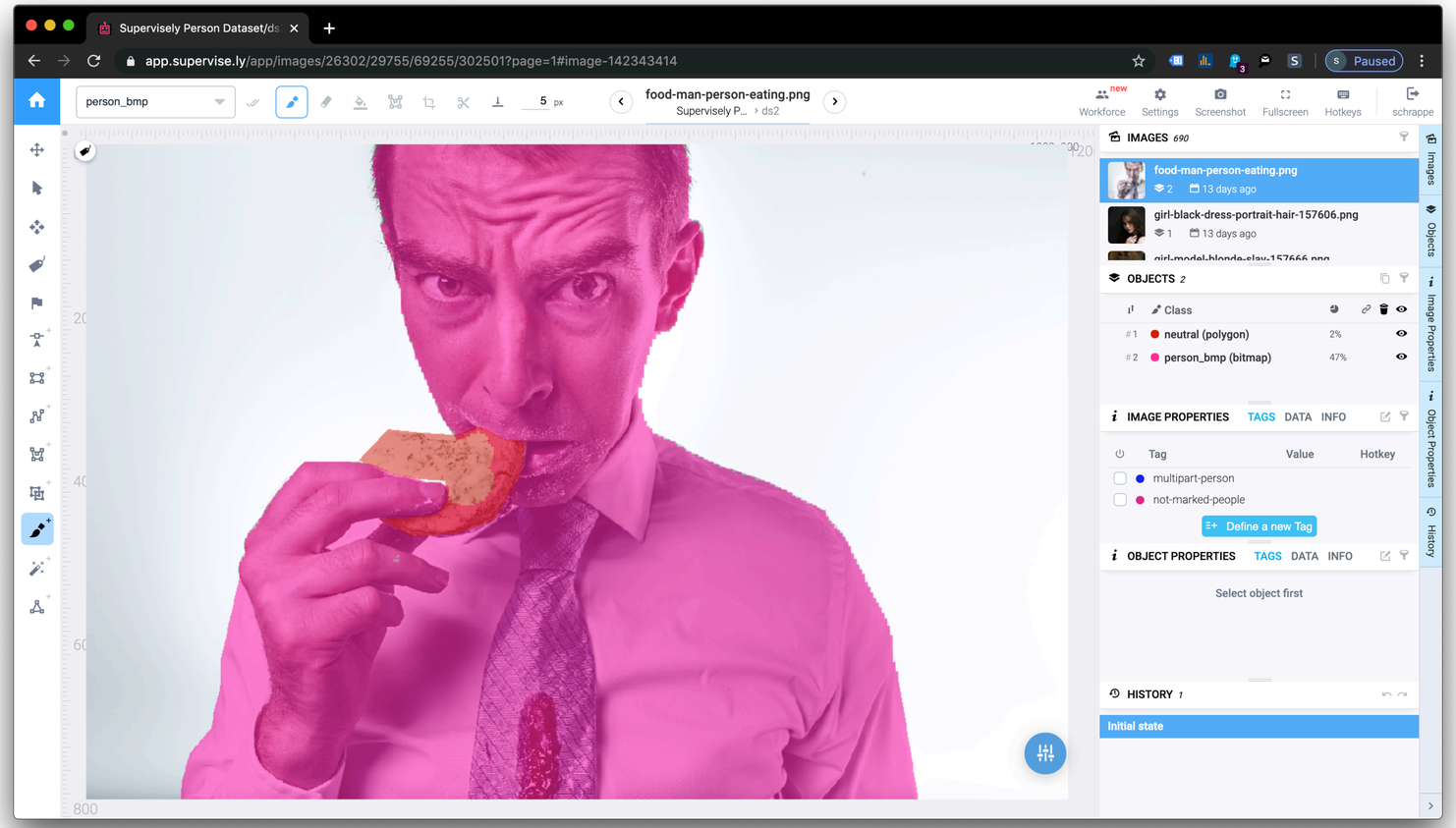


Source: http://cocodataset.org

Supervisely Human Dataset

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Supervisely (https://supervise.ly/) has made a tagged dataset publicly available through its portal for education purposes. Images are properly masked at a pixel level and can be edited online if so desired:



