Machine Learning Engineer Nanodegree

Capstone Project

Sankirna Joshi
September 23rd, 2017

Identify Car Boundary and Remove its background from an image

I. Definition

Project Overview

Carvana, a leading used-car sales company uses a revolving photo studio to click the images of the cars in their inventory from various angles. They then remove the photo studio background from their pictures and apply the car mask to different appealing backgrounds. The methodology that exists today for picture editing involves a lot of manual effort and requires considerable skill and time. Carvana has a huge inventory of cars and they want to automate this task. As, this problem belongs to the subdomain of computer vision called as image segmentation, I take up this challenge to generate the car masks using image segmentation techniques.

For this project, I created a deep learning model to generate the car mask. The model was trained on the dataset provided by Carvana on this Kaggle competition.

Problem Statement

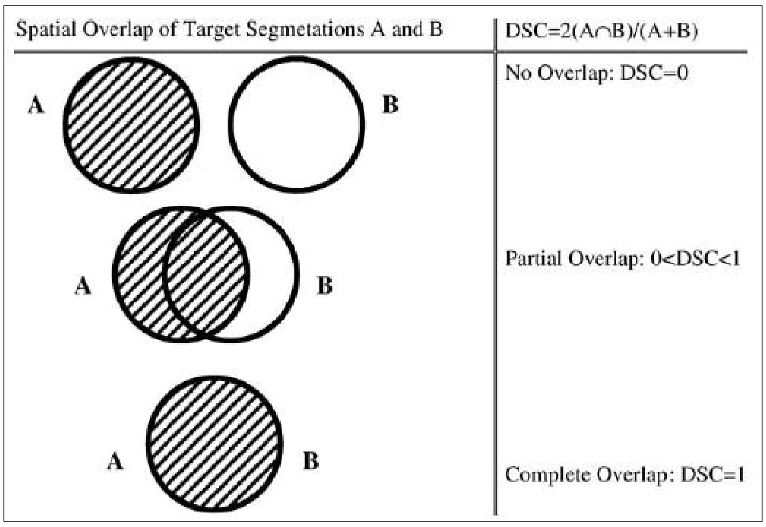
Background removal is currently a manual task or at best, a semi-automatic technique. We want to develop a deep learning algorithm that automatically removes the background from the car images by use of image segmentation techniques. Following steps will be performed.

- Download the dataset and preprocess the images.
- Train a deep learning classifier on this data that can generate the car mask.
- Submit the car masks to Kaggle and receive a decent score.

Metrics

In image segmentation, we segregate the image's pixels into multiple pixel groups called segments and check if every pixel is in its desired segment. Thus our metric should find out the overlap or intersection between the actual segmentation and desired segmentation. For this purpose, we will Dice coefficient given by:

$$\mathbf{DSC} = \mathbf{2} * \frac{|A \cap B|}{|A| + |B|}$$



A: predicted set of pixels

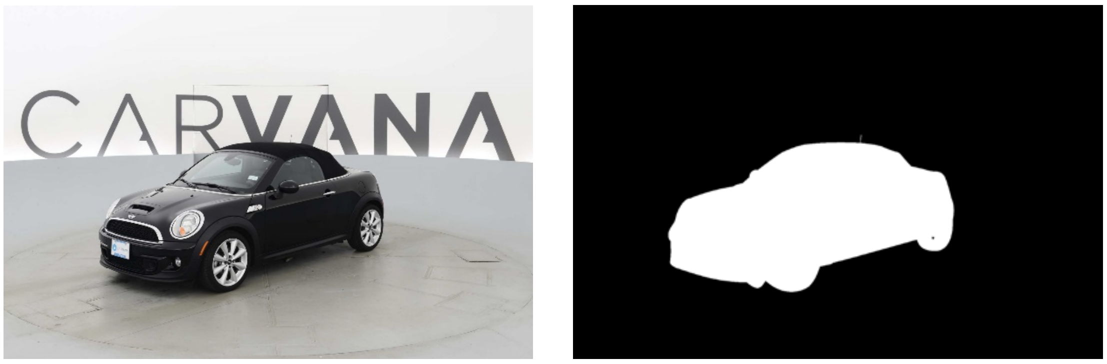
B: ground truth.

