Carvana Image Masking Challenge

Koh Yong Tat

Introduction

Carvana, an online used car retailer, has took pictures of used cars using their custom rotating photo studio that automatically captures and processes 16 standard images of each vehicle in their inventory. Bright reflections and cars with similar colors as the background cause automation errors, which requires a skilled photo editor to manually edit.

In this Kaggle challenge, the objective is to develop an algorithm that automatically removes the photo studio background. This will allow Carvana to superimpose cars on a variety of backgrounds.

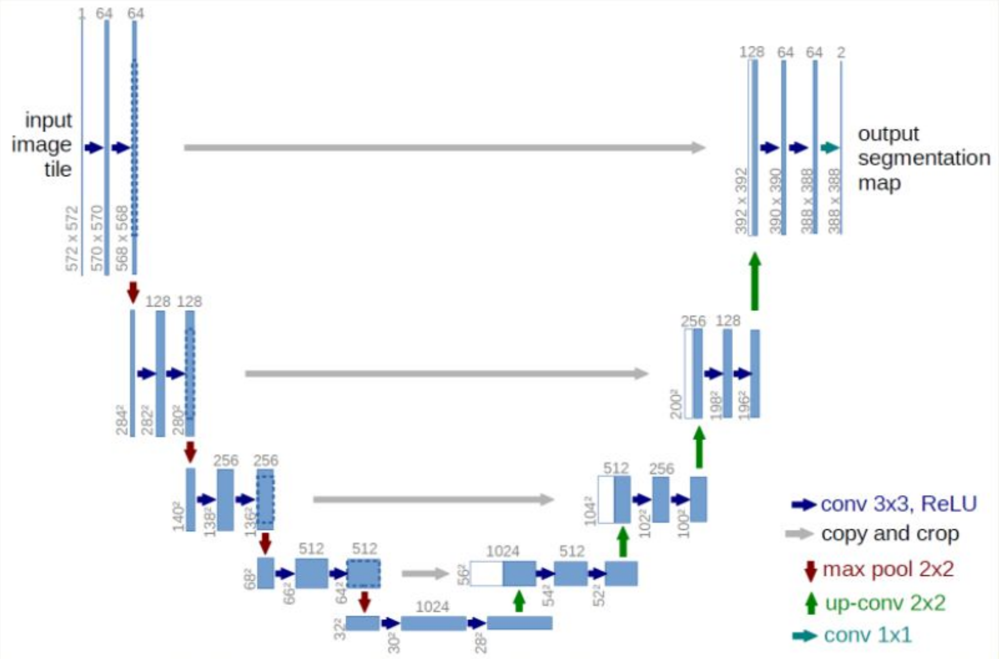
The dataset consists of photos covering different vehicles with a wide variety of year, make, and model combinations.

The approach is to build a UNet Convolutional Neural Network (UNet-CNN) model to predict a mask image from the images. The predicted mask will be encoded using run-length encoding (rle) and exported in .csv format to submit to Kaggle for scoring.

Kaggle Link:

https://www.kaggle.com/c/carvana-image-masking-challenge/overview

What is the UNet CNN



U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

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

U-Net: Convolutional Networks for

Biomedical Image Segmentation

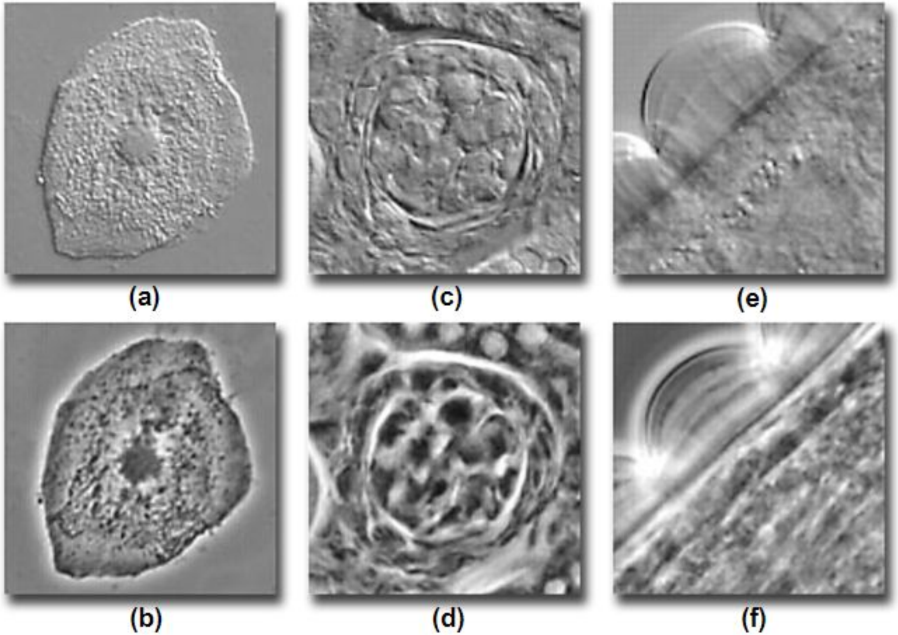
The u-net is convolutional network architecture for fast and precise segmentation of images.

- Won the Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI 2015

- Won the Cell Tracking Challenge at ISBI 2015 on the two most challenging transmitted light microscopy categories (Phase contrast and DIC microscopy) by a large margin.

Phase Contrast vs Differential Interference Contrast (DIC)

Figure 1 - *Transparent Specimens in Phase Contrast and DIC*



- Images a,c,e are DIC
- Images b,d,f are Phase Contrast

Source: https://www.microscopyu.com/tutorials/comparison-of-phase-contrast-and-dic-microscopy

General Workflow

Creating datasets

- Train data, consisting a total of 5088 images (.jpg) and masks(.gif) with 1912 x 1280 resolution is split into **training** and **validation** data.
- 318 unique cars are found in the train data and randomly split into 254 for training, 64 for validation.
- Image processing and data augmentation is applied and data is flowed into Tensorflow datasets.