Supervisely Dataset Examples

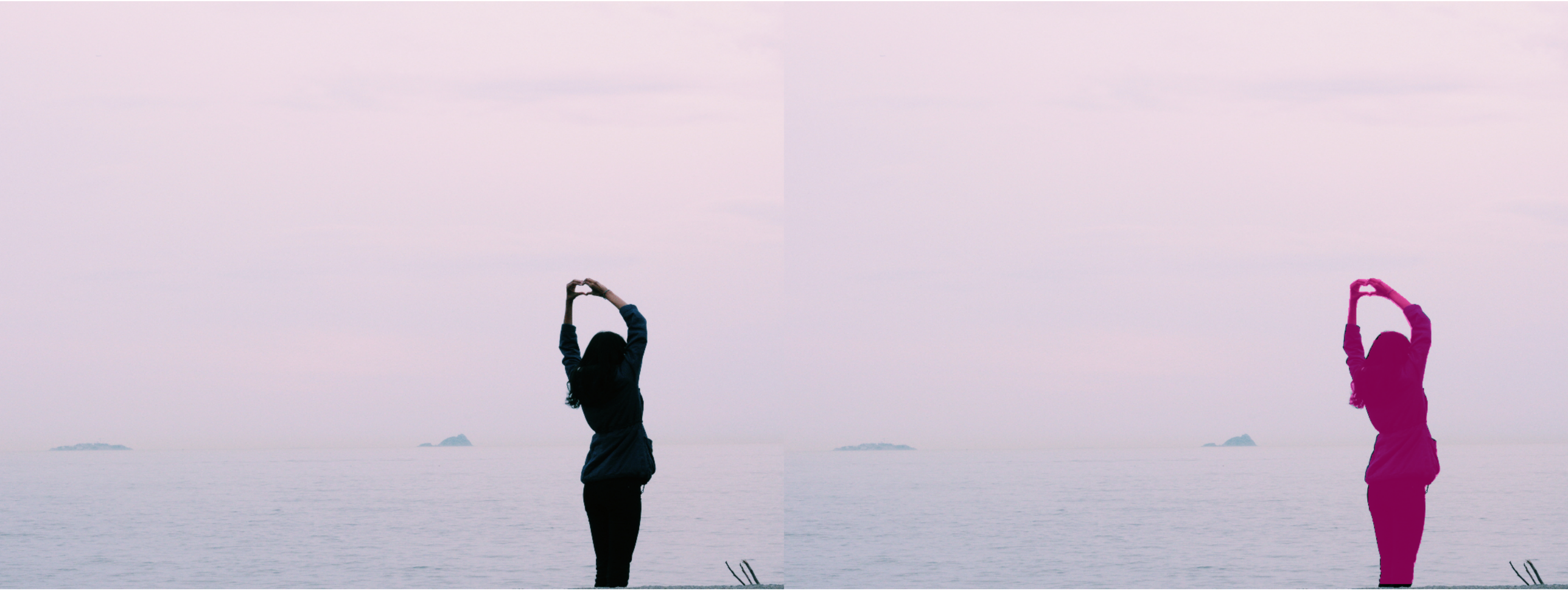
Supervisely Dataset Examples

Supervisely Dataset Examples

Supervisely Dataset Examples

Supervisely Dataset Examples

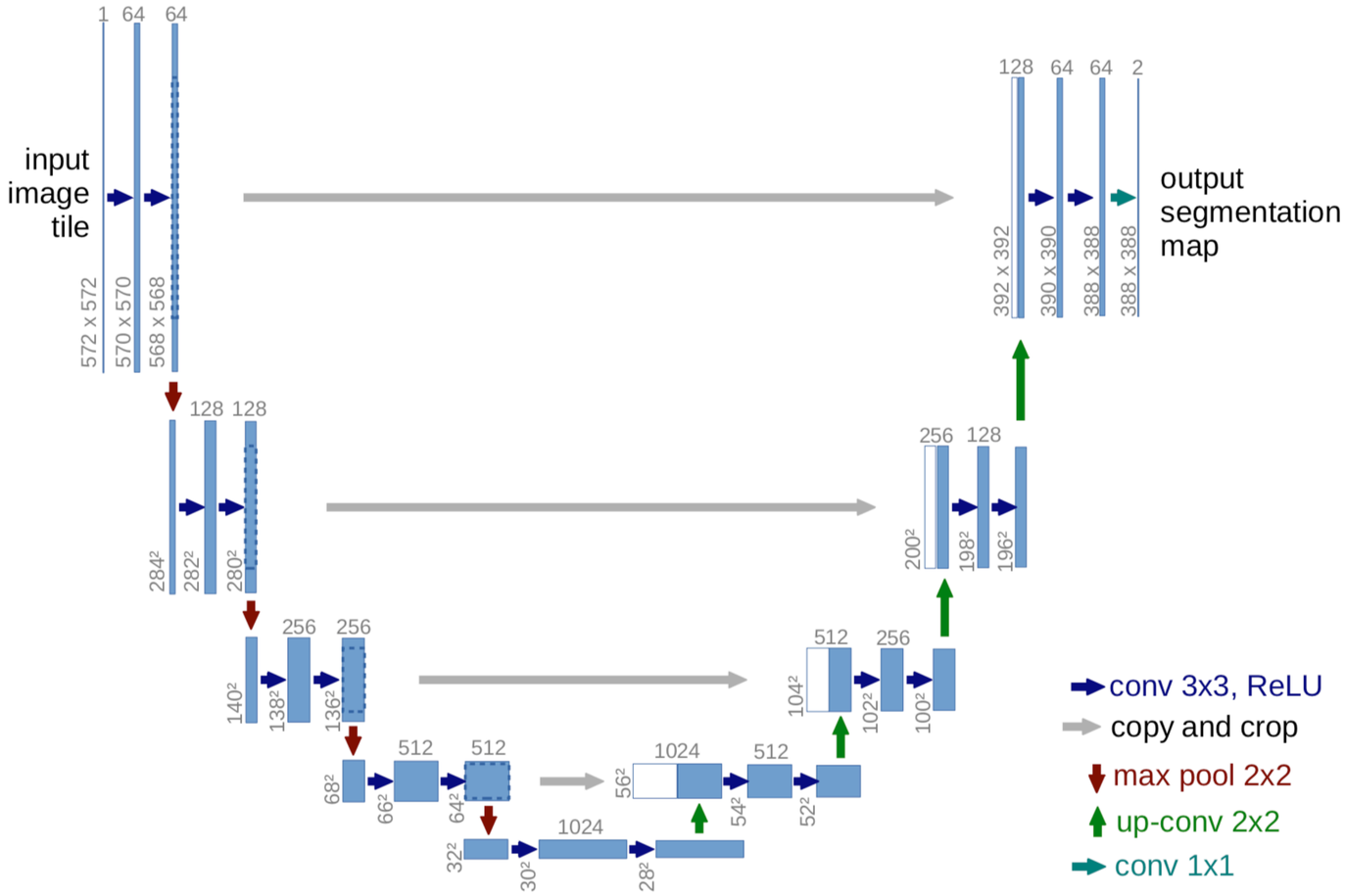
Supervisely Dataset Examples



UNET Model

- We decided to use the UNET model for image segmentation as it has a simpler architecture than Mask R-CNN, is focused on segmentation and performance is not an issue on this project.

- As all layers are convolutional or pooling, variable-shape image input is possible, as long as image dimensions can be halved 4 times resulting in integer dimensions upon pooling operations, so they can be reshaped to original dimension on upsampling layers (i.e. 128x130 maxpooled with (2,2) kernel becomes 64x65, then 32x32, which upsampled becomes 128x128)



Source: U-Net: Convolutional Networks for Biomedical Image Segmentation
Olaf Ronneberger, Philipp Fischer, and Thomas Brox
Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany

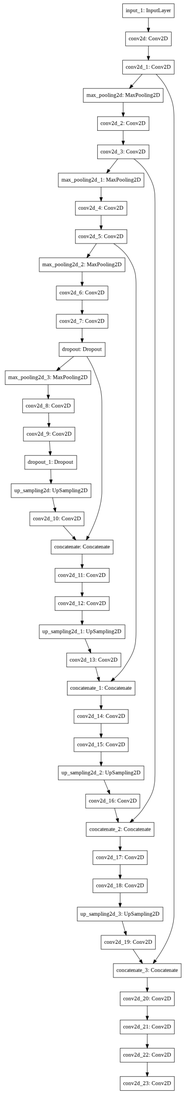
UNET Model Assembled in Keras

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

```
# Assembling the UNET Model using Keras

# Input shape left as variable as the first layer is convoluted - so it will accept variable-size images
inputs = keras.layers.Input(shape=(None, None, 3))

conv1 = keras.layers.Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
conv1 = keras.layers.Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)
pool1 = keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = keras.layers.Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
conv2 = keras.layers.Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
pool2 = keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = keras.layers.Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
conv3 = keras.layers.Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
pool3 = keras.layers.MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = keras.layers.Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
conv4 = keras.layers.Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4)
drop4 = keras.layers.Dropout(0.5)(conv4)
pool4 = keras.layers.MaxPooling2D(pool_size=(2, 2))(drop4)

conv5 = keras.layers.Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
conv5 = keras.layers.Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5)
drop5 = keras.layers.Dropout(0.5)(conv5)

up6 = keras.layers.Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(keras.layers.UpSampling2D(size = (2,2))(drop5))
merge6 = keras.layers.concatenate([drop4,up6], axis = 3)
conv6 = keras.layers.Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
conv6 = keras.layers.Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6)

up7 = keras.layers.Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(keras.layers.UpSampling2D(size = (2,2))(conv6))
merge7 = keras.layers.concatenate([conv3,up7], axis = 3)
conv7 = keras.layers.Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
conv7 = keras.layers.Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7)

up8 = keras.layers.Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(keras.layers.UpSampling2D(size = (2,2))(conv7))
merge8 = keras.layers.concatenate([conv2,up8], axis = 3)
conv8 = keras.layers.Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
conv8 = keras.layers.Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8)

up9 = keras.layers.Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(keras.layers.UpSampling2D(size = (2,2))(conv8))
merge9 = keras.layers.concatenate([conv1,up9], axis = 3)
conv9 = keras.layers.Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9)
conv9 = keras.layers.Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv9 = keras.layers.Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv10 = keras.layers.Conv2D(1, 1, activation = 'sigmoid')(conv9)