II. Analysis

Data Exploration

The Carvana dataset, is divided into two parts: train and test. The train dataset will be used to train and validate our model whereas the test dataset will be only used to submit predictions and make observations of our models performance.

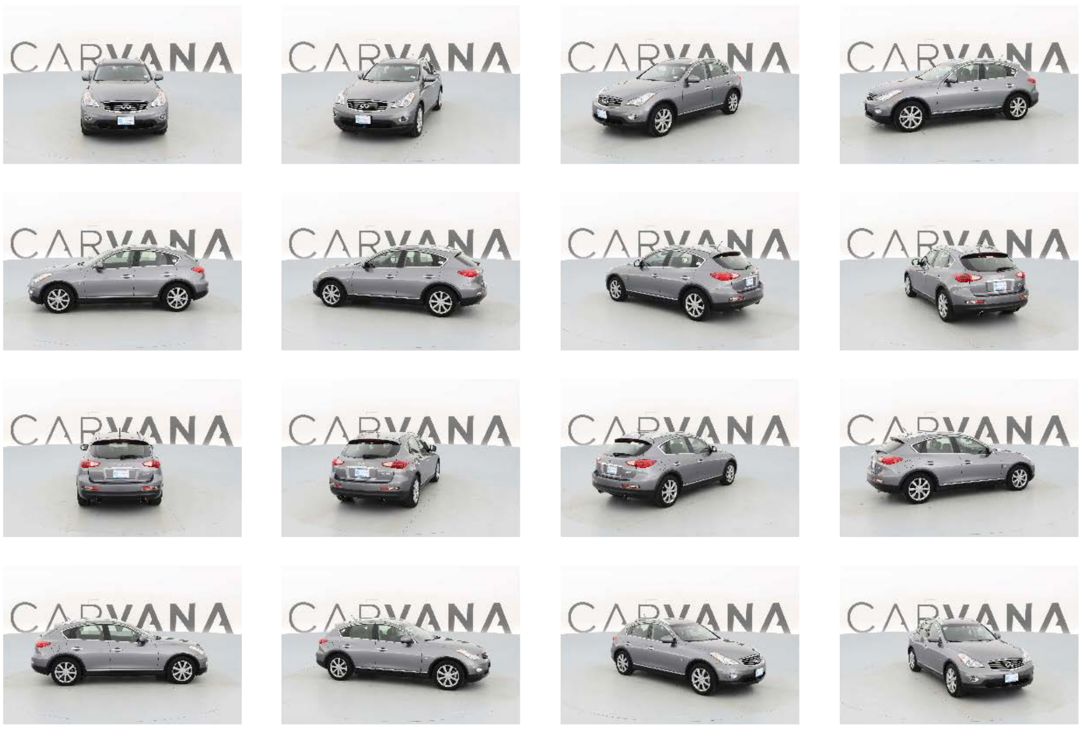
The training dataset consists of a total of 318 unique cars and the images of these cars are taken from 16 different angles, making the total count of images to be 5088. Thus, we have 5088 coloured unique car images and corresponding grayscale car mask images. (Example below)



Each image in the dataset is 1918*1280 pixels in size. The dataset also consists of a train_masks.csv file which contains the details of the train mask images in run length encoded format and another metadata.csv that contains information such as brand, make, model etc. Car images will be used as inputs to the model and car masks as a target.

