RICE UNIVERSITY COMP340

Auto Segmentation

Will LeVine & Cole Morgan

December 1, 2018

Abstract

We present a unique pipeline to segment individual automobiles in photographs, and to perform additional feature tagging. The pipeline consists of aggressive downsampling to limit computing power requirements, followed by a binarization of pixels represented by several layers of linear and non-linear features. This method results in a train F1 of .966 and a test F1 of .966. Using predicted binary masks for these images, we present color tagging as an example of feature tagging. This method consists of selecting those pixels which were predicted foreground, pipelined with DBSCAN clustering on RGB ratio features.

Contents

1 Introduction **3**

1.1 Dataset Examples . . . 3

1.2 Formal Description of Task . . . 3

1.3 Segmentation Metric . . . 3

2 Methods **4**

2.1 Downsampling . . . 4

2.2 U-Net as a Learned Binarizer . . . 4

2.2.1 General U-Net Architecture . . . 4

2.2.2 Our U-Net Architecture . . . 4

2.2.3 Why This Architecture? . . . 5

2.2.4 Hand Designed Features . . . 5

2.2.5 Shallow Models . . . 6

2.3 Color Clustering . . . 6

2.3.1 RGB Ratio Feature Extraction . . . 6

2.4 Lessons Learned . . . 6

A Appendix **8**

A.1 Car Image Examples . . . 8

A.2 Image Mask Examples . . . 8

A.3 Downsampling Example . . . 8

A.4 Training Accuracy and Loss During Training . . . 8

A.5 U-Net Architecture . . . 8

A.6 U-Net Outputs . . . 8

A.7 RGB Ratio Transformation . . . 8

1 Introduction

The used U.S. car market generates over 100 billion annually and is quickly growing. Current methods for isolating car images require time and money in order to have it done by a professional in photoshop. We offer a method to expedite and automate the process of isolating the car image making it cheaper, easier, and less time intensive. We then take the isolate car image and cluster them by related features such as color in order to make finding specific types of cars easier for buyers. Our pipeline as a whole allows individuals to easily market their car to interested buyers reducing cost and effort by all parties involved. The data we used for this task can be found in Kaggle's Carvana Image Masking Challenge.

To begin understanding the task at hand, we're going to dive in and check out some images from the dataset. We'll continue by explaining exactly what needs to be done for a given image, followed by some commentary on the specific challenges of this task and how we resolved these challenges.

1.1 Dataset Examples

The dataset contains an assortment of images of sizes ranging of various colors orientations all of size (1918, 1280). To limit the need for extreme Random Access Memory (RAM) or processing power, we downsampled these images to (128, 128). We could have solved this problem instead by purchasing extra compute and memory via a service like Amazon Web Services but due to limited financial resources we decide to instead solve the problem by downsampling the images. The downsampling consisted of taking the 1918 x 1280 image and converting it to a 128 x 128 image using Pillows defualt image resizing. Downsampling the images negatively affected the visuals of our final presentation, however, we do not believe the validity of the image segmentation or clustering was influenced.

Observe a sample of our downsampled car images in Figure A.1 and our downsampled image masks in Figure A.2.

1.2 Formal Description of Task

Here is a mathematical description of our task. Given a dataset $\mathcal{D}$ composed of images, $x \in \mathbb{R}^d$, we'd like to learn a function $f : \mathbb{R}^d \mapsto \mathbb{N}$ mapping each input image to a discrete color label in the natural numbers. It is key to understand that this task uses image mask segmentation as an intermediate; more formally, given a dataset $\mathcal{D}$ composed of images, $x \in \mathbb{R}^d$, we'd like to learn a function $f : \mathbb{R}^d \mapsto \{0, 1\}^d$, mapping each pixel to a 0 (for background) or 1 (for foregroud, meaning part of the car)

1.3 Segmentation Metric

For this task, we decided to use the F1 metric. The F1 metric is defined as follows:

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{1.1}$$

where

$$precision = \frac{TP}{TP + FP} \tag{1.2}$$

and

$$recall = \frac{TP}{TP + FN} \tag{1.3}$$

2 Methods

Our pipeline consists of a few distinct steps. We begin with aggressive downsampling, intended to reuce the need for processing power. The result is a new, smaller dataset ripe for training and inference.