model = keras.Model(inputs = inputs, outputs = conv10)
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
conv2d (Conv2D)	(None, 256, 256, 64)	1792	input_1[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 64)	36928	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 128	73856	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 128	147584	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 256)	295168	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 256)	590080	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 512)	1180160	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 512)	2359808	conv2d_6[0][0]
dropout (Dropout)	(None, 32, 32, 512)	0	conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 512)	0	dropout[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 1024)	4719616	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 1024)	9438208	conv2d_8[0][0]
dropout_1 (Dropout)	(None, 16, 16, 1024)	0	conv2d_9[0][0]
up_sampling2d (UpSampling2D)	(None, 32, 32, 1024)	0	dropout_1[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 512)	2097664	up_sampling2d[0][0]
concatenate (Concatenate)	(None, 32, 32, 1024)	0	dropout[0][0] conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 512)	4719104	concatenate[0][0]
conv2d_12 (Conv2D)	(None, 32, 32, 512)	2359808	conv2d_11[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 64, 64, 512)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 256)	524544	up_sampling2d_1[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 512)	0	conv2d_5[0][0] conv2d_13[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 256)	1179904	concatenate_1[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 256)	590080	conv2d_14[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 128, 128, 256	0	conv2d_15[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 128	131200	up_sampling2d_2[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 256	0	conv2d_3[0][0] conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 128	295040	concatenate_2[0][0]
conv2d_18 (Conv2D)	(None, 128, 128, 128	147584	conv2d_17[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 256, 256, 128	0	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 256, 256, 64)	32832	up_sampling2d_3[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 128	0	conv2d_1[0][0] conv2d_19[0][0]
conv2d_20 (Conv2D)	(None, 256, 256, 64)	73792	concatenate_3[0][0]
conv2d_21 (Conv2D)	(None, 256, 256, 64)	36928	conv2d_20[0][0]
conv2d_22 (Conv2D)	(None, 256, 256, 2)	1154	conv2d_21[0][0]
conv2d_23 (Conv2D)	(None, 256, 256, 1)	3	conv2d_22[0][0]

Total params: 31,032,837
Trainable params: 31,032,837
Non-trainable params: 0



Data Preparation and Training

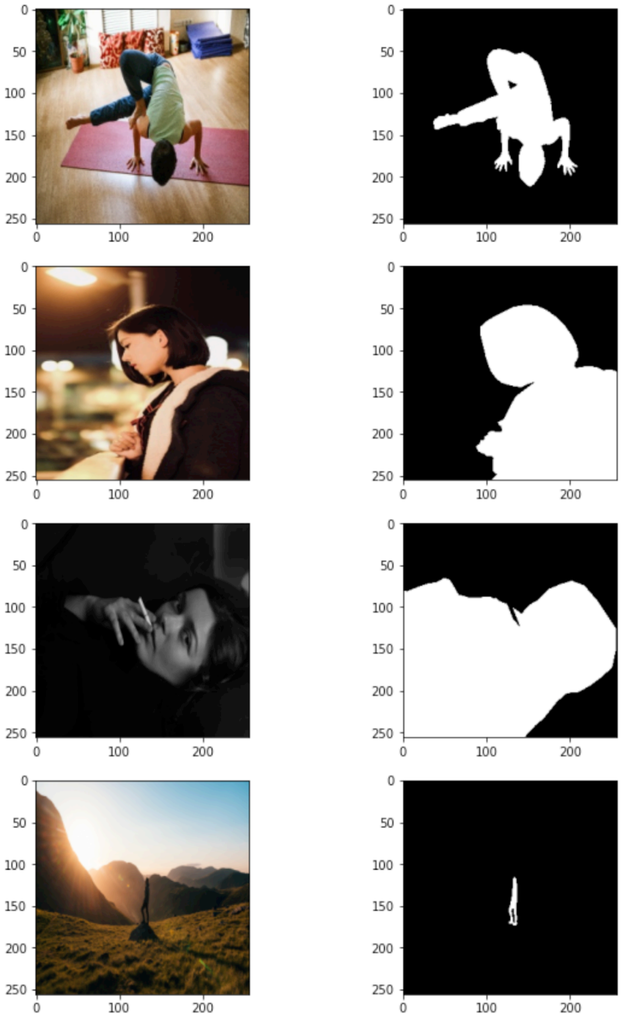