Build UNet CNN with transfer learning <u>no optimization</u> [MobileNetV2 + pix2pix]

- Downsampling encoding is built with layers from MobileNetV2 neural net.
- Upsampling decoding layers is built from upsampling layers from pix2pix model.
- Model is optimized for Tensorflow image segmentation tutorial but not for Carvana dataset.

Build UNet CNN with transfer learning <u>and optimization</u> [MobileNetV2 + pix2pix]

- Model is optimized for Carvana dataset by tuning the arguments for upsampling layers.

Build UNet CNN with transfer learning and optimization [VGG19 + pix2pix]

- Downsampling encoding is built with layers from VGG19 neural net.
- Upsampling decoding layers is built from upsampling layers from pix2pix model.
- Due to limited computational power, this model is not trained.

Build UNet CNN from scratch

- UNet CNN is build from scratch with 4 downsampling layers and 4 upsampling layers.
- Due to limited computational power, this model is not trained.

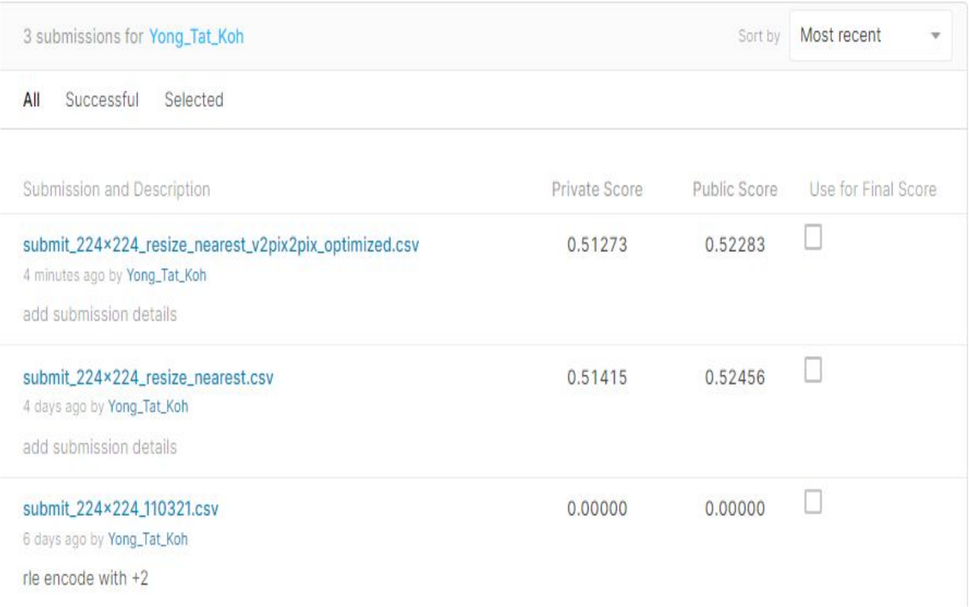
Run-length encode (rle) predicted masks

- Test data, consisting a total of 100,064 images (.jpg) with 1912 x 1280 resolution is flowed into a tensorflow dataset.
- Trained models will predict the masks for test data.
- The masks are encoded with rle and exported as a .csv file tagged to the image file names.

Submit rle masks generated from predicted images to Kaggle in as .csv

- It takes around 3.5 hours to rle the predicted masks as the test database is very large.

- Rle is done on a local machine.



Creating datasets

Train data, consisting a total of 5088 images (.jpg) and masks(.gif) with 1912 x 1280 resolution is split into **training** and **validation** data.

1) File paths and file names are put into a pandas dataframe.
2) Using from_tensor_slices function, the training and validation datasets are created as tensorflow datasets.

File paths and file names are put into a pandas dataframe

	train_path	car_id	mask_path	mask_id
0	data_lq\train\00087a6bd4dc_01.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_01_mask.gif	00087a6bd4dc
1	data_lq\train\00087a6bd4dc_02.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_02_mask.gif	00087a6bd4dc
2	data_lq\train\00087a6bd4dc_03.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_03_mask.gif	00087a6bd4dc
3	data_lq\train\00087a6bd4dc_04.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_04_mask.gif	00087a6bd4dc
4	data_lq\train\00087a6bd4dc_05.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_05_mask.gif	00087a6bd4dc
5	data_lq\train\00087a6bd4dc_06.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_06_mask.gif	00087a6bd4dc
6	data_lq\train\00087a6bd4dc_07.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_07_mask.gif	00087a6bd4dc
7	data_lq\train\00087a6bd4dc_08.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_08_mask.gif	00087a6bd4dc
8	data_lq\train\00087a6bd4dc_09.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_09_mask.gif	00087a6bd4dc
9	data_lq\train\00087a6bd4dc_10.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_10_mask.gif	00087a6bd4dc
10	data_lq\train\00087a6bd4dc_11.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_11_mask.gif	00087a6bd4dc
11	data_lq\train\00087a6bd4dc_12.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_12_mask.gif	00087a6bd4dc
12	data_lq\train\00087a6bd4dc_13.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_13_mask.gif	00087a6bd4dc
13	data_lq\train\00087a6bd4dc_14.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_14_mask.gif	00087a6bd4dc
14	data_lq\train\00087a6bd4dc_15.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_15_mask.gif	00087a6bd4dc
15	data_lq\train\00087a6bd4dc_16.jpg	00087a6bd4dc	data_lq\train_masks\00087a6bd4dc_16_mask.gif	00087a6bd4dc
16	data_lq\train\02159e548029_01.jpg	02159e548029	data_lq\train_masks\02159e548029_01_mask.gif	02159e548029

- Unique car id is selected from the file names.
- Each car has 16 images taken at different angles.

For example:

00087a6bd4dc is the unique tag name for one car.

02159e548029 is the unique tag name for another car.