Exploratory Visualization

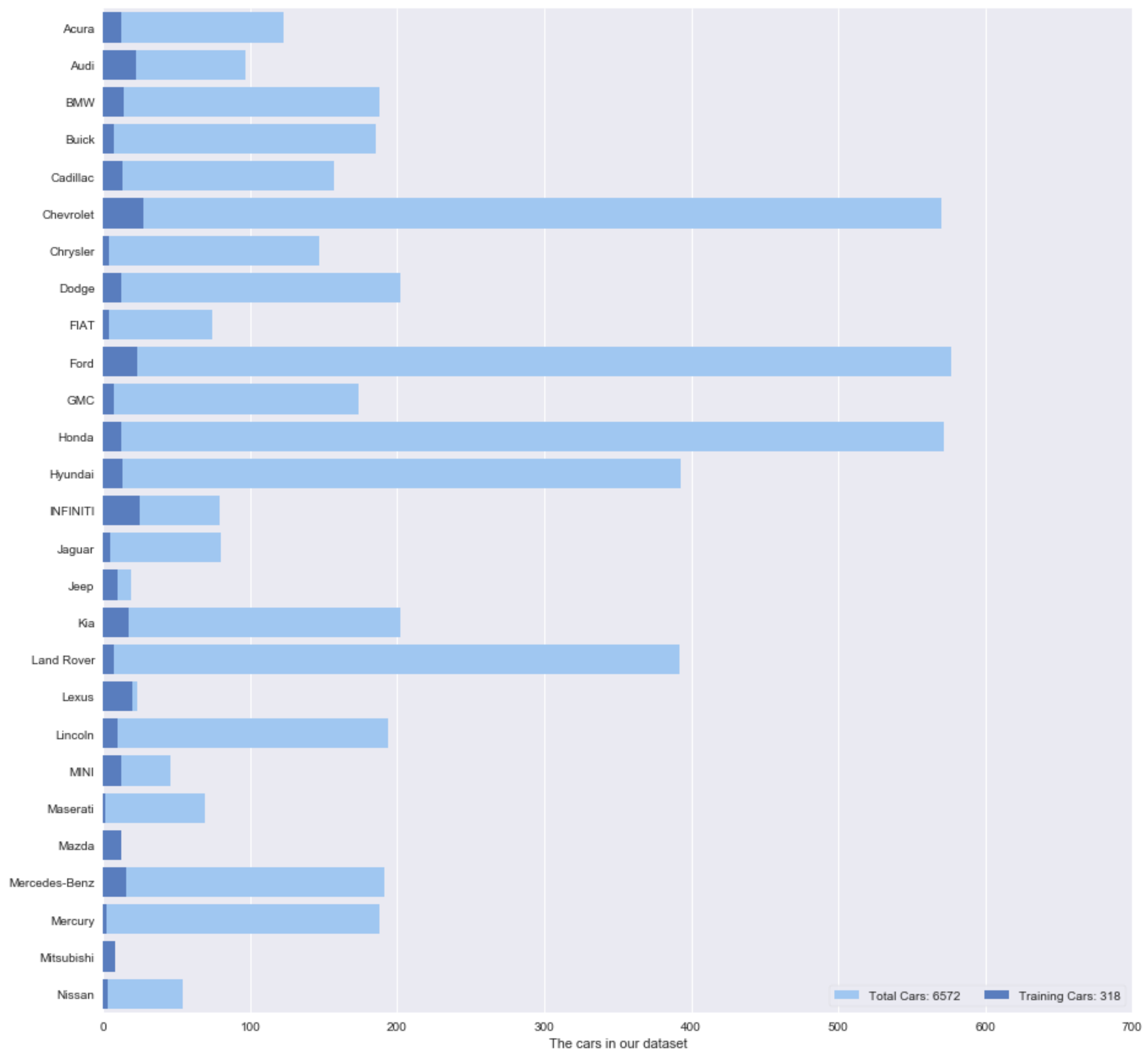
- Let us visualize all the 16 angles from which a car image is clicked.



- Let us visualize a few different cars on in the same angle.

We can notice a couple of things from this. The car images are in the center and take around 50% of the image. The cars cast a shadow which can make it difficult to see the edges on the bottom especially if the car is black.

- Let us see the distribution of cars on our data.



We can see that there are unequal distribution of cars in the datasets. However, it should not influence our model much as our model should be able to learn the car features(edges, wheels shape etc) regardless of the car make and brand and generate the mask.

