Car image segmentation using Convolutional Neural Nets

Ashish Malhotra Follow
Sep 6, 2017 · 4 min read

This work is a collaboration between Ashish Malhotra and Jovan Sardinha.

There are four main tasks in computer vision other than just basic image classification:

- Semantic segmentation—pixel level coloring of the objects in the image
- Classification + Localization—object classification + drawing bouding box in an image with single object
- Object detection—draw bounding boxes around multiple objects of different classes in an image
- Instance segmentation—pixel level coloring of multiple objects of different classes in an image

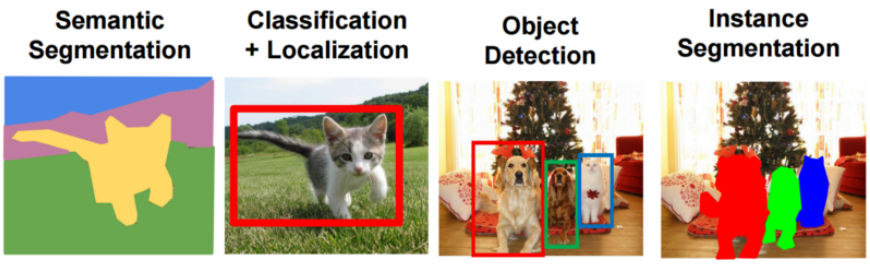

Fig 1. Various computer vision tasks (image credits: Slide 19 http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf)

For the purposes of this post we will be diving deep into semantic segmentation for cars as part of the Carvana Image Masking Challenge on Kaggle.

Objective

The objective of the carvana image masking challenge is to generate masks for cars in images with extremely high precision.

Fig 2. Input image of car

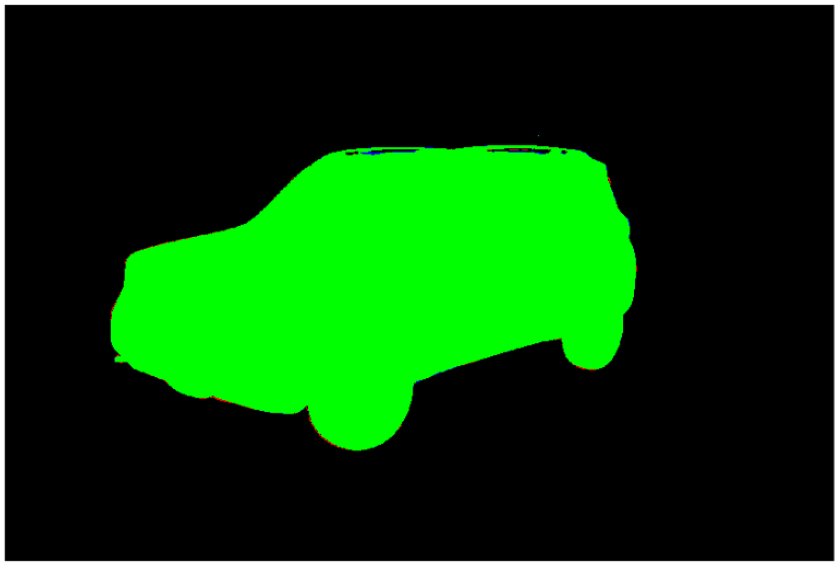

Fig 3. Output mask of car

Evaluation

The evaluation metric used for this competition was Dice coefficient. Dice coefficient is defined as follows:

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

where X is the predicted set of pixels and Y is the ground truth. A higher dice coefficient is better.

A dice coefficient of 1 can be achieved when there is perfect overlap between X and Y. Since the denominator is constant, the only way to maximize this metric is to increase overlap between X and Y.

Model

There are several popular models for semantic segmentation in recent deep learning literature like SegNet, FCN, Deconv networks etc. After some initial experimentation the SegNet architecture was used as a base for building our custom architecture.

