Machine Learning Engineer Nanodegree

Capstone Project

Tapish Rathore

Kaggle's Carvana Image Masking Challenge

Project Overview

Pixel-level semantic segmentation is the task of assigning a class label to each pixel in an image. It involves classifying an object in an image as well as detecting its location with the highest possible resolution. Humans are exceptionally good at this task. For example, when we see a pigeon sitting on a tree, we have no difficulty in distinguishing which part comprises the tree and which the pigeon, even if we only have a single image and are not allowed to move. There are several use cases for pixel level segmentation in the field of computer vision, such as detecting road signs[1], quantification of tissue volumes[2], study of anatomical structure[3] and land cover classification[4]. It has been used extensively in medical imaging[5] and autonomous car driving[6]. With recent advancements in training CNNs to learn image features for detecting and classifying objects in images[7], efforts have been made to use CNNs for semantic segmentation as well[8]. This has resulted in several state-of-the-art CNN architectures such as Mask R-CNN[9] and SegNet[10].
The author is also personally motivated to work in this domain as he has been involved in designing a computer vision pipeline for classification of food grains[11] in the past and had implemented a non-learning approach to segmentation. Now he would like to explore it from a deep learning perspective.

Problem Statement

The problem statement is specified on Kaggle's webpage for the Carvana Image Masking Challenge[12]. The challenge is to develop an algorithm which distinguishes each pixel in a given image as either belonging to a car or the background. The effectiveness of the algorithm will be calculated using the mean of Dice coefficient[13] over all images. The Dice coefficient is a metric to compare the pixel-wise agreement between the ground truth and the mask predicted by the algorithm. A potential solution for the given task is to train a Convolutional Neural Network on the given training data to generate an image mask classifying each pixel. The problem statement can also be expanded to analyze if the algorithm is robust enough to segment cars in images of other datasets (explained below).

Metrics

The Dice coefficient is the proposed evaluation metric for this task. It can be calculated using the formula –

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

where X is the set of pixels predicted as belonging to the car by the proposed solution and Y is the ground truth. When both X and Y are empty, the Dice coefficient is assumed to be 1.
The formula can be rewritten in the following manner –

$$\frac{2}{\frac{1}{|X \cap Y|/|X|} + \frac{1}{|X \cap Y|/|Y|}}$$

which can be interpreted as the harmonic mean of 2 numbers – the ratio of true positives to predicted positives (precision) and the ratio of true positives to ground truth positives(recall). Here, true positive means the pixels labeled as belonging to the car by both the segmentation algorithm and given ground truth labels, predicted positives refers to pixels labeled as belonging to the car by the algorithm, and ground truth refers to the pixels labeled as belonging to the car by the given data.
The Dice coefficient is analogous to the F-score metric[19]. It punishes both false negatives(pixels that actually belong to a car but are left out) and false positives(pixels that do not belong to the car but are treated as if they do) equally badly. The number of mistakes is measured relative to the number of true positives. And true negatives (pixels that do not belong to the car and are classified correctly) are not taken into account. These properties make the Dice coefficient an acceptable metric for evaluating segmentation masks generated by different models for our task.