Algorithms and Techniques

The algorithm used in this task is a modification of convolutional neural networks. It is called as fully convolutional densenets (FC-DenseNets) and are state of the art techniques in semantic image segmentation. CNNs lose the spatial features and resolution as we move down the network due to pooling and longer strides. As a result, they are suited for single predictions like the class of the image. For image segmentation tasks, we need to recover the spatial resolution as well as the finer details in the image as we need to classify on a per-pixel basis. FC-DenseNets enable this by adding upsampling to recover spatial resolution and skip connections to recover finer details.

The following parameters can be tuned to optimize the classifier:

- Image size (Better resolution images to capture finer details, more pixels, finer results)
- Batch Size (Images to use per step)
- Epoch (Length of training)
- Learning Rate (Speed to take update steps)
- Weight decay (Regularization to control weight update)
- Optimizer alogrithm (Choice of optimizer to update weights)

I will be carrying out the implementation of the The One Hundred Layers Tiramisu model. Our problem has a constant background (studio) and changing foreground objects (cars). Tiramisu model seems to be an appropriate candidate to try as this model got state-of-the-art results in image segmentation on CamVid dataset in a similar environmental setting.

The Tiramisu model architecture is shown in the adjacent image.

It consists of Dense blocks which is a concatenation of previous feature maps and are placed on both down-sampling and up-sampling paths. In the down-sampling path, the input to Dense block is concatenated with the output whereas in up-sampling it is not.

Skip connections are used to compensate the resolution loss due to pooling.

The model is trained on below parameters:

- Loss function: dice loss (1 – Dice Coefficient)
- Optimizer: RMSprop
- Learning Rate: 0.0001 (Reducing learning rate)
- Metrics: Dice Coefficient

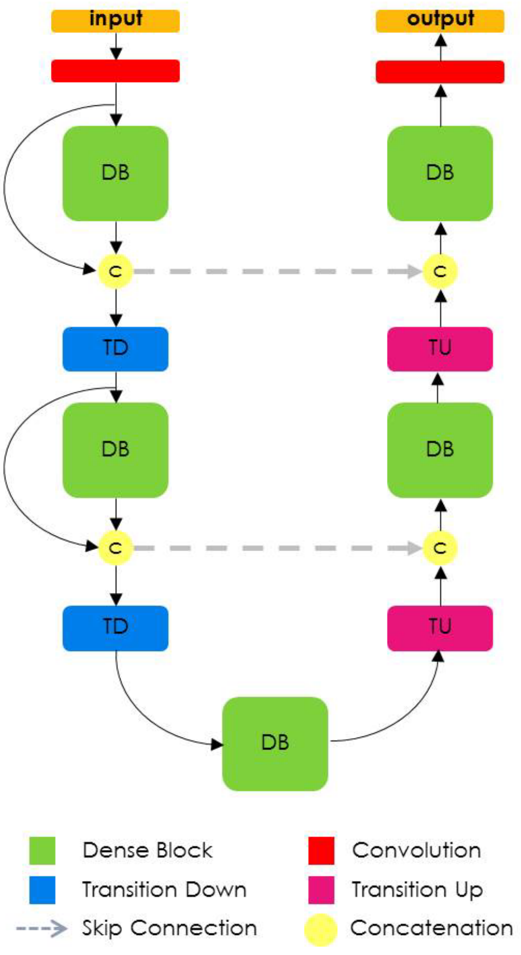


Fig 1. FC-DenseNet

Benchmark

The benchmark used for this model is the mean dice score of 0.985 achieved on the same test data shared in the discussions here. This benchmark is chosen based on the following aspects:

- The architecture used for this benchmark model is a custom U-Net which is a practical and proven image-segmentation model unlike VGG-16 or other classification models.
- This model and score are publicly shared so that we can validate them.
- I can compare the results from my solution directly on the same data, thus proving to be a valid benchmark score.

PS: In the proposal doc, I had suggested that I would use as a benchmark, a contemporary classification model like VGG-16 and modify it to make suitable for the task. But I got very poor dice score on the model. As a result I decided to consider a new benchmark and I chose the above benchmark to get a direct comparison to an actual working image segmentation model.