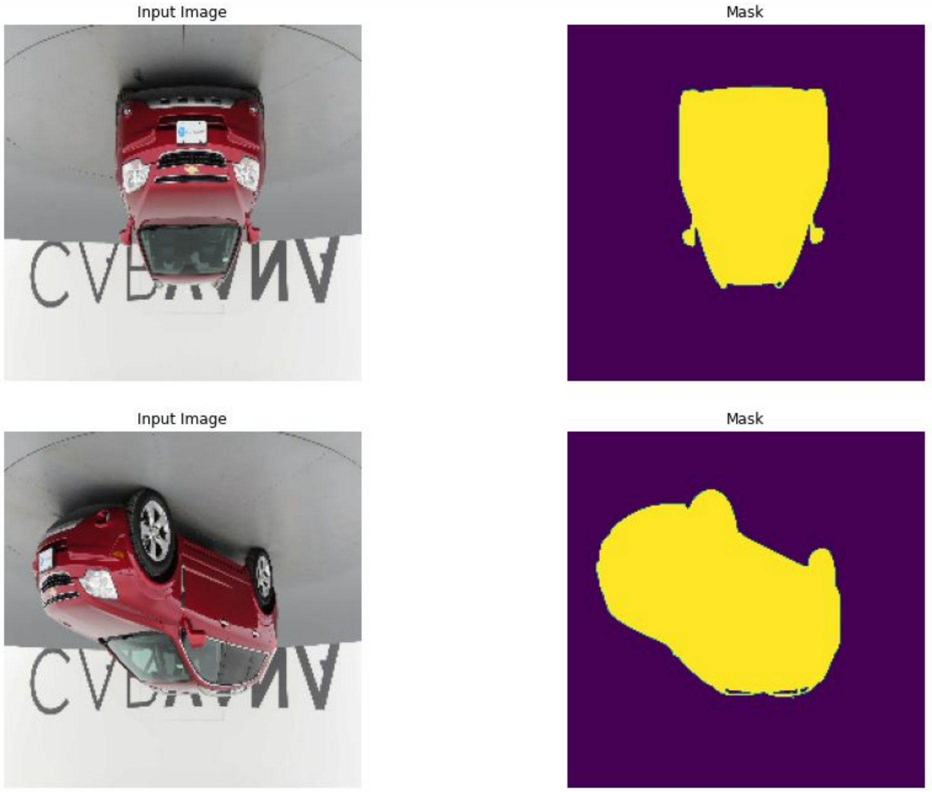
from_tensor_slices() function to create training and validation tensorflow datasets

Create training and validation tensorflow dataset

```
# create training and validation tensorflow dataset
# from_tensor_slices is used so that data can feed in batches
# from_tensors feed all of the data into the model and not able to split into batches
train_tf = tf.data.Dataset.from_tensor_slices((train_df['train_path'].values, train_df['mask_path'].values))
val_tf = tf.data.Dataset.from_tensor_slices((val_df['train_path'].values, val_df['mask_path'].values))
```

from_tensor_slices() function is used to create training and validation datasets. The datasets are tensorflow objects, which allows tensorflow to flow the data into the UNet CNN models later.

Examples of image augmentation



Images and masks are randomly flipped left / right and flipped up / down.

Build UNet CNN with transfer learning <u>no optimization</u> [MobileNetV2 + pix2pix]

- Downsampling encoding is built with layers from MobileNetV2 neural net.
- Upsampling decoding layers is built from upsampling layers from pix2pix model.
- Model is optimized for Tensorflow image segmentation tutorial but not for Carvana dataset. Output of the model is set to 3 layers.
- Model can only take in 224 x 224 resolution.

Downsampling encoder layers from MobileNetV2

```
input_shape = (224, 224, 3)
base_model = tf.keras.applications.MobileNetV2(input_shape= input_shape, include_top=False)

# Use the activations of these layers
layer_names = [                 # output shapes
     'block_1_expand_relu',    # (112, 112, 96)
     'block_3_expand_relu',    # (56, 56, 144)
     'block_6_expand_relu',    # (28, 28, 192)
     'block_13_expand_relu',   # (14, 14, 576)
     'block_16_project',       # (7, 7, 320)
]
downstack_layers = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs= downstack_layers)

down_stack.trainable = False
```

Trainable has been set to **False** to prevent the weights from getting updated during training.

```
[<KerasTensor: shape=(None, 112, 112, 96) dtype=float32 (created by layer 'block_1_expand_relu')>,
 <KerasTensor: shape=(None, 56, 56, 144) dtype=float32 (created by layer 'block_3_expand_relu')>,
 <KerasTensor: shape=(None, 28, 28, 192) dtype=float32 (created by layer 'block_6_expand_relu')>,
 <KerasTensor: shape=(None, 14, 14, 576) dtype=float32 (created by layer 'block_13_expand_relu')>,
 <KerasTensor: shape=(None, 7, 7, 320) dtype=float32 (created by layer 'block_16_project')>]
```

Output shapes from downstack layers.

Upsampling decoder layers from model created in TensorFlow pix2pix tutorial

```
# upsampler function is using a Conv2DTranspose with strides = 2,2 to half the size of the image

up_stack = [                        # output shapes
    pix2pix.upsample(512, 3),  # (14, 14, 512)
    pix2pix.upsample(256, 3),  # (28, 28, 256)
    pix2pix.upsample(128, 3),  # (56, 56, 128)
    pix2pix.upsample(64, 3),   # (112, 112, 64)
]
```

- The decoder will be the upsample block that is already implemented Pix2Pix in TensorFlow Examples.
- pix2pix.upsample is a function from the model built in tensorflow example.
- The number of filters and filter shape is optimized for the tensorflow image segmentation tutorial.