We then pose an intermediary binary classification problem: given a pixel, does it belong to a car? To binarize our images, we depend on a type of Deep Neural Network (DNN) called a U-Net.

With our predicted binary mask, we proceed with the clustering problem: given an image of a car, what is the color of the car? This is accomplished using DBSCAN preprocessed with RGB-ratio feature extraction.

2.1 Downsampling

After recognizing that 32GB of RAM was not enough to handle the original resolution of the images, we downsampled from (1918, 1280) to (128, 128). To do so, we used Pillow's default resize function. An example of such downsampling can be seen in figure A.3

2.2 U-Net as a Learned Binarizer

Stacking our predictions from our linear models with our 5 features, we complete our pipeline using a DNN functioning as a learned binarizer called a "U-Net" [1]. The goal of the U-Net is to segment the image. That is, the U-Net learns a function $f : \mathbb{R} \mapsto \{0, 1\}$ for each pixel such that a class label is assigned to each pixel. For the purpose of our pipeline, the classes are

1. 0 if we predict a pixel is not part of a car
2. 1 if we predict a pixel is part of a car

A plot of the training accuracy and loss per epoch can be found in figure A.4.

2.2.1 General U-Net Architecture

To segment the image, a U-Net "supplement[s] a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. In order to localize, high resolution features from the contracting path are combined with the upsampled output" [**unet**]. Stacking the high-resolution features with the unsampled output results in symmetry between the expansive path and the contracting path, resulting in a u-shape as seen in Figure A.5. For the image borders, U-Net extrapolates missing information by mirroring the original image accross the image borders.

2.2.2 Our U-Net Architecture

Our U-Net Architecture is very similar to the original U-Net Architecture. The only modifications were as follows:

[1] The Original U-Net Proposition Paper

1. We added Dropout layers at the end of each 2D convolution block with a hyper parameter of .5. We chose .5 because it is the hyperparameter recommended by Geoff Hinton. These 9 drop out layers helped reduce overfitting by reducing the number of parameters updated after each data example.

2. We added Batch Normalization at the end of each 2D convolution block. Batch normalization improves f1 score and train time by reducing covariant shift and minimizing higher order interactions between layers. Covariate shift refers to the change in distribution of inputs, specifically the change in inputs between interior layers due to changing output of the previous layer. When covariate shift is present it means that intermediate layers continually need to adapt to the shifting distribution of input which is the output of the layer before. Batch normalization reduces covariate shift by centering the mean around zero and scales the variance to one. Centering the mean around zero and scaling the variance to one also helps reduce higher order interactions between layers which when present causes learning rates at later layers to change drastically. By reducing higher order interaction between layers batch normalization allows the use of much higher initial learning rates. In effect batch normalization allows for much smoother and quicker gradient descent where the models later layers will have more stable parameters. This effect means the model as a whole will be quicker and more likely to finding local minima which is why we see a decreased train time with batch normalization and an increased f1 score.

3. We used early stopping with a patience of 3 for the training process. This was simply to reduce overfitting, but also had a positive secondary effect of reducing training time.

Other that this modification, our architecture is exactly that described in The Original U-Net Proposition Paper. The Keras code to describe the first iteration of our U-Net architecture was taken from a Kaggle Kernel.

2.2.3 Why This Architecture?

U-Net seemed to be the default segmentation network according to several kaggle users. Furthermore, after training the U-Net, we found that it sufficiently segmented the foreground to aid the color clustering, thus we decided to use it. Given more time or computing power, we would have tried to make the network more shallow or deep. Additionally, we would have tried VGG-segnet and Keras-FCN.

2.2.4 Hand Designed Features

We also attempted to map our data into a computer vision and HSV feature-space with goals of extracting as much information as possible and creating as much contrast between foreground and background as possible. Thus, we extracted as many features as we could think of and fed these features into the U-Net.

Below, we describe the information each feature encodes and the intuition behind including that information:

1. Raw RGB Values: We first decided the include the RGB values themselves. Several other existing pipelines relied solely on the RGB values and performed quite well, meaning the RGB values must encode valuable information, so we included them.