III. Methodology

Data Pre-processing

I performed the below pre-processing steps on the data:

- Split the dataset into train-test split in (80:20) ratio.
- Change train mask images from gif format to png format to work with cv2.
- Resize the all images and masks to (256,256) size and normalize the pixel values.

Implementation

The FC-DenseNet model was implemented using Keras functional API. The model consists of 3 parts:

- Downsampling path consisting of DenseBlocks (DB) and Transition down layers (TD).
- Bottleneck DensebBlock
- Upsampling path consisting of DensebBlocks and Transition up layers (TU).

The following tables summarize the model:

Architecture	
Layers	**Filters**
Input Image	m = 3
3 x 3 Convolution	m = 48
DB (4 layers) + TD	m = 112
DB (5 layers) + TD	m = 192
DB (7 layers) + TD	m = 304
DB (10 layers) + TD	m = 464
DB (12 layers) + TD	m = 656
DB (15 layers)	m = 880
TU + DB (12 layers)	m = 1072
TU + DB (10 layers)	m = 800
TU + DB (7 layers)	m = 560
TU + DB (5 layers)	m = 368
TU + DB (4 layers)	m = 256
1 x 1 Convolution	m = 1
Sigmoid	

Layer
Batch Normalization
ReLU
3×3 Convolution
Dropout $p = 0.2$

The structure of layer in a DB

Transition Down (TD)
Batch Normalization
ReLU
1×1 Convolution
Dropout $p = 0.2$

The structure of a TD layer

Transition Up (TU)
3×3 Transposed Convolution $stride = 2$

The structure of a TU layer

Fig 2: Model Architecture

The 'Capstone.ipynb' Jupyter Notebook consists of the above mentioned blocks inside the functions dense_block(), transition_down() and transition_up() and the subsequent code cells contain the building of the actual model.

The model was initialized using the 'he_uniform' and trained using RMSprop. Learning rate of 1e-4 was used to train the model. Validation set was used to EarlyStop the training with the patience of 8 epochs. The model was regularized with a weight decay of 1e-4 and dropout of 0.4.

Dice coefficient was used as the metric and Dice loss as the loss function. The model was fit using a fit_generator method so that all both the training and the validation images are loaded into arrays in batches by the train_generator and valid_generator respectively and then fed to the model. Batch size of 4 was used.

Refinement

Initial training achieved a mean dice score of 0.9926 when the model was run for 10 epochs with no weight decay and dropouts.

This score was improved to 0.9944 by using the below techniques:

- Reducing learning rate by factor of 0.1 if there was no reduction in validation dice loss even after 4 consecutive epochs.
- Adding 20% dropout after every convolution so that 20% of weights were randomly dropped
- Adding weight decay.

All this led to a slight improvement in the overall score. The final model score is 0.9944 which is based on about (25%) of the test data as per kaggle rules.

IV. Results

Model Evaluation and Validation

The FC-DenseNet model performs very well on new unseen data. The model solution was tested over 100064 images and it got a mean dice score of 0.9944 over a random 25% of the data. This means that the model could on average correctly mask over 99.44% of the car image which is quite reasonable considering that the model was trained on smaller (256x256) images and scored on larger (1918x1280) images.

This model takes images of (256, 256, 3) shape and outputs a mask image of shape (256, 256, 1). The input image goes through multiple DenseBlocks and Transition Down units in the down path. The up path consists of Transition Up layers followed by DensBlocks. (Please refer Fig 2) Finally before the output we apply a sigmoid function. The model can take an arbitrary size image and can be recompiled to train on images of different sizes if needed. The skip connections and the concatenations allow the model to be very deep without an exponential growth of parameters. Also, due to concatenation, the flow of information is easy between layers and spatial information is preserved to some extent as required by our problem. The model is very effective to identify finer details in the images such as (gaps in tyres, exhaust silhouette, gaps in the image grills etc) and produces outputs that are on par with our expectations(Please refer Fig 3 for reference.) As the model's inputs as well as outputs will always be images shot from the Carvana studio I did not need to perform sensitivity analysis on this model as it is expected that the model will always have reliable inputs.