Unoptimized Model Structure

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
model (Functional)              [(None, 112, 112, 96 1841984     input_2[0][0]
__________________________________________________________________________________________________
sequential (Sequential)         (None, 14, 14, 512)  1476608     model[0][4]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 14, 14, 1088) 0           sequential[0][0]
                                                                 model[0][3]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 28, 28, 256)  2507776     concatenate[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 28, 28, 448)  0           sequential_1[0][0]
                                                                 model[0][2]
__________________________________________________________________________________________________
sequential_2 (Sequential)       (None, 56, 56, 128)  516608      concatenate_1[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 56, 56, 272)  0           sequential_2[0][0]
                                                                 model[0][1]
__________________________________________________________________________________________________
sequential_3 (Sequential)       (None, 112, 112, 64) 156928      concatenate_2[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 112, 112, 160 0           sequential_3[0][0]
                                                                 model[0][0]
__________________________________________________________________________________________________
conv2d_transpose_4 (Conv2DTrans (None, 224, 224, 3)  4323        concatenate_3[0][0]
==================================================================================================
Total params: 6,504,227
Trainable params: 4,660,323
Non-trainable params: 1,843,904
__________________________________________________________________________________________________
```

As the upsampling layers are not optimized, the concatenation layers are incorrect, leading to loss of information.

Based on the model, the layers that are concatenated must have the same depth.

Here, the both parts of the concatenated layers do not have the same depth as the upsampler code is not tweaked.

Based on the final output layer shape : (7, 7, 320)

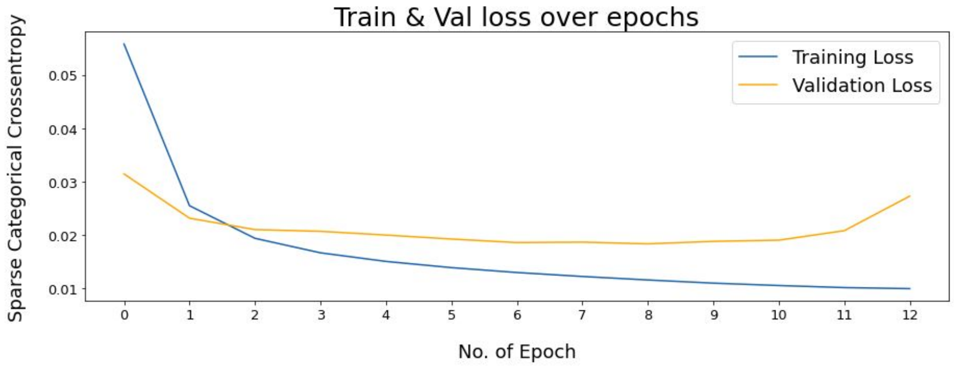
Downsample output = (14,14, 576)
Upsample output = (14,14, 512)
concatenate : (14,14, 576) + (14,14, 512) = (14, 14, 1088)
should be (14,14, 576) x 2 = (14,14, 1152)

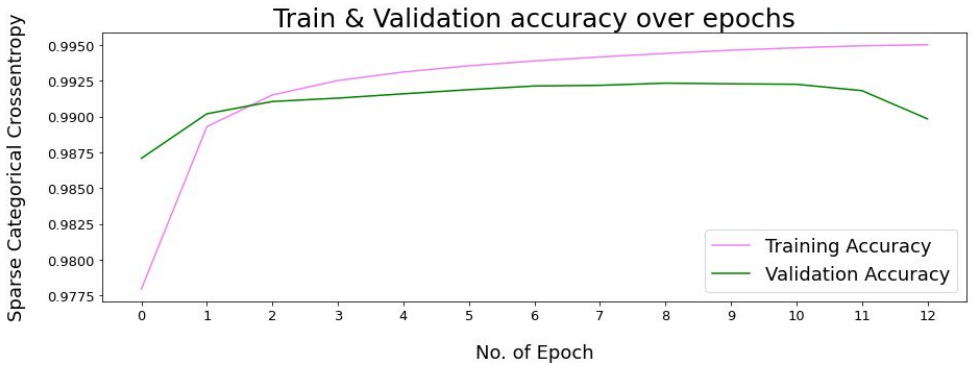