2. Bilateral Filter: A Bilateral Filter convolves the image with a weighted Gaussian kernal. This denoises the image, while still preserving the edges. We included it to get rid of background salt and pepper.

3. 50/99 Image Rescaling: Image Rescaling widens the data distribution and increases contrast. We grabbed the 50th percentile and rescaled everything below and including that value to be 0, and we grabbed the 99th percentile and rescaled everything above that value to be 1. We included this to increase contrast between foreground and background, and to filter out salt and pepper.

4. Adaptive Histogram Equalization: Adaptive Histogram equalization increases contrast locally. We included this so that the regressors could recognize smaller nuclei among salt and papper.
5. Dilation: A dilation performs a uniform kernel convolution accross the image, thus setting each pixel to be the average of its neighbors and itself. This increases the area of each nucleus. We included this so that the regressors could pick out small nuclei.
6. HSV: This stands for hue, saturation, and value. This color space describes colors (i.e. hue) in terms of their shades (i.e. saturation) and brightness value.

Ultimately, the model using these features overfit and did not generalize very well to the test set.

2.2.5 Shallow Models

We also attempted to use the following shallow models: Logistic Regression, SVM, and Decision Trees. With each model, we used a validation set to select the best hyperparameters. However, we found that the best model was an SVM with a train F1 of 0.561 and test F1 of 0.550. Thus, we decided to use U-Net instead. Intuitively, this is because the majority of the cars in the dataset are black, white or gray, making it very difficult to distinguish from the grayscale background.

2.3 Color Clustering

To cluster the images based off of color, we first fed the image through the U-Net, finding the foreground mask. An example of such can be found in figure A.6. We then applied this mask to the original downsampled image. We pipelined this step with DBSCAN preprocessed with RGB ratio feature extraction.

2.3.1 RGB Ratio Feature Extraction

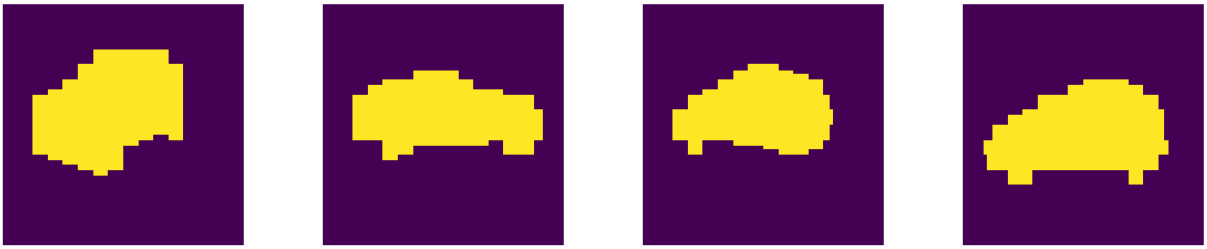
For clustering the cars we used the output of the U-Net to isolate the car itself. Thus, we could then determine the mean rgb value for each car. In figure A.7, we plot the mean RGB value in for all the cars present in our data set. We then noticed the data formed a number of linear lines radiating from the origin which would be difficult to seperates into clusters. And in fact when plotting the clusters produced from raw RGB values, it only clustered the cars in terms of various shades of gray. We then decided to try to separate the data by taking the ratios of RGBs. This, we found, created group that could then be cluster. This transformation can also be found in figure A.7.

2.4 Lessons Learned

The most important lesson that we learned as a group over the course of this project is to immediately question a perfect train and test accuracy and the important role that visualizing results has in discovering and fixing the error. We learned this lesson because of an error we made while splitting our test and train data. We split out data such that all pixels not just images were randomly assigned to the train or test set. This mistake meant that we were training in effect on random noise. We did not realize our error, however, until we decided to visualize our findings. By not continually visualizing our outputs we lost a considerable amount of time and had to redo work that we thought we were already done with. Thankfully the fix was a simple matter of changing the input for our test train split function, however, it's still unfortunate that we did not realize this error earlier but have in the end learned the importance of visualizing your output.