Justification

The benchmark model got a mean dice score of 0.985 whereas my model got a score of 0.9944. This is definitely an improvement over the benchmark score. The results as seen in fig. 3 can be used to justify that the model works well to unseen data as well as works as expected. The model will work reliably if used on lower size images so that the coarse background information will be lost. However, for using it for large resolution images, we would need to use better deep learning hardware to train the model on higher size input images or even full size input images which was not permitted for me due to the hardware restrictions.

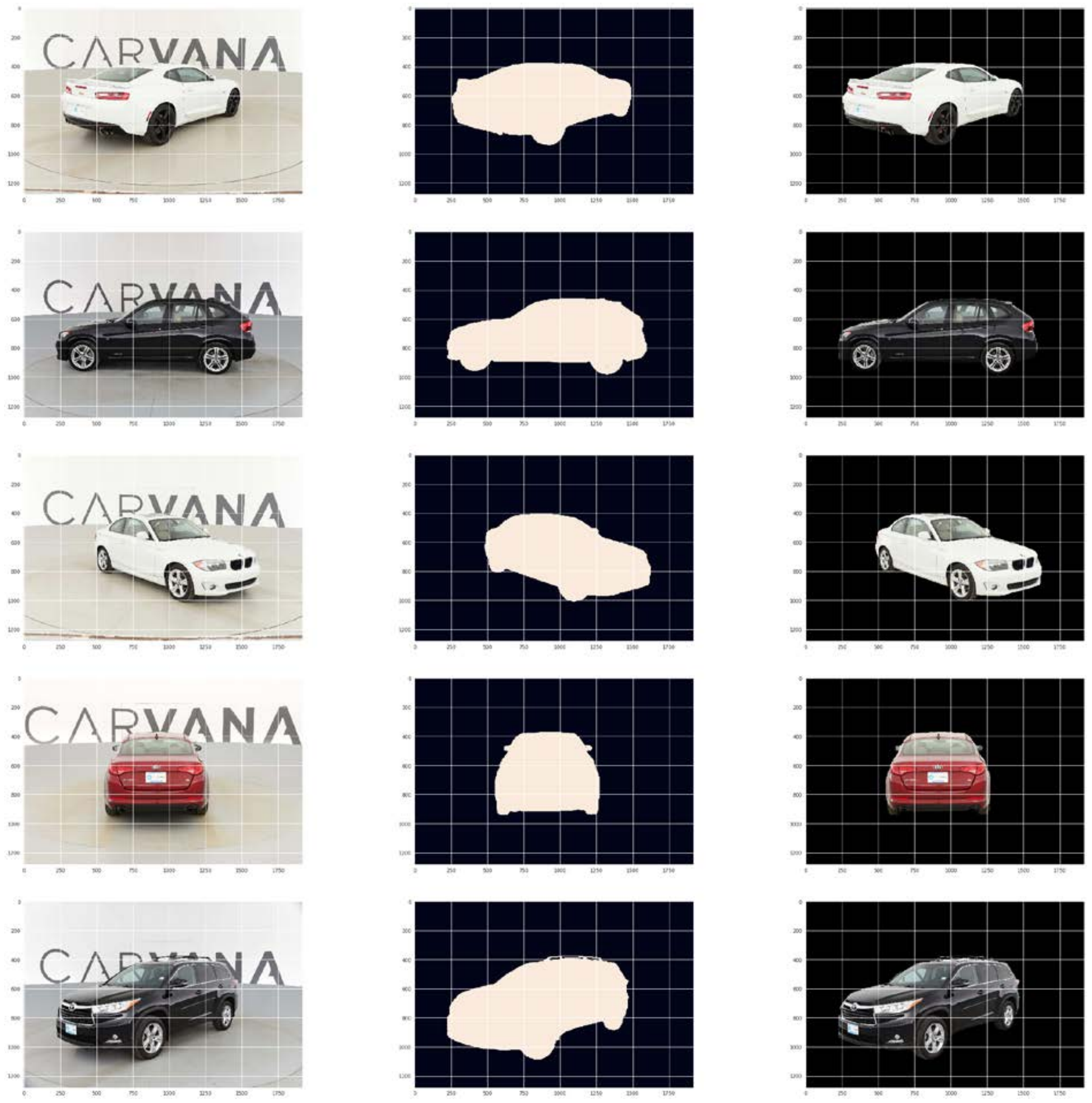
V. Conclusion

Free-Form Visualization

From the below results as seen in fig 4, we can confidently say that the results are promising.

Some of the important details from our results are noted below

- The mask generated is coarse at the mask boundaries.
- The model could identify the important shapes like side-view mirrors, wheels, spoilers, grills etc of the car well.
- The model could identify created masks with gaps in the right places such as the gap in wheels etc.



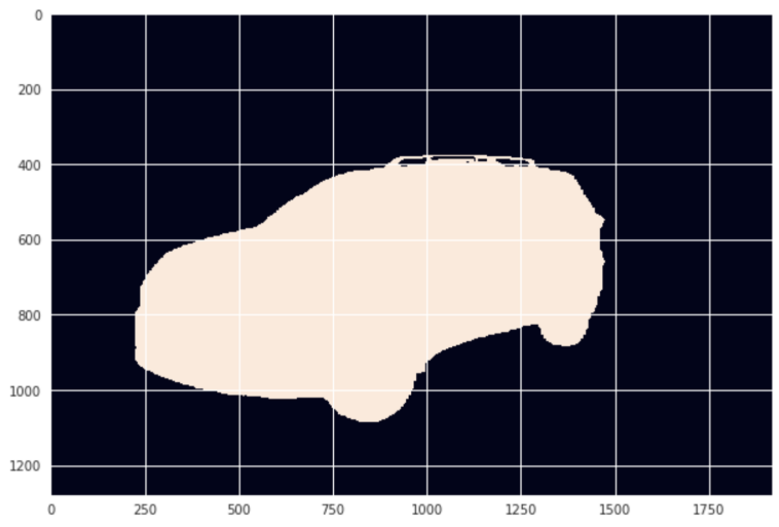


Fig 3 Image Mask Results

First row contains test images

Second row contains model outputs

Third row contains the masked output.

Reflection

The process used for this project can be summarized using the following steps:

1. A challenging problem was found to be solved.
2. The data was downloaded and preprocessed.
3. A suitable benchmark was identified for the image segmentation model.
4. A classification model was built from scratch.
5. The classifier was trained using batch process in an iterative manner until the best parameter values were found.
6. The model was used to predict the output mask for the test images in a batchwise manner and output was saved into a submission file.
7. The file was uploaded to Kaggle to calculate the mean dice score over the images.
8. The model was tested on the test images by applying the mask and results were discussed.

The step 4 and 5 were somewhat challenging. It involved learning the new Keras functional model in order to create my FC-DenseNet which is like a u-shaped model. The skips and concatenate connections were complicated and the multiple deep layers made the model difficult to create. Also, my RAM could not handle the array of images at once and so I had to manage them using a generator.

Overall, I am very pleased at the results and also happy to be involved in a Kaggle competition. Also, I'm happy that the project involved cars somehow and would like to see how this model would work on an outdoor car image.

Improvement

The two things I would definitely like to improve are the smoothness of the mask boundaries and decrease the training time for the model. Currently on the hardware available at my disposal, the model took around 40 minutes to train per epoch! Even an image size of 32x32 takes around 500 sec per epoch to train. It would be interesting to play with different optimizers and their learning rates to see I can manage to improve the training time. The depth of the model can be changed as well, but the results from experiments here suggest that the model benefits from deeper layers and more parameters. As discussed before, I would like to increase the input size to the model. I firmly believe that the model will produce very smooth results for input size of around (1024, 1024).