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- So as to maintain required resources within reasonable limits, we used a fairly small dataset with 3,053 images (ds1, ds2 and ds6 subsets from Supervisely) and masks reshaped to 256x256 pixels.

- Initial training of the UNET model was performed on Google Colab Pro using a Tesla P100 PCIe 16 GB GPU and a 25 GB RAM instance.

- No image augmentation was used on the initial training run.

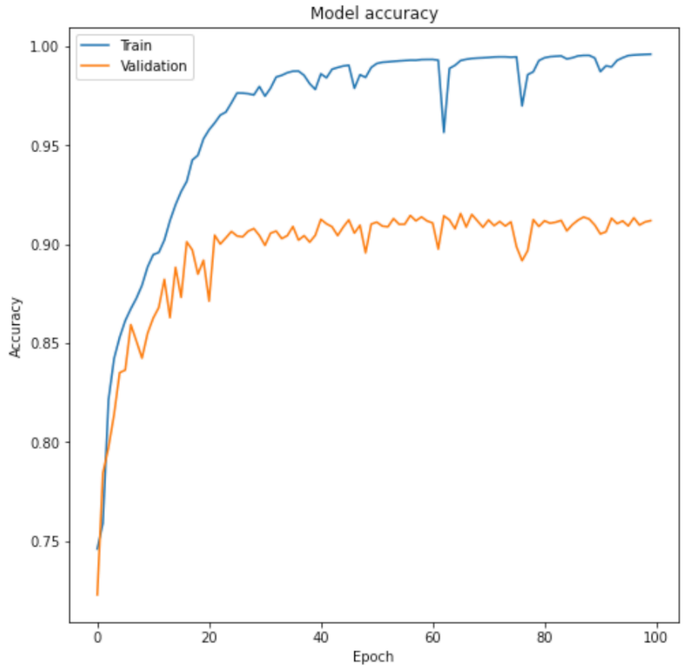
Sample of training images and masks

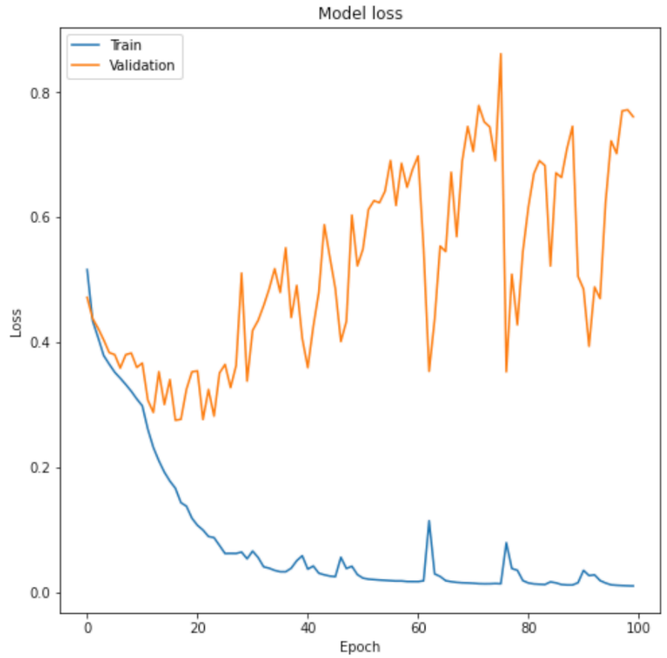


Initial Training Results

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Training with 100 epochs, with early stopping on validation accuracy and saving the best model takes 2h51min, close to overfitting but validation accuracy is kept fairly stable.

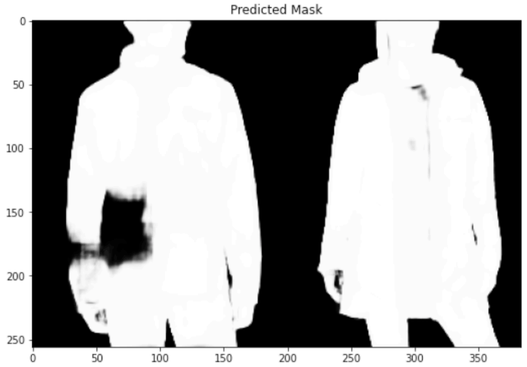


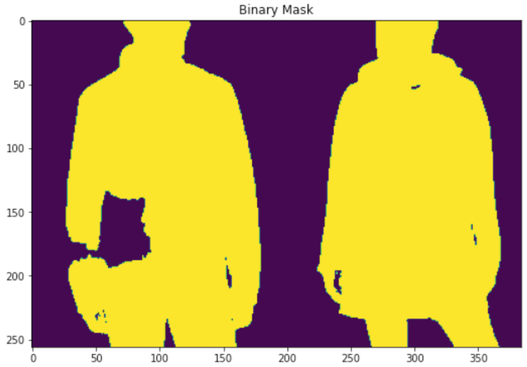


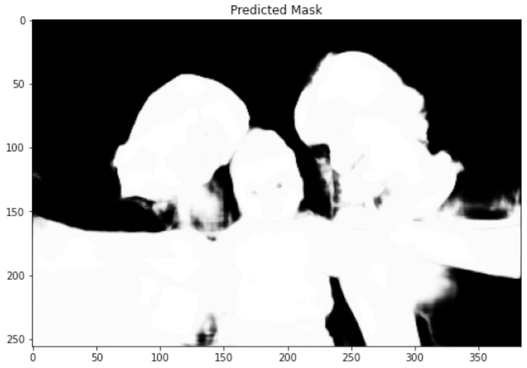
Initial Results

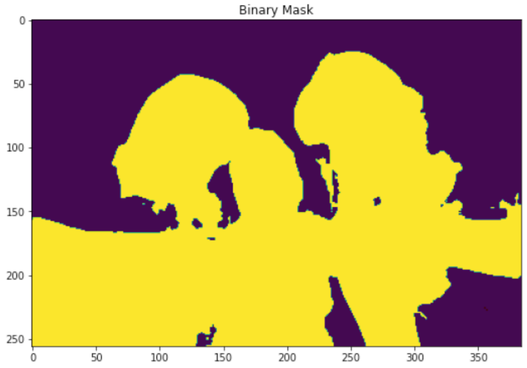
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

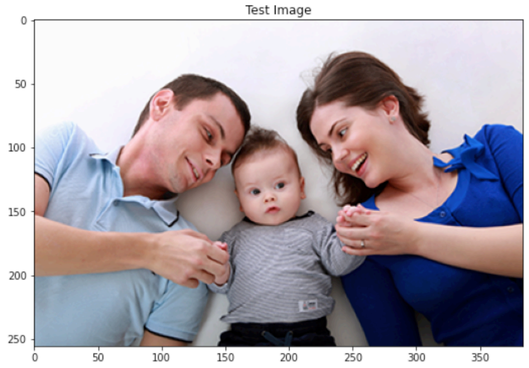
- With the limited dataset, testing on new images clearly shows some image features were not captured, leading to misclassified areas as shown below:

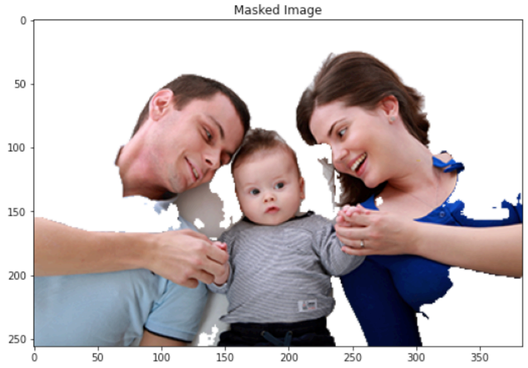








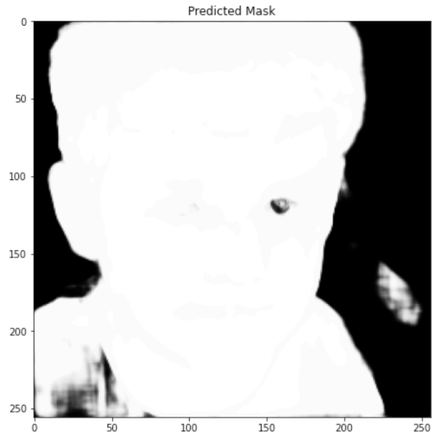


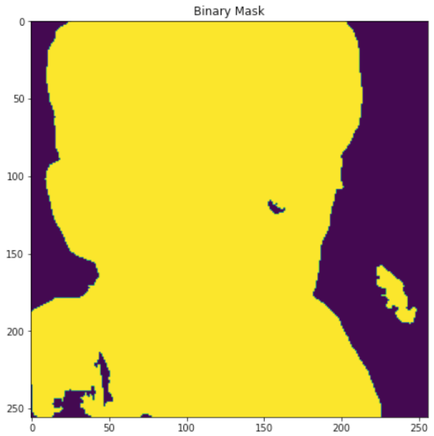


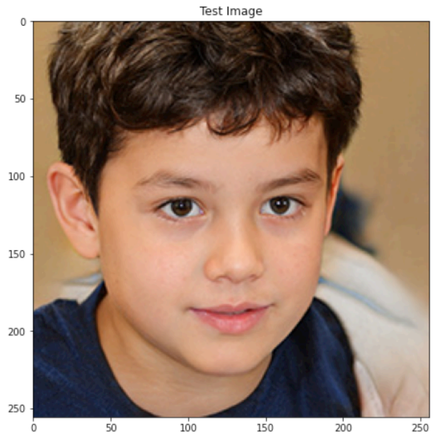
Initial Results

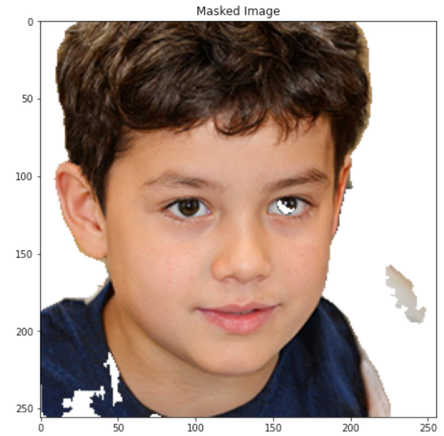
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

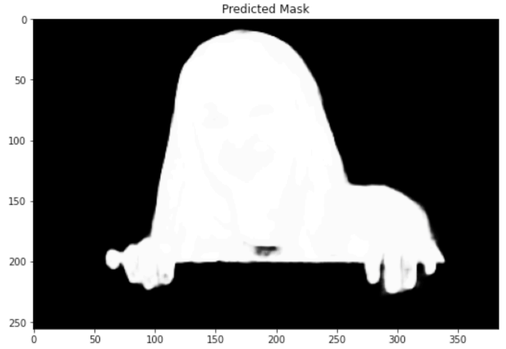
- With the limited dataset, testing on new images clearly shows some image features were not captured, leading to misclassified areas as shown below:

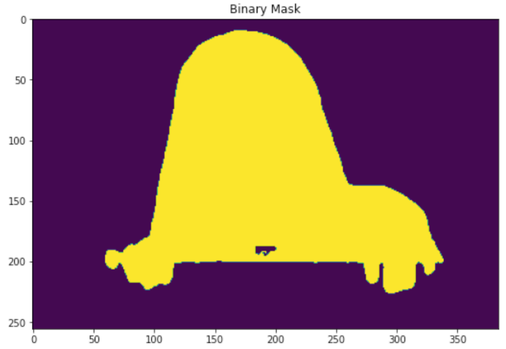


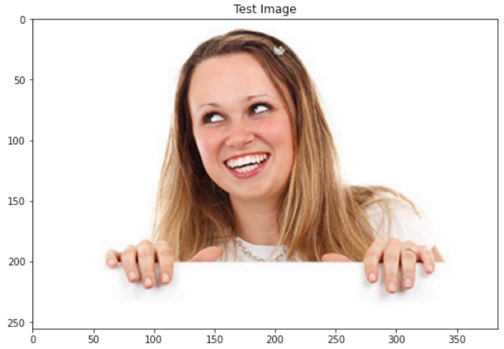








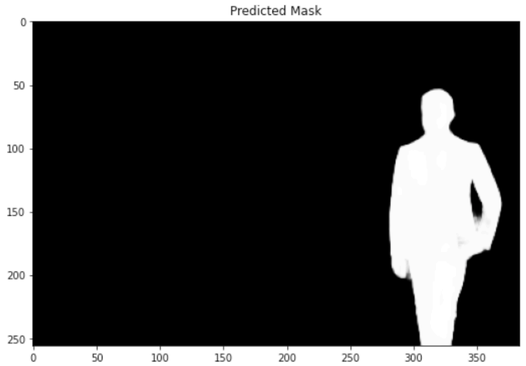


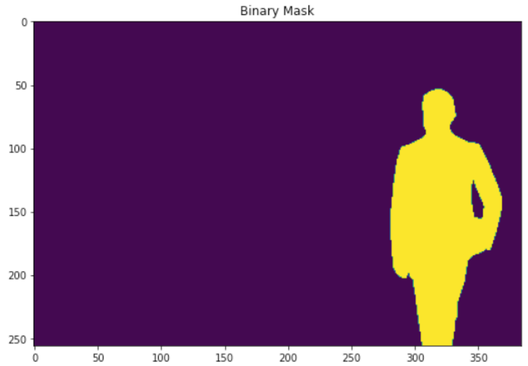


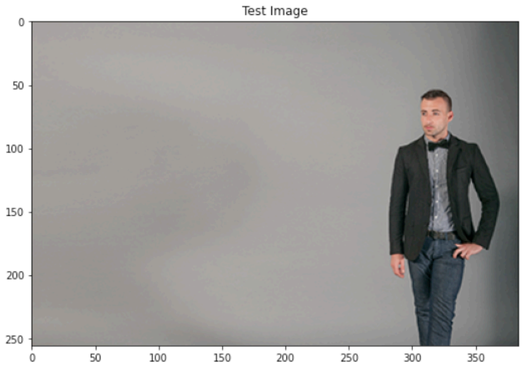
Initial Results

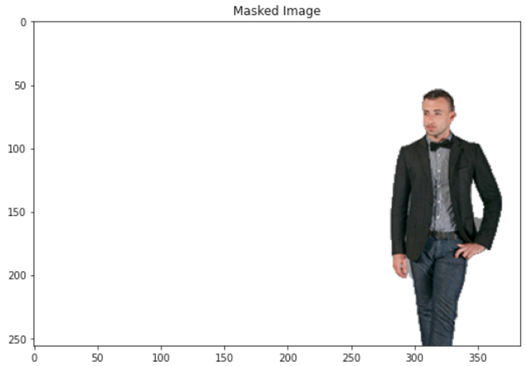
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

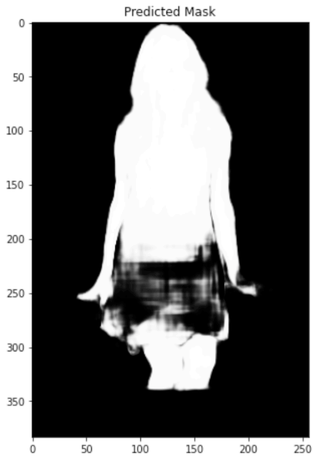
- With the limited dataset, testing on new images clearly shows some image features were not captured, leading to misclassified areas as shown below:

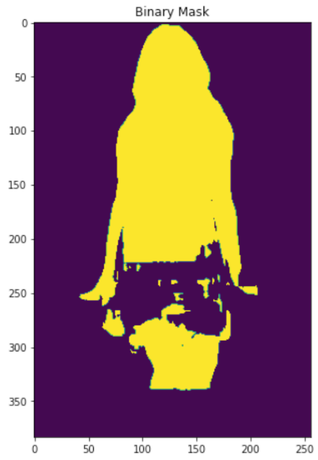


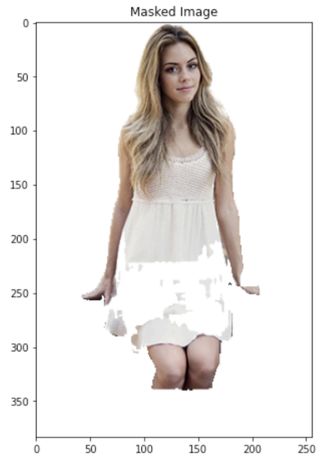












Augmenting the Image Dataset

- So as to generate a larger set of images, allowing the model to better generalize on features, we used Keras' ImageDataGenerator capability.

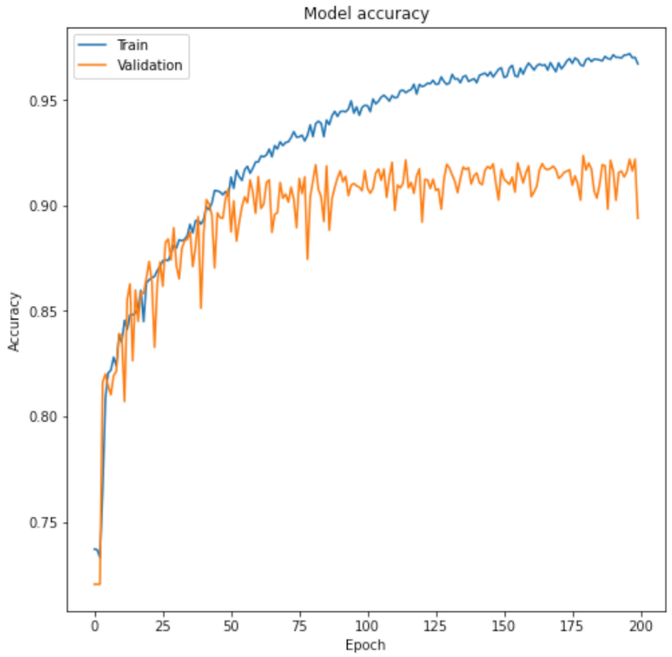
- Random transformations on shear, rotation, zoom and dimension were added to the training dataset (but not to the validation dataset), retraining the model.

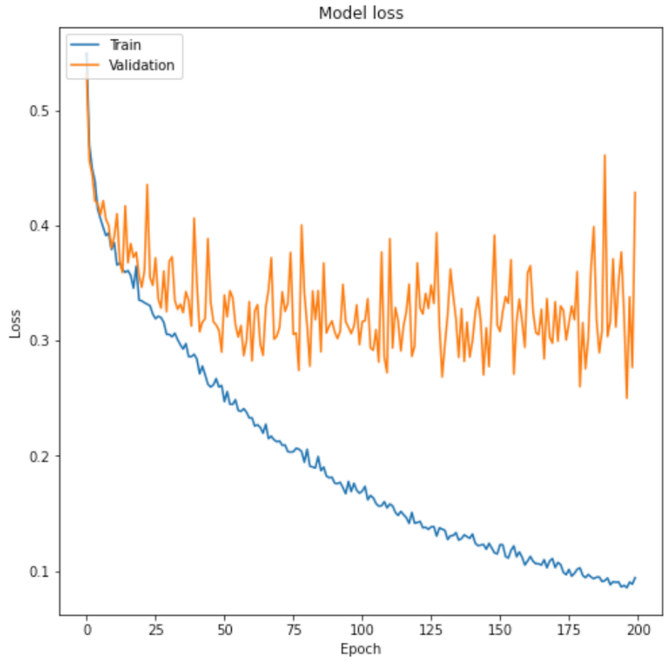
```
from tensorflow.keras.preprocessing import image