Downsample output = (28,28, 192)
Upsample output = (28,28, 256)
concatenate_1 : (28,28, 192) + (28,28, 256) = (28, 28, 448)
should be (28,28, 192) x 2 = (14,14, 384)

Downsample output = (56,56, 144)
Upsample output = (56,56, 128)
concatenate_2 : (56,56, 144) + (56,56, 128) = (28, 28, 272)
should be (56,56, 144) x 2 = (14,14, 288)

Downsample output = (112,112, 96)
Upsample output = (112,112, 64)
concatenate_3 : (112,112, 96) + (112,112, 64) = (112, 112, 160)
should be (112,112, 96) x 2 = (112,112, 192)

Model is not overfitted

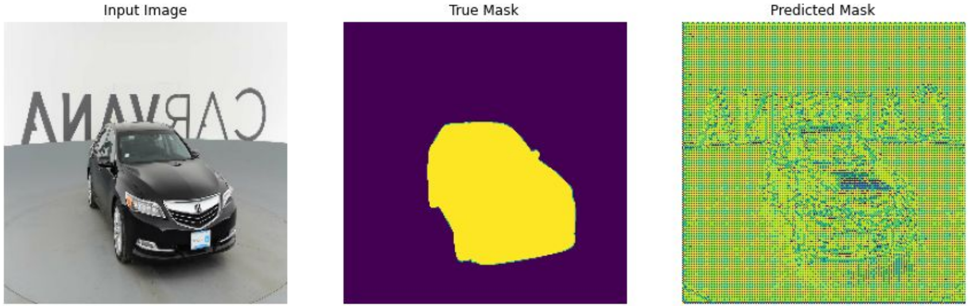




- There is early stopping in the model algorithm.
- The epoch with the best score is at epoch 9. The model is also reverted to the best weight.
- Train and validation scores are quite close.

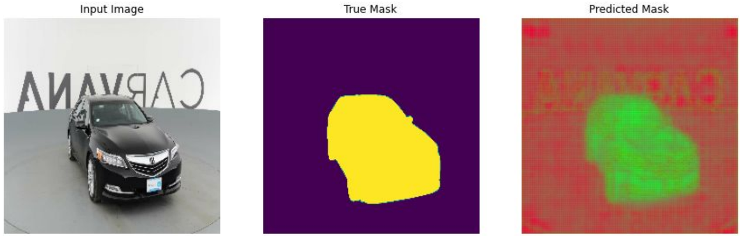
Model Results (model_tl1)

Before training model



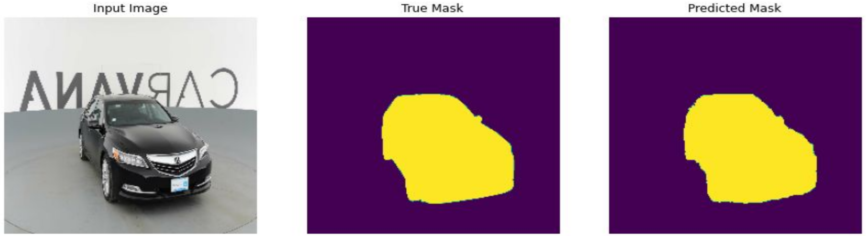
After training model

Model predicts for 3 masks as output channel is set to 3. Last layer of model, which Transpose conv2D layer has been set to 3.

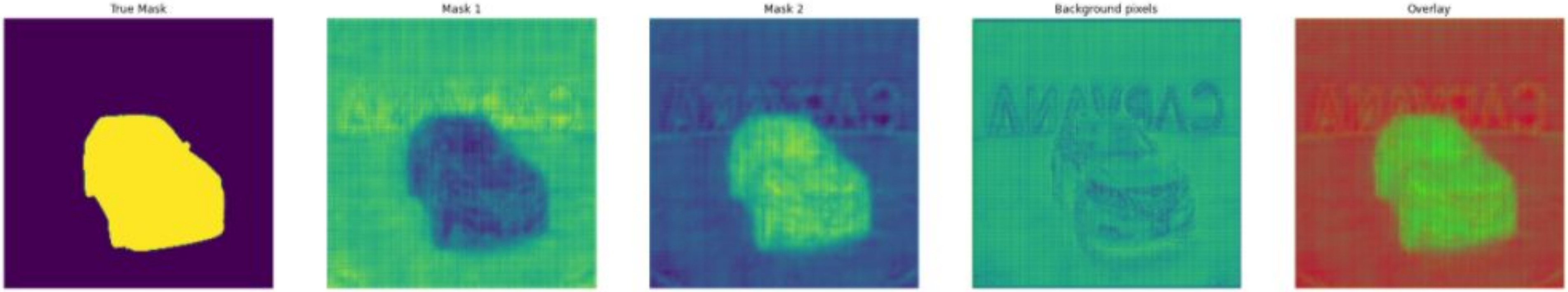


Create mask from the output layers by selecting for the last axis, where pixel values are either 0 (black) or 1 (white).

By using values from this code, ultimately it will indicate which pixel is black and which is white, hence creating a 1 channel mask.



Pixels not classified properly due to no activation function in Conv2D output layer



- The activation of the filter in the Conv2DTranspose resulted in values not scaled between 0 and 1, although the pixel values are scaled between 0 and 1 during data processing.
- By looking at the individual layers / channels from the image, it can be seen clearly the pixels are not classified correctly due to no activation specified for Conv2D in the last layer.
- Mask 1, Mask 2 and Background pixels channels are isolated from the Overlay.
- This will not affect the mask predicted as it is creating from the last layer.

Build UNet CNN with transfer learning <u>and optimization</u> [MobileNetV2 + pix2pix]

- Downsampling encoding is built with layers from MobileNetV2 neural net. This is unchanged.
- Upsampling decoding layers is built from upsampling layers from pix2pix model. Layers will be changed in the upsampling code
- Model will be optimized for Carvana dataset. Output of the model is set to 3 layers.
- Model can only take in 224 x 224 resolution.

Tweak Upsampling decoder layers

Re-define upsampler decoder

- modify the previous decoder so that the depth matches during concatenation