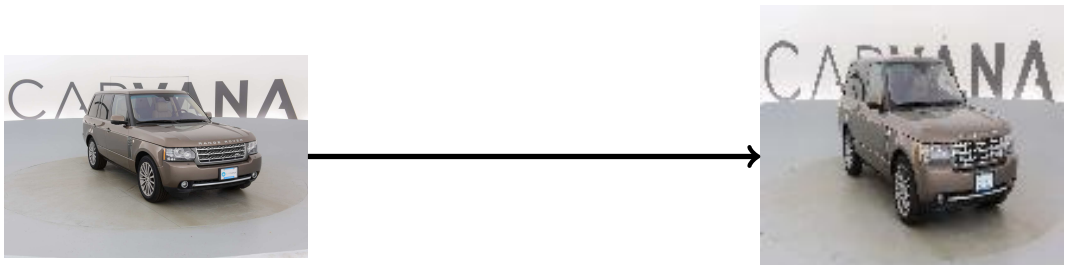
(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure A.1: Downsampled Image Examples



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

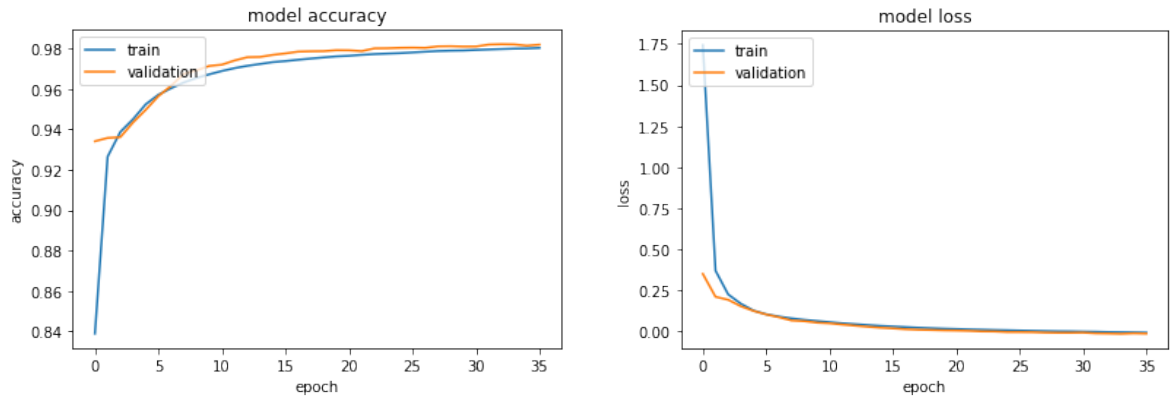
Figure A.2: Downsampled Mask Examples



Original Image

Downsampled Image

Figure A.3: An example of our image downsampling.