train_size = 2750

# Training data will be X[0:train_size]
# Validation data will be X[train_size:]

# Training masks will be y[0:train_size]
# Validation masks will be y[train_size:]

# Augmenting the dataset
image_datagen = image.ImageDataGenerator(shear_range=0.5, rotation_range=20, zoom_range=0.2, width_shift_range=0.2, height_shift_range=0.2, fill_mode='reflect')
mask_datagen = image.ImageDataGenerator(shear_range=0.5, rotation_range=20, zoom_range=0.2, width_shift_range=0.2, height_shift_range=0.2, fill_mode='reflect')

# Keep the same seed for image and mask generators so they fit together
image_datagen.fit(X[0:train_size], augment=True, seed=42)
mask_datagen.fit(y[0:train_size], augment=True, seed=42)

augmented_X_train = image_datagen.flow(X[0:train_size],batch_size=15,shuffle=True, seed=42)
augmented_y_train = mask_datagen.flow(y[0:train_size],batch_size=15,shuffle=True, seed=42)

train_generator = zip(augmented_X_train, augmented_y_train)
```

Training With the Augmented Dataset

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- Training the augmented dataset with 200 epochs took 4 hours on the P100 GPU. Although close to overfitting training data, validation accuracy and loss were a bit unstable, suggesting we were still missing feature generalization, but overall results were satisfactory for the scope of this project.

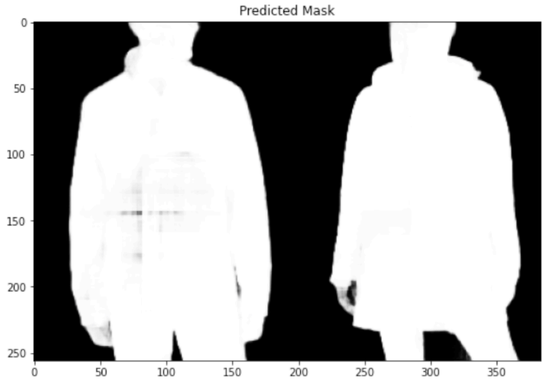


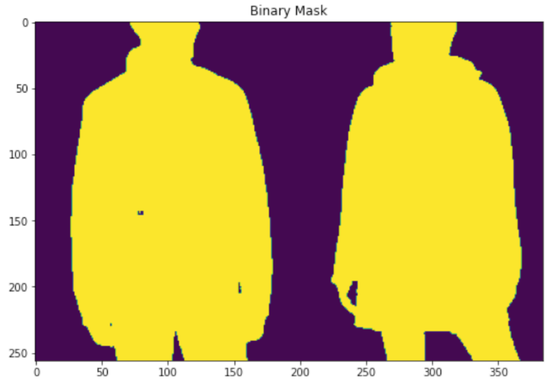


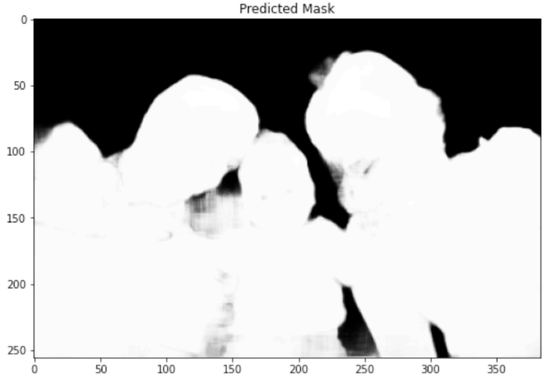
Test Results after Image Data Augmentation

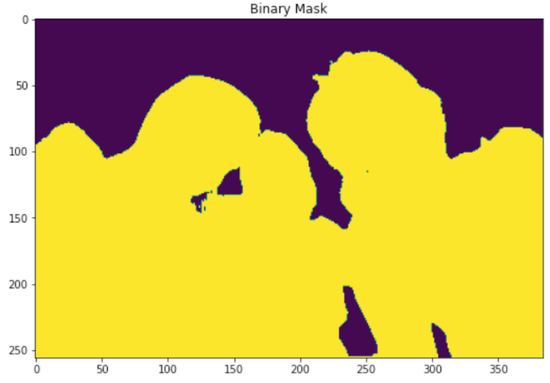
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

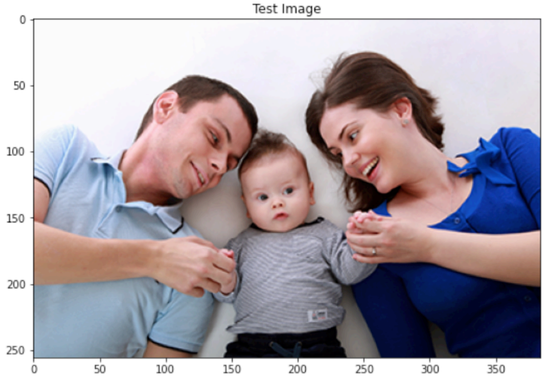
- After training with randomized transformations applied to the training dataset, better results were achieved as shown below, proving the validity of augmentation with Keras's ImageDataGenerator:

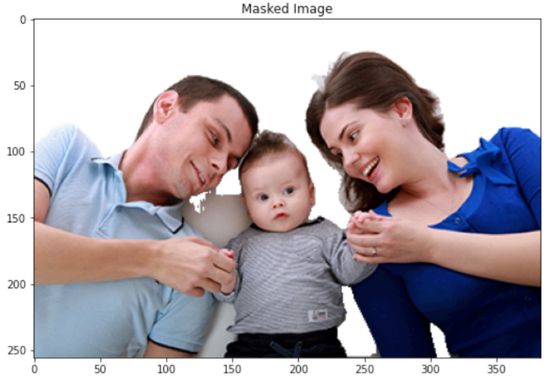






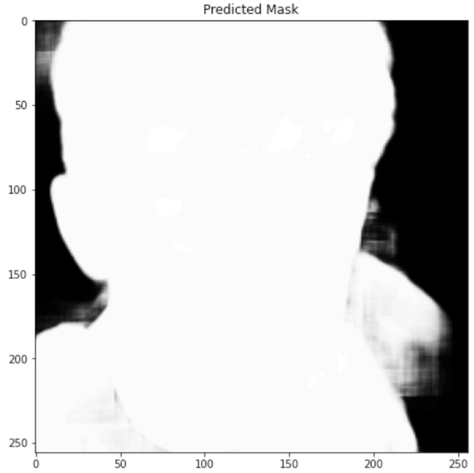


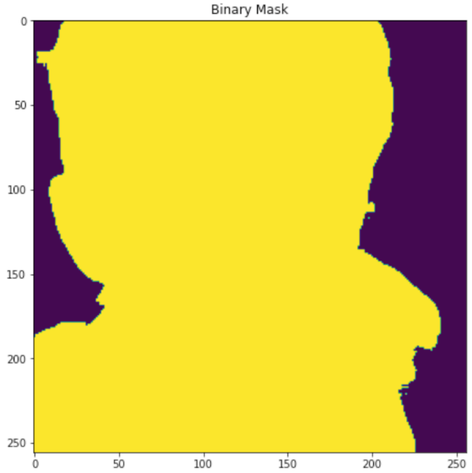


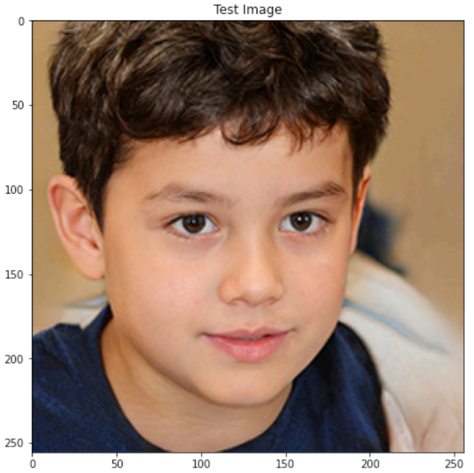


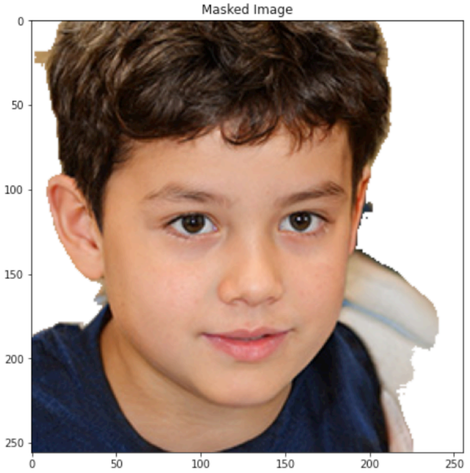