Analysis

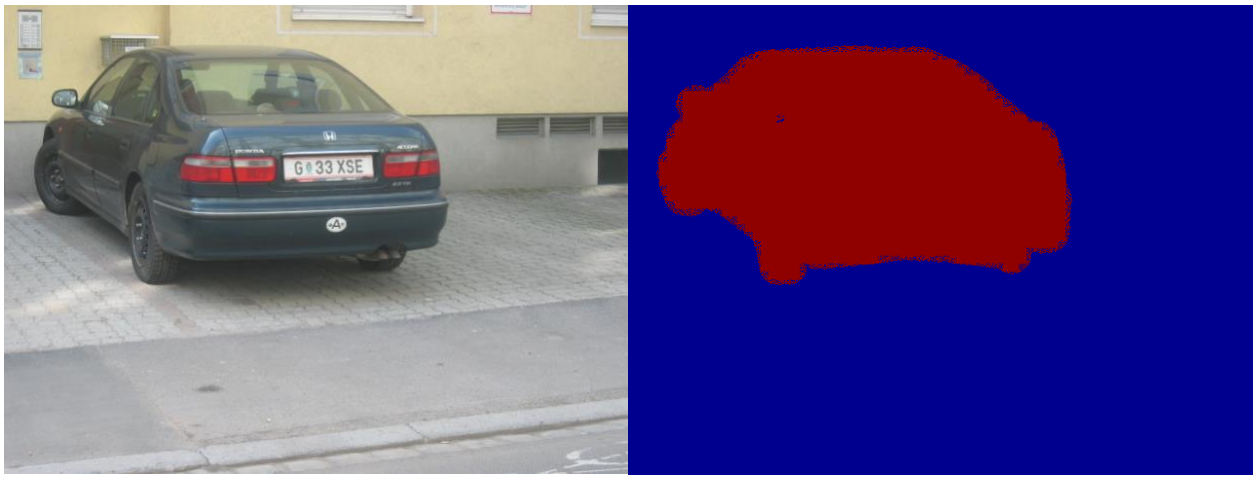
Data Exploration

The dataset used in the competition is provided on the competition website[14]. Carvana has provided a total of 5088 images for training along with the corresponding segmentation mask for each image. The images are colored and their masks are black and white, white pixels representing the car and black pixels representing everything else. Each image (and its mask) has dimensions 1918 X 1280. There is a total of 6573 different cars used during this whole task, which includes both training and testing. The training images are made up of 318 different cars, each shot from 16 different angles, in front of the same background.
Since the number of images is limited, especially for training larger CNNs, the images will be augmented during preprocessing.

(a) *(b)*

Fig. 1: Sample image (a) and its segmentation mask (b) from Carvana Image Dataset

To test the CNN model in a general setting, images were taken from the Pascal VOC Database[15], specifically from the TU Graz-02 Database[16]. These images have cars in natural settings with varied backgrounds. There is a total of 223 images along with their corresponding masks. All images and masks are of dimension 640 X 480. Segmentation masks for these images are present in multiple colors. They are thresholded to black and white during preprocessing.



(a) *(b)*

Fig. 2: Sample image (a) and its segmentation mask (b) from Pascal Object Recognition Database

Exploratory Visualizations

One of the major characteristics of the Caravana dataset is that the size of cars is considerably uniform in all images. To visualize this, the area of the smallest enclosing box around the car was calculated for all images. This area was divided by the area of the image.

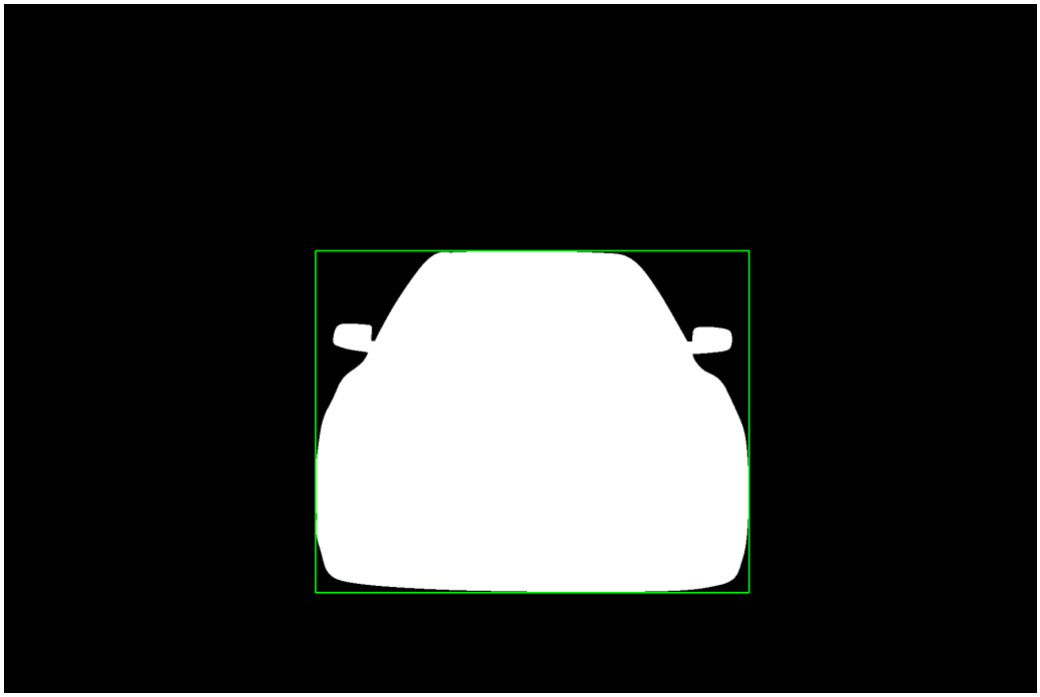