```
up_stack_redef = [                    # shape of layer to upsample = (7,7,320)

    # re-define arguments so that it matches output shape to the left
    pix2pix.upsample(576, 3),  # (14, 14, 576)
    pix2pix.upsample(192, 3),  # (28, 28, 192)
    pix2pix.upsample(144, 3),  # (56, 56, 144)
    pix2pix.upsample(96, 3),   # (112, 112, 96)
]
```

pix2pix.upsample(**512**, 3) => (576 , 3)

pix2pix.upsample(**256**, 3) => (192 , 3)

pix2pix.upsample(**128**, 3) => (144 , 3)

pix2pix.upsample(**64**, 3) => (96 , 3)

The pix2pix upsample filter layers are tweaked so that both parts of the concatenated layers will have the same depth.

Unoptimized Model Structure

```
Model: "model_4"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
model (Functional)              [(None, 112, 112, 96 1841984     input_5[0][0]
__________________________________________________________________________________________________
sequential (Sequential)         (None, 14, 14, 576)  1661184     model[3][4]
__________________________________________________________________________________________________
concatenate_12 (Concatenate)    (None, 14, 14, 1152) 0           sequential[3][0]
                                                                 model[3][3]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 28, 28, 192)  1991424     concatenate_12[0][0]
__________________________________________________________________________________________________
concatenate_13 (Concatenate)    (None, 28, 28, 384)  0           sequential_1[3][0]
                                                                 model[3][2]
__________________________________________________________________________________________________
sequential_2 (Sequential)       (None, 56, 56, 144)  498240      concatenate_13[0][0]
__________________________________________________________________________________________________
concatenate_14 (Concatenate)    (None, 56, 56, 288)  0           sequential_2[3][0]
                                                                 model[3][1]
__________________________________________________________________________________________________
sequential_3 (Sequential)       (None, 112, 112, 96) 249216      concatenate_14[0][0]
__________________________________________________________________________________________________
concatenate_15 (Concatenate)    (None, 112, 112, 192 0           sequential_3[3][0]
                                                                 model[3][0]
__________________________________________________________________________________________________
conv2d_transpose_7 (Conv2DTrans (None, 224, 224, 3)  5187        concatenate_15[0][0]
==================================================================================================
Total params: 6,247,235
Trainable params: 4,403,235
Non-trainable params: 1,844,000
__________________________________________________________________________________________________
```

After tweaking the upsampler layers in the code, the concatenated layers are correct.

Based on the final output layer shape : (7, 7, 320)

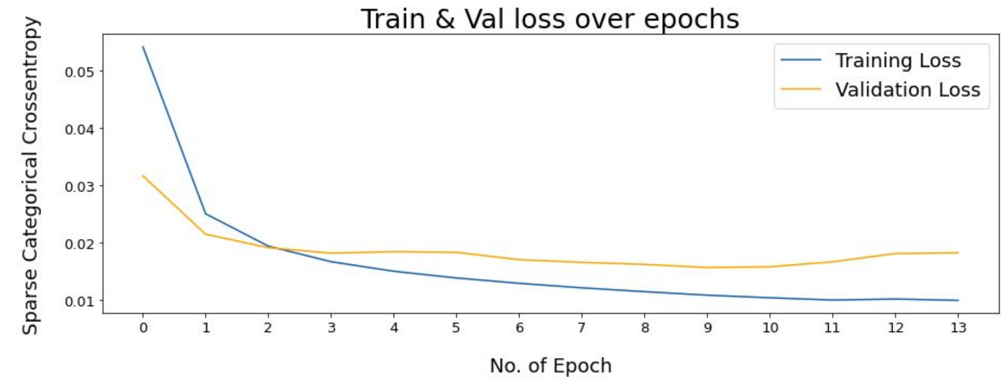
Downsample output = (14,14, 576)
Upsample output = (14,14, **576**)
concatenate : (14,14, 576) + (14,14, 576) = (14, 14, 1152)

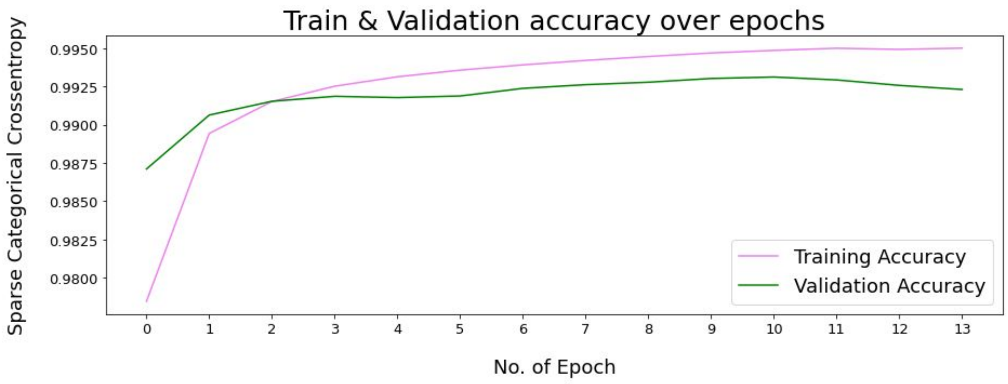
Downsample output = (28,28, 192)
Upsample output = (28,28, **192**)
concatenate_1 : (28,28, 192) + (28,28, 192) = (28, 28, 384)

Downsample output = (56,56, 144)
Upsample output = (56,56, **144**)
concatenate_2 : (56,56, 144) + (56,56, 144) = (28, 28, 288)

Downsample output = (112,112, 96)
Upsample output = (112,112, **96**)
concatenate_3 : (112,112, 96) + (112,112, 96) = (112, 112, 192)

Model performance improved after optimization

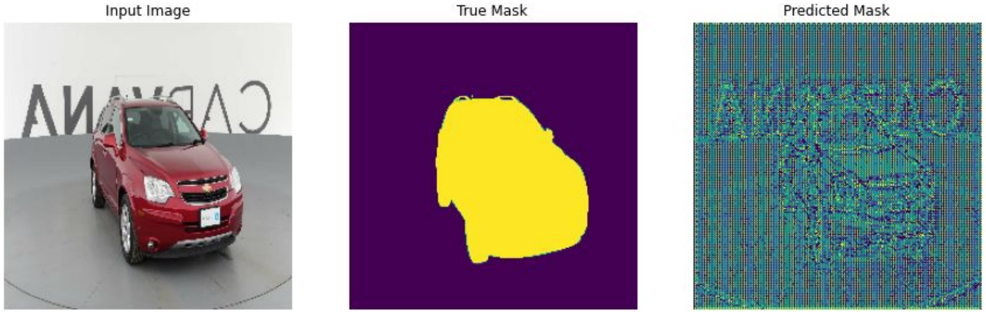