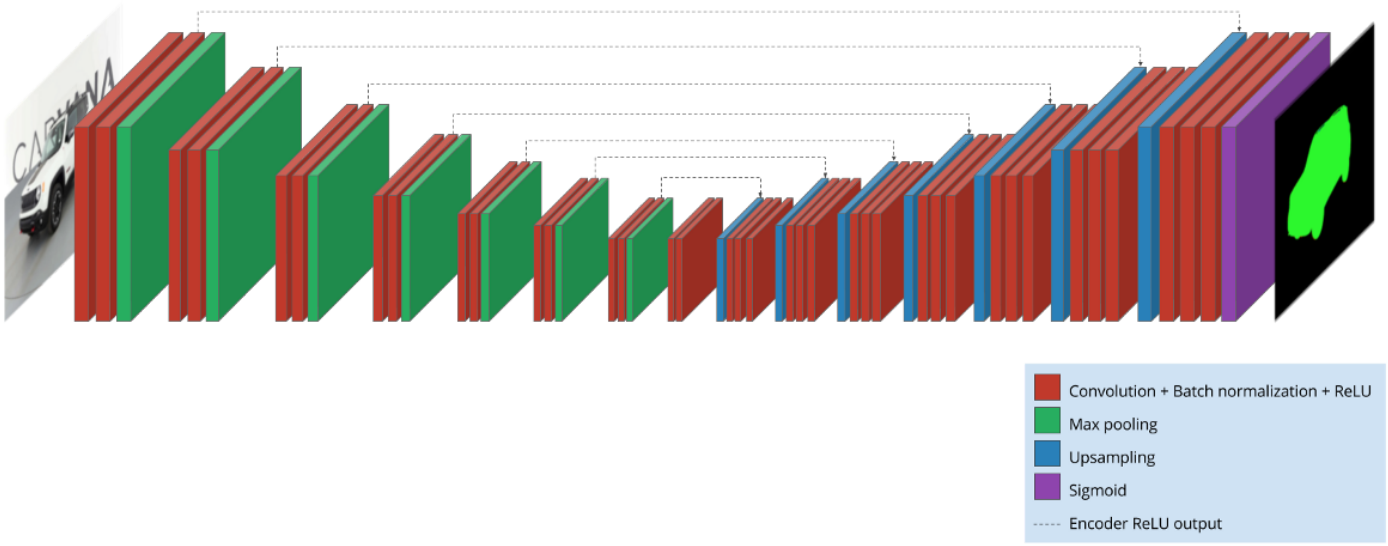

Fig 4. Model architecture

The final model consists of 7 encoder, 2 center convolutions, 7 decoder and 1 final convolutional classification layer. We settled on this architecture as it was the model with the largest number of encoder layers that would fit on our hardware (Tesla K80 GPUs 12GB memory).

Each encoder layer was made up of (conv2d, batchnorm, relu) x 2, maxpool.

```
down2 = Conv2D(128, (3, 3), padding='same')(down1_pool)
down2 = BatchNormalization()(down2)
down2 = Activation('relu')(down2)
down2 = Conv2D(128, (3, 3), padding='same')(down2)
down2 = BatchNormalization()(down2)
```

Center layers were (conv2d, batchnorm, relu) x 2.

```
center = Conv2D(1024, (3, 3), padding='same')(prev_layer_out
center = BatchNormalization()(center)
center = Activation('relu')(center)
center = Conv2D(1024, (3, 3), padding='same')(center)
center = BatchNormalization()(center)
```

Each decoder layer was made up of upsampling2d, concatenate, (conv2d, batchnorm, relu) x 3, maxpool. Important point to note is that each decoder layer used the output (before maxpool) from the corresponding encoder layer to extract learnable features. This is similar to SegNet architecture.

```
up2 = UpSampling2D((2, 2))(up3)
up2 = concatenate([down2, up2], axis=3) # <-- Use output fr
up2 = Conv2D(128, (3, 3), padding='same')(up2)
up2 = BatchNormalization()(up2)
up2 = Activation('relu')(up2)
up2 = Conv2D(128, (3, 3), padding='same')(up2)
up2 = BatchNormalization()(up2)
up2 = Activation('relu')(up2)
```

Data Augmentation

Some minor data augmentation was performed by randomly shifting, scaling and rotating the input images.

Training

Model was built using Keras with Tensorflow backend. Training was done using following hyperparameters:

- Early stopping criteria was used where `min_dela = 0.0001`
- Optimizer used: `Adam(lr=1e-4, beta=1, beta_2=0.999, epsilon=1e-08, decay=0.0)`
- Loss used: bce_dice_loss = binary_cross_entropy_loss + (1 - dice_coefficient)

Validation set dice coefficient stabilized around 0.9956 after ~13 epochs.

Post Analysis

Analysis was done on the worst performing images from the validation set. The biggest insight from this process was that the body of most cars

were getting segmented extremely well. The errors were mostly around the edges of the segmentation mask. These consisted of dark shadows near the wheels, cars painted the same as background color, really small antennas, roof racks etc. These are all things which would be difficult for a human to differentiate as well. In my opinion, the model performed equal or greater than human level for this task.

Fig 5. Input image

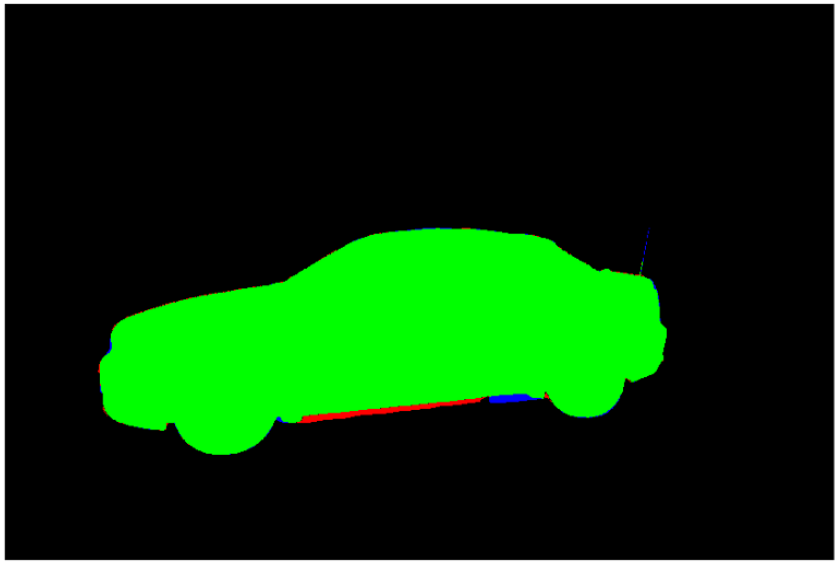

Fig 6. Predicted mask. Red: False positives, Green: true positives, Blue: false negatives

Code

Full implementation details can be found here.

References

- *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*, 2015 [pdf] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla
- *Fully Convolutional Networks for Semantic Segmentation*, 2016 [pdf] Evan Shelhamer, Jonathan Long, Trevor Darrell
- *Learning Deconvolution Network for Semantic Segmentation*, 2015 [pdf] Hyeonwoo Noh, Seunghoon Hong, Bohyung Han