Test Results after Image Data Augmentation

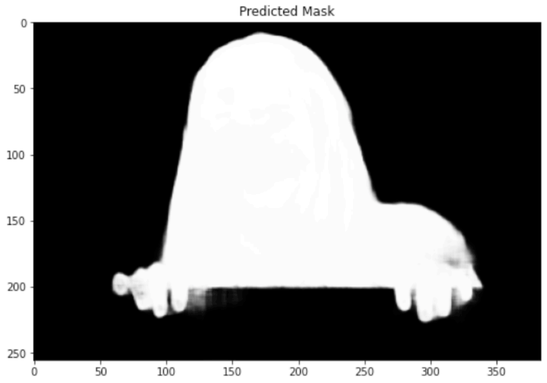
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

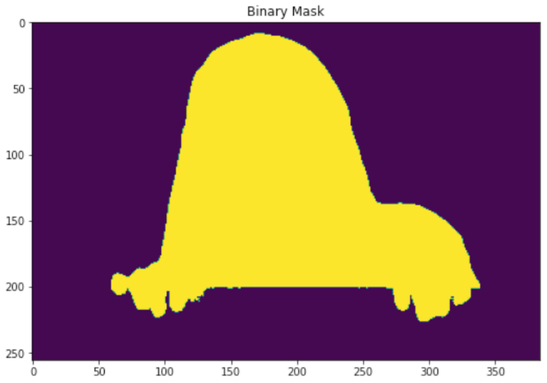


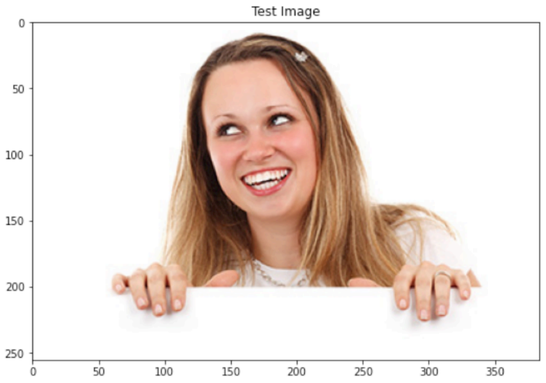






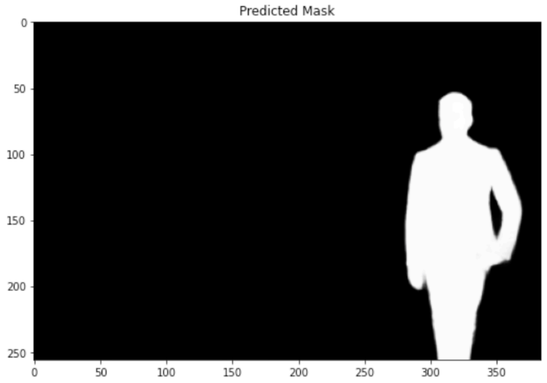


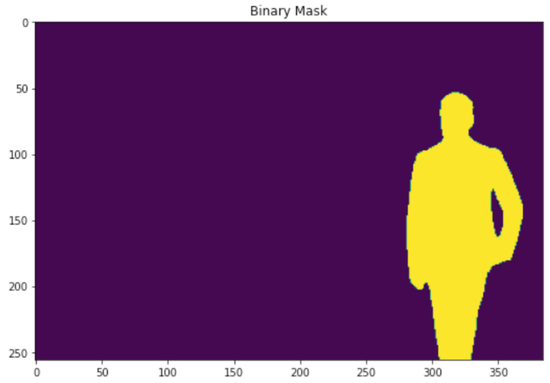


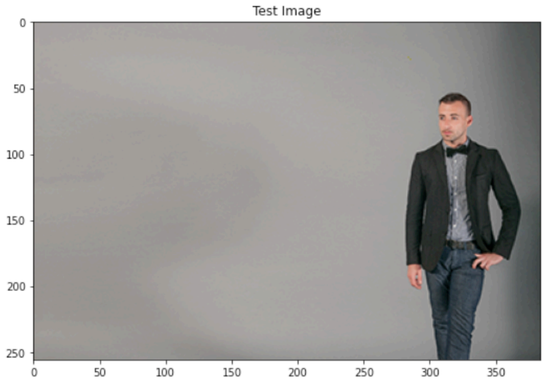


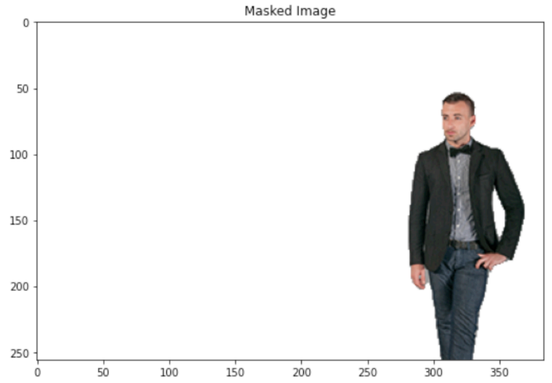
Test Results after Image Data Augmentation

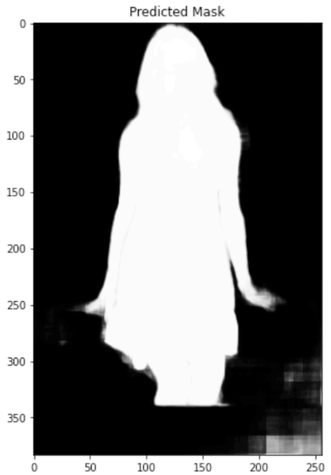
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

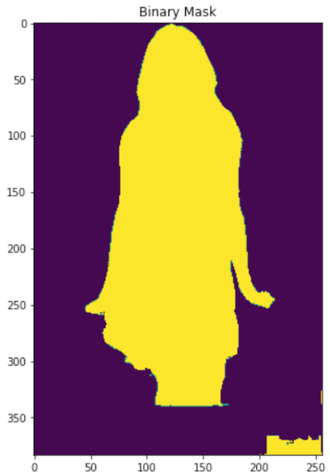


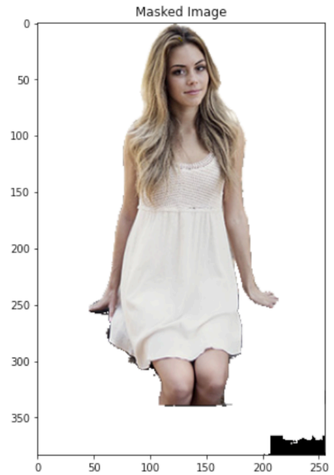






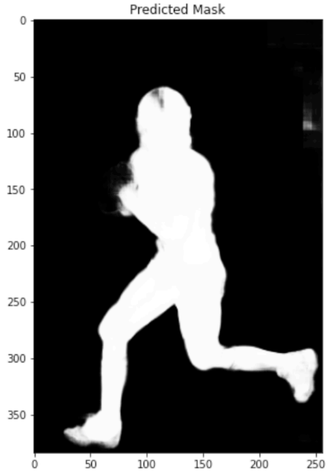


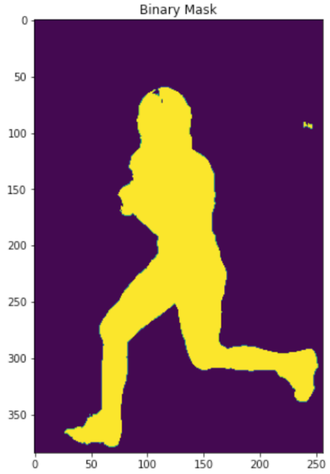


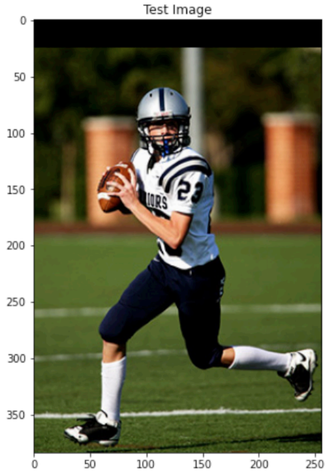


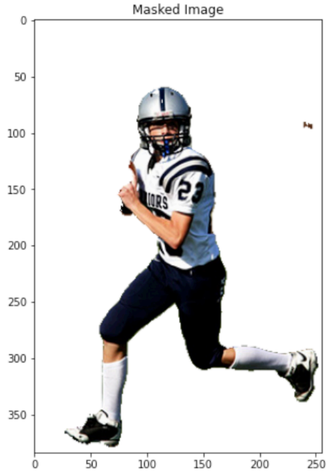
Test Results after Image Data Augmentation

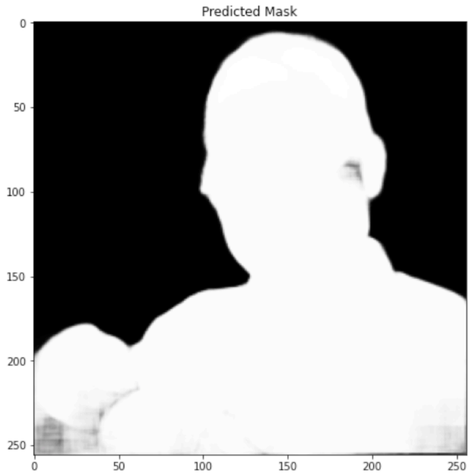
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

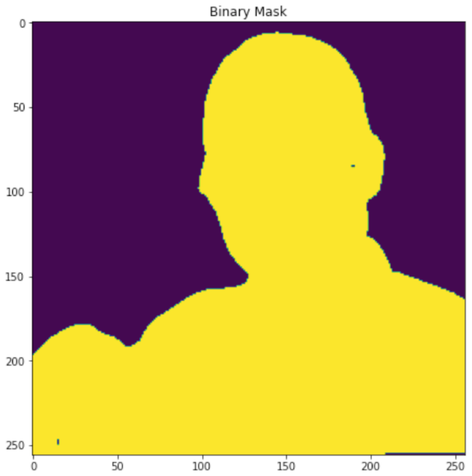


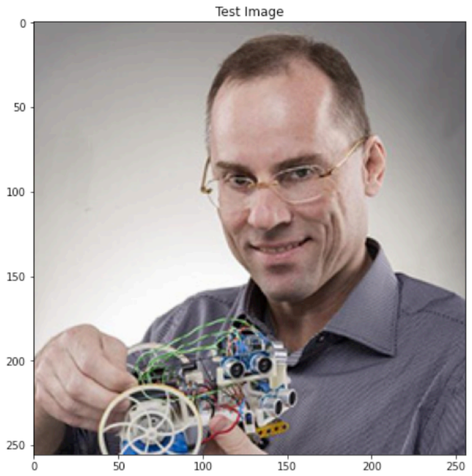


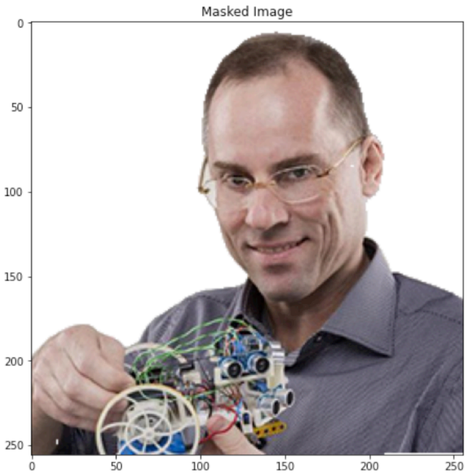






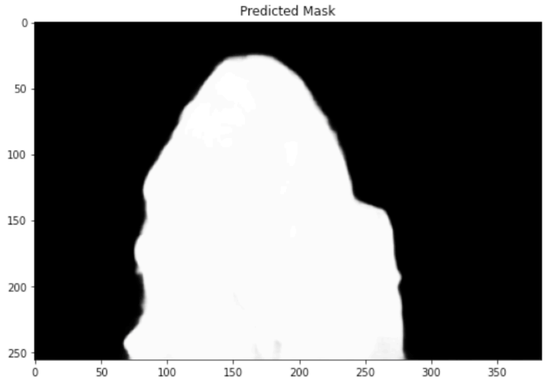


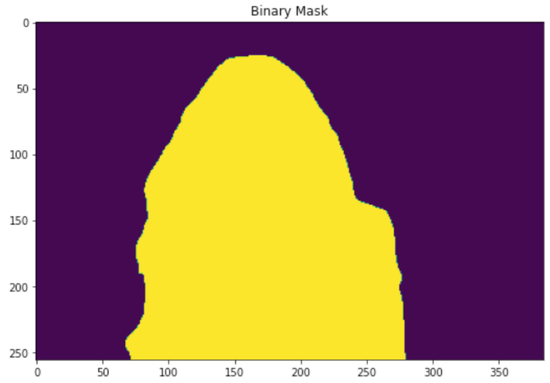


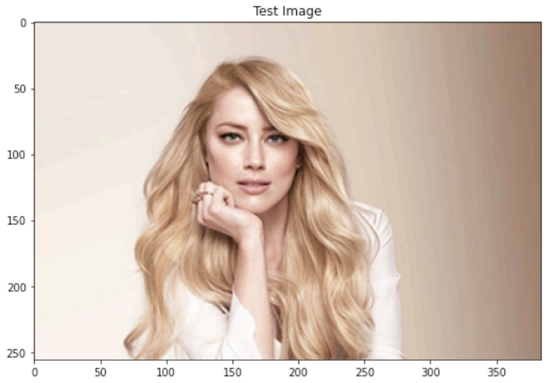


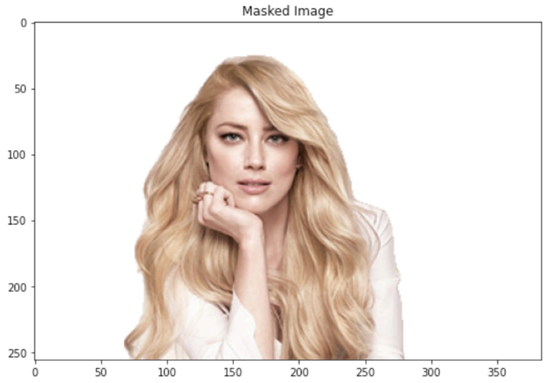
Test Results after Image Data Augmentation

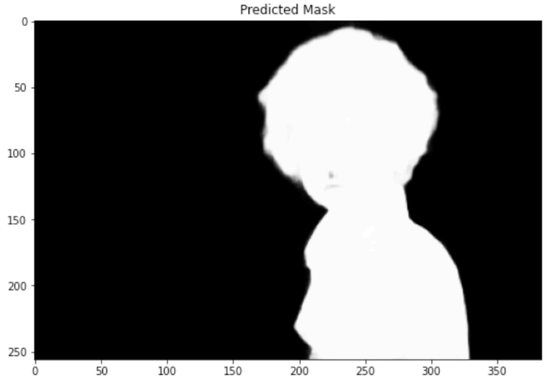
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

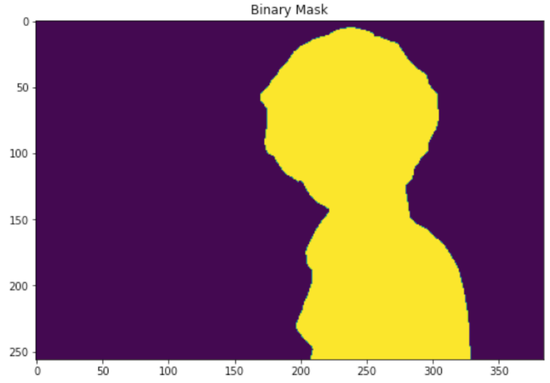


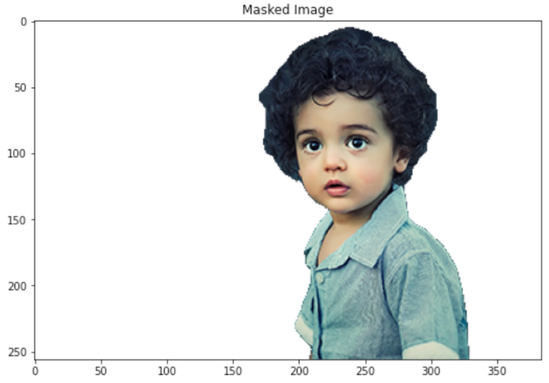


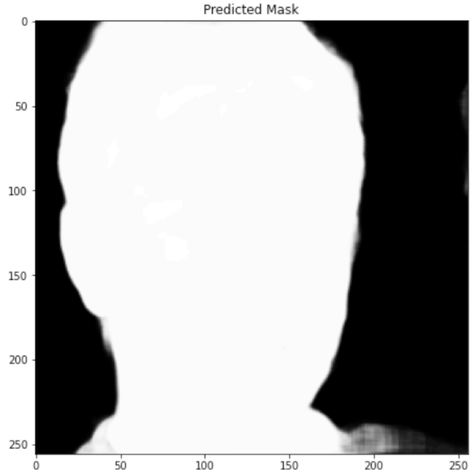


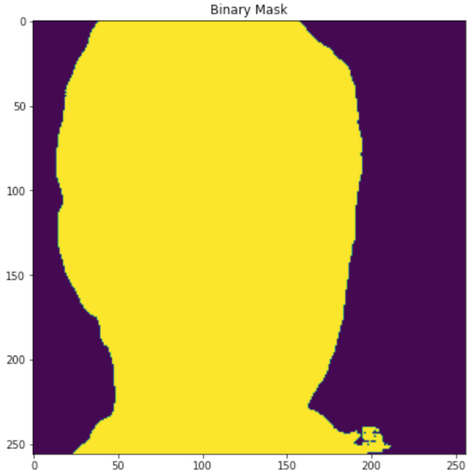


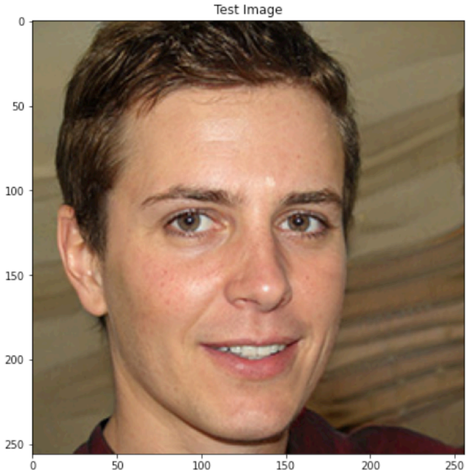


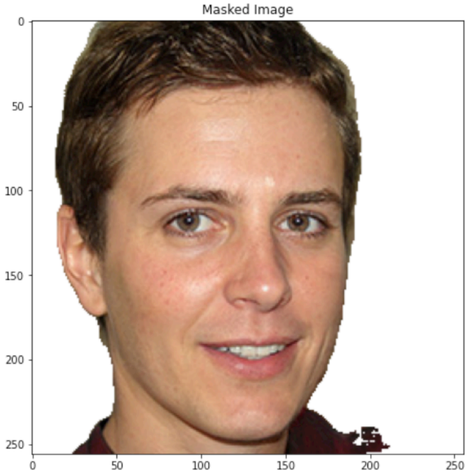


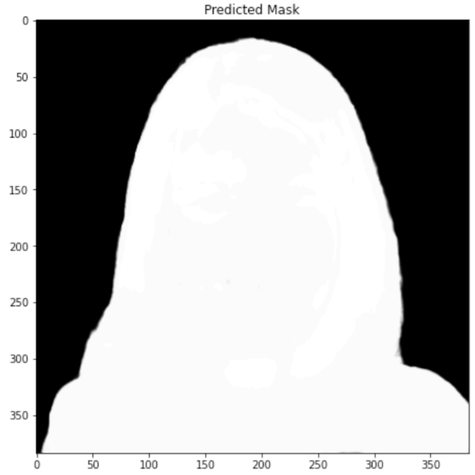


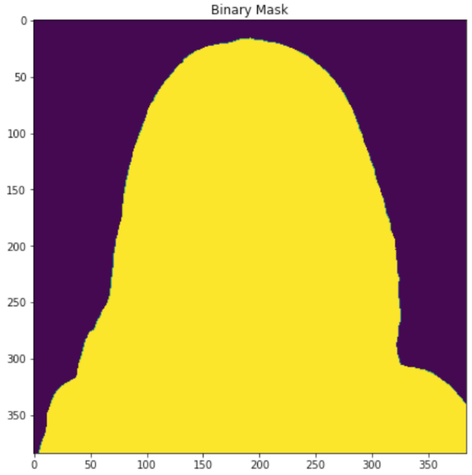

Predicted Mask

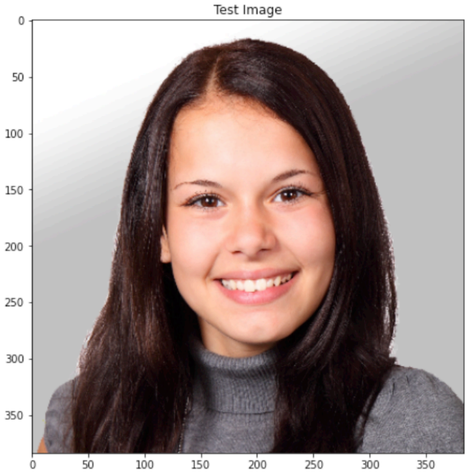

Binary Mask

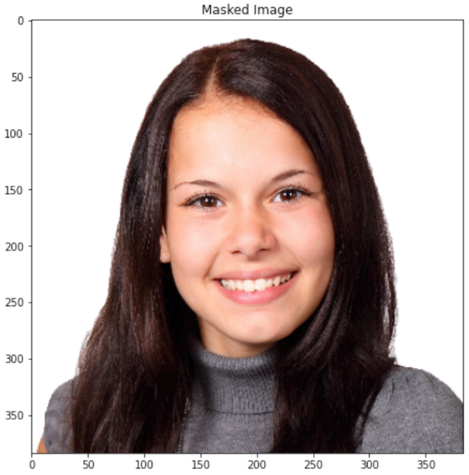

Test Image


Masked Image


Predicted Mask


Binary Mask


Test Image


Masked Image

Conclusions and Recommendations

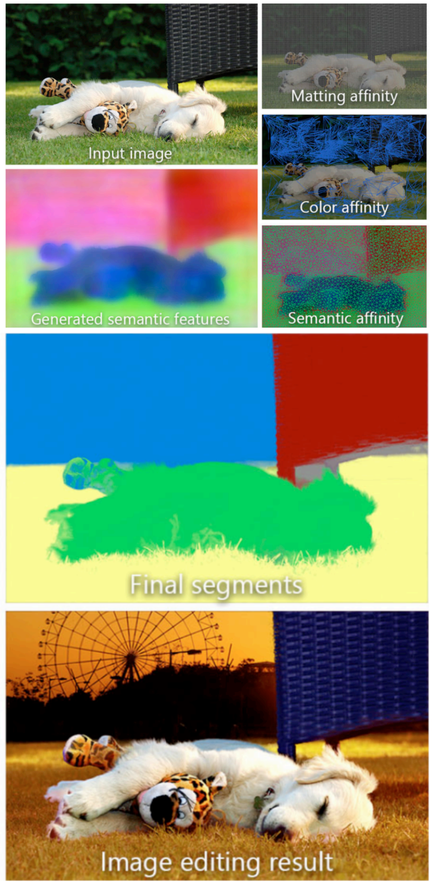