- The accuracy and low has narrowed between the train and validation compared to without optimization.
- There is early stopping in the model algorithm.
- The epoch with the best score is at epoch 10. The model is also reverted to the best weight.
- Train and validation scores are closer than that in the unoptimized model.

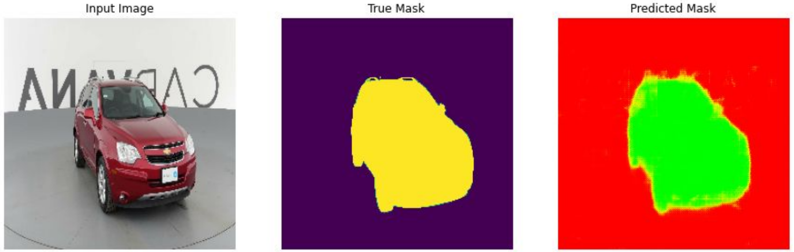
Model Results (model_tl2)

Before training model



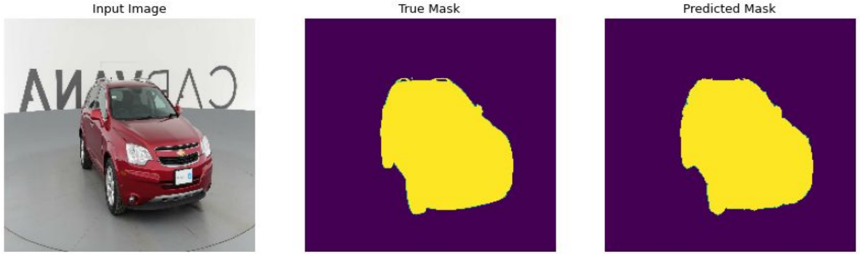
After training model

Model predicts for 3 masks as output channel is set to 3. Last layer of model, which Transpose conv2D layer has been set to 3.

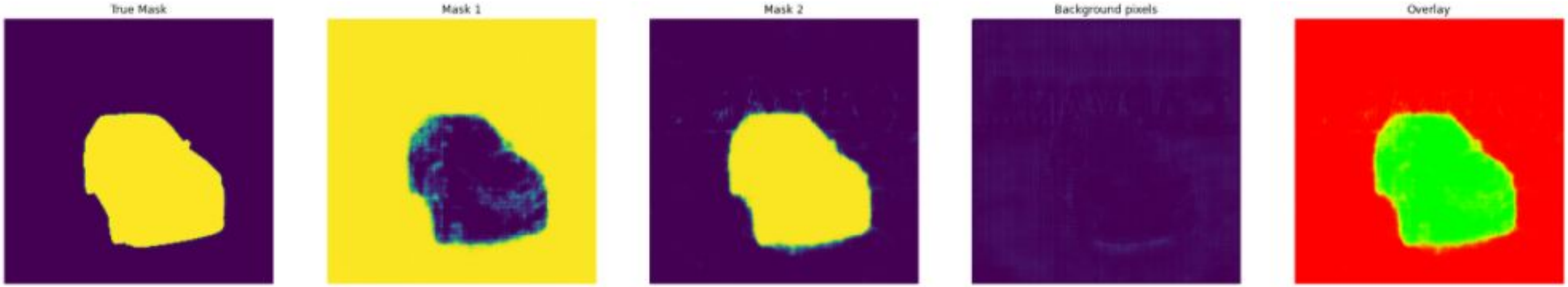


Create mask from the output layers by selecting for the last axis, where pixel values are either 0 (black) or 1 (white).

By using values from this code, ultimately it will indicate which pixel is black and which is white, hence creating a 1 channel mask.



Pixels are classified properly due to sigmoid activation function in Conv2D output layer



- The activation function is specified for the Conv2DTranspose in the last layer. Even with the activation of the filter, the activation function in the Conv2DTranspose will convert the values between 0 and 1, as shown above.
- After training, the pixels are classified according to the mask label. Pixels classified as red is the background (0) and pixels classified as green (1) is the signal. Yellow is the overlap of red and green at equal intensity. There are 2 classes of pixel, 0 and 1 in the mask but the output from the model is 3.
- From here, we are able to see there are some overlaps between the pixels labeled as background and pixels labeled as the mask of the car. There are 2 mask layers overlapping.
- Mask 1, Mask 2 and Background pixels channels are isolated from the Overlay.
- In Mask 1, there are some background pixels misclassified as signal. In Mask 2, there are some signal pixels misclassified as background.
- From the overlay, it can be seen Mask 1 is less accurate than Mask 2 is less accurate than Mask 1, as more of the pixels (yellow) are misclassified as background. However, Mask 2 is more defined than Mask 1.
- This will not affect the mask predicted as it is creating from the last layer.

Build UNet CNN with transfer learning and optimization [VGG19 + pix2pix]

- Downsampling encoding is built with layers from VGG19 neural net.
- Upsampling decoding layers is built from upsampling layers from pix2pix model. Layers will be changed in the upsampling code.
- Model is optimized for Carvana dataset, but it is not trained due to limited computational power. Training through the first epoch is estimated to be 1.5 hours, and the following epoch will take even longer to train due to backward propagation.
- The input shape for the model is set at (512, 512, 3). A higher resolution input leads to higher resolution upscaling as well, giving a better quality mask.
- Model can take in image resolution with more than 512 x 512.

Why use highest resolution image to predict for mask

1) It is best to train based on the highest resolution image to predict for the highest resolution mask.
2) A high resolution image contains more pixel data than a low resolution image. A model needs as much data as possible to learn the maximum features from the image. If a low resolution image is used as the training image, with the lack of data the predicted mask will be more inaccurate.
3) The Kaggle submission requires submission of mask with resolution of 1912 x 1280, as a submission of mask with 224 x 224 resolution yields a score of 0.
4) When the 224 x 224 mask is resized to 1912 x 1280 resolution with the “nearest neighbor” algorithm, the score yielded is around 0.52.
5) However, training time has to be balanced with feasibility if there is time constraint.

Model Summary

- Total Parameters to train for UNet CNN VGG19 + pix2pix with optimization