(a) UNet Train/Validation Accuracy History

(b) UNet Train/Validation Loss History

Figure A.4: The training loss and accuracy per epoch

A Appendix

A.1 Car Image Examples

A.2 Image Mask Examples

A.3 Downsampling Example

A.4 Training Accuracy and Loss During Training

A.5 U-Net Architecture

A.6 U-Net Outputs

A.7 RGB Ratio Transformation

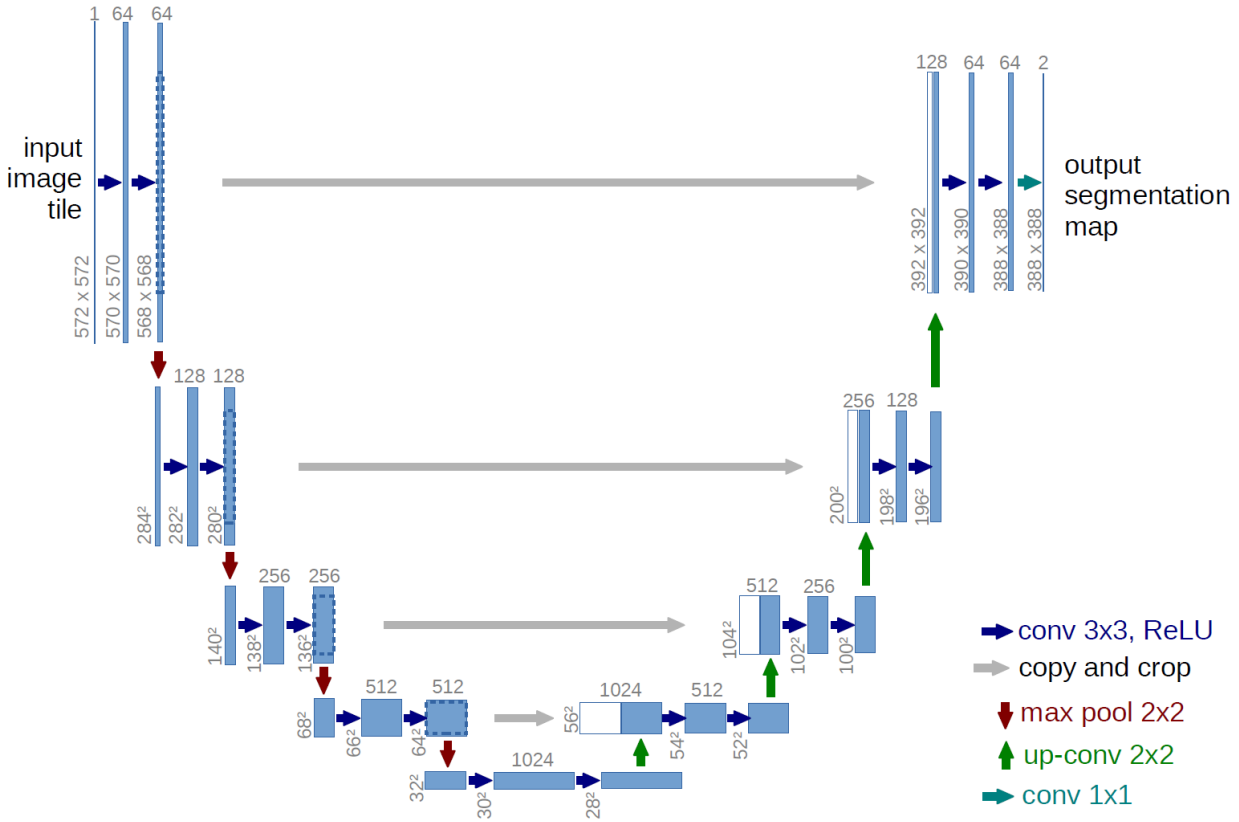
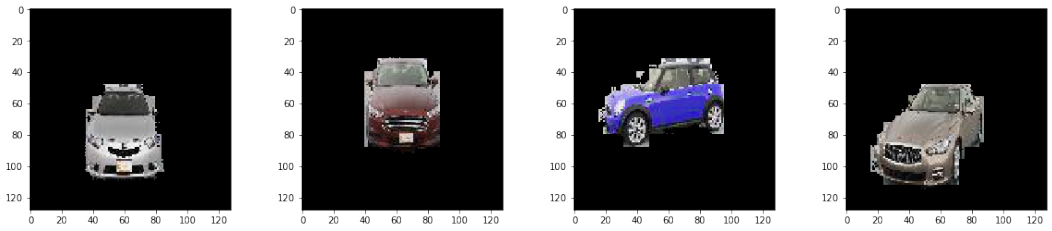
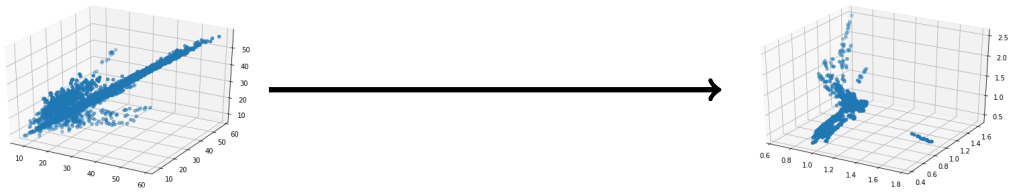


Figure A.5: The U-Net Architecture, as presented in The Original U-Net Proposition Paper



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure A.6: UNet Output Examples



RGB Plot RGB Ratio Plot

Figure A.7: A plot of the raw RGB values (left) and the RGB ratios (right)