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- For our specific purposes, UNET fully trained on a dedicated dataset provided better results on semantic segmentation than Mask R-CNN trained on a larger, but generic, dataset (COCO).

- Data augmentation proved to be a crucial step to increase generalization of features.

- Additional quality enhancements could be achieved by increasing not only the training data set size, but also its resolution, which was kept low given limited resources on Google Colab Pro.

- Further enhancements may rely on more advance techniques, such as described by YAĞIZ AKSOY et al, on "Semantic Soft Segmentation", combining semantic features with matting affinity, color affinity and semantic affinity to generate masks with softer edges, as shown on the right:



Source: Semantic Soft Segmentation: http://cfg.mit.edu/sites/cfg.mit.edu/files/sss_3.pdf

References

Mask R-CNN: https://arxiv.org/abs/1703.06870

Understanding Semantic Segmentation with UNET: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Mask-RCNN and U-net Ensembled for Nuclei Segmentation: https://arxiv.org/abs/1901.10170

A Keras Pipeline for Image Segmentation: https://towardsdatascience.com/a-keras-pipeline-for-image-segmentation-part-1-6515a421157d

Background removal with deep learning: https://towardsdatascience.com/background-removal-with-deep-learning-c4f2104b3157

Releasing “Supervisely Person” dataset for teaching machines to segment humans: https://hackernoon.com/releasing-supervisely-person-dataset-for-teaching-machines-to-segment-humans-1f1fc1f28469

U-Net: Convolutional Networks for Biomedical Image Segmentation: https://arxiv.org/pdf/1505.04597.pdf

Coco Dataset: http://cocodataset.org/

Supervisely Dataset: https://supervise.ly/

Practical Image Segmentation with UNET: https://tuatini.me/practical-image-segmentation-with-unet/

nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation: https://arxiv.org/abs/1809.10486

Semantic Soft Segmentation: http://cfg.mit.edu/sites/cfg.mit.edu/files/sss_3.pdf

Thank you!

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

- https://github.com/schrappe/semantic-segmentation