```
_____________________________________________________________________________________________
conv2d_transpose_6 (Conv2DTrans (None, 512, 512, 3)  6915        concatenate_2[0][0]
=============================================================================================
Total params: 21,364,035
Trainable params: 8,415,491
Non-trainable params: 12,948,544
```

- Training time for initial epoch

```
Epoch 1/20
  2/508 [..............................] - ETA: 1:30:05 - loss: 1.2594 - accuracy: 0.3911
```

Training time for initial epoch is around 1.5hours on a local machine. Following epochs will take even longer to train due to backward propagation.

Build UNet CNN from scratch

- Downsampling encoding and Upsampling decoding layers are built from scratch.
- Model is optimized for Carvana dataset, but it is not trained due to limited computational power. Training through the first epoch is estimated to be 30 minutes, and the following epoch will take even longer to train due to backward propagation.
- The input shape for the model is set at (224, 224, 1).

Model Summary

- Total Parameters to train for UNet CNN built from scratch with input resolution of 224 x 224

```
conv2d_22 (Conv2D)              (None, 224, 224, 1)  33          activation_21[0][0]
==================================================================================
Total params: 9,325,601
Trainable params: 9,318,945
Non-trainable params: 6,656
```

- Training time for first epoch

```
Epoch 1/20
  2/508 [..............................] - ETA: 30:30 - loss: 0.6961 - accuracy: 0.6091
```

Training time for [MobileNetV2 + pix2pix] with optimization

```
Epoch 1/20
508/508 [==============================] - 713s 1s/step - loss: 0.1042 - accuracy: 0.9585 - val_loss: 0.0316 - val_accuracy: 0.
9871
```

- Compared to UNet CNN with transfer learning, UNet CNN built from scratch has a much longer training time.
- This is due to the layers are already pre-trained in the neural net used in transfer learning.
- This speeds up the training process.

Comparison between VGG19 + pix2pix vs UNet CNN build from scratch with input resolution of 224 x 224

Total Parameters to train for VGG19 + pix2pix 512 x 512

```
conv2d_transpose_6 (Conv2DTrans (None, 512, 512, 3)
==================================================
Total params: 21,364,035
Trainable params: 8,415,491
Non-trainable params: 12,948,544
```

Total Parameters to train for UNet CNN build from scratch 224 x 224

```
conv2d_22 (Conv2D)              (None, 224, 224, 1)
=====================================================
Total params: 9,325,601
Trainable params: 9,318,945
Non-trainable params: 6,656
```

- Usually, higher resolution images take much longer time to train as more information is fed into the model.
- Even at a higher resolution, there are less parameters to train using transfer learning to train than from neural nets built from scratch.
- Transfer learning is good for feature extraction and reducing training time. However, it also depends on what the pre-trained model is trained on. For example, VGG19 is trained on 1 million images consisting of a variety of images and able to classify into 1000 classes with around 95% accuracy.
- A pre-trained model which is specifically train in faces may not be suitable for use in this Carvana challenge.

Run-length encode (rle) predicted masks

1) Test data, consisting a total of 100,064 images (.jpg) with 1912 x 1280 resolution is flowed into a tensorflow dataset.

2) Only 2 models are used to predict the masks for test data.

3) UNet CNN [MobileNetV2 + pix2pix] with optimization and without optimization

4) The masks are encoded with rle and exported as a .csv file tagged to the image file names.

Submit rle masks generated from predicted images to Kaggle in as .csv

3 submissions for Yong_Tat_Koh   Sort by Most recent

All   Successful   Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submit_224×224_resize_nearest_v2pix2pix_optimized.csv 4 minutes ago by Yong_Tat_Koh add submission details	0.51273	0.52283	☐
submit_224×224_resize_nearest.csv 4 days ago by Yong_Tat_Koh add submission details	0.51415	0.52456	☐
submit_224×224_110321.csv 6 days ago by Yong_Tat_Koh rle encode with +2	0.00000	0.00000	☐

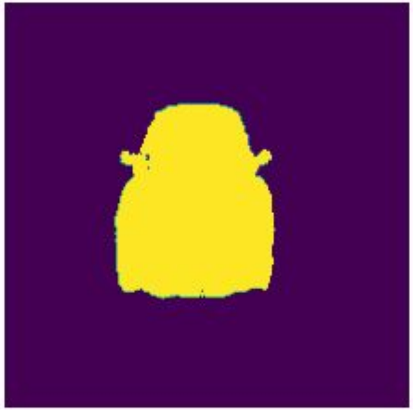
- The predicted masks in submit_224x224_110321.csv is run-length encoded (rle) at 224 x 224 resolution. The resizing algorithm used is "bilinear".
- The predicted masks in submit_224x224_resize_nearest.csv is rle at 224 x 224 resolution. The resizing algorithm used is "nearest". The trained model used is [MobileNetV2 + pix2pix] no optimization .
- The predicted masks in submit_224x224_resize_nearest_v2pix2pix_optimized.csv is rle at 224 x 224 resolution. The resizing algorithm used is "nearest". The trained model used is [MobileNetV2 + pix2pix] **with optimization.**

Possible Reasons for low Kaggle score

- The score is low at around 0.52 because the data in 224 x 224 is used to calculate and estimate for the values when they are resized at 1912 x 1280.
- If the input is at the highest resolution of 1280 x 1280 (needs to be a square), based on the UNet CNN architecture it will lead to accurate true upscaling.
- Input needs to be a square for simplicity due to the Convolutional 2D layers that divide by 2 each time. A long dimension will lead to an odd number at some point. For example, 1912 x 1280 after 4 layers of Conv2D = 119.5 x 80
- There are workarounds, like cropping rectangular images to squares during image pre-processing. For simplicity, it is best to keep the image as a square.
- The score difference between the unoptimized and optimized model is small. The mask outputs from the unoptimized and optimized model are similar which explains for the the close score.

Conclusion

- The optimized MobileNetV2-pix2pix UNet CNN can be deployed as a prototype. It is able to generate a reasonably accurate mask from the test dataset.
- However, there is still room for improvement to create models that can predict for more accurate masks.



Predicted mask from 1 test image

Future Work

- It is my first project with Image Segmentation, there are various factors that could be further considered and tweaked.
- Weight initialization can be explored further to optimize the models and possibly reduce training time and increase accuracy.
- Different loss functions such as dice loss function or the combination of dice and binary cross entropy loss function can be tested to see if it improves the segmentation.
- Conv3D can also be experimented with.
- Another improvement that can be done is to search for platforms to train the models to reduce training time.

Thank you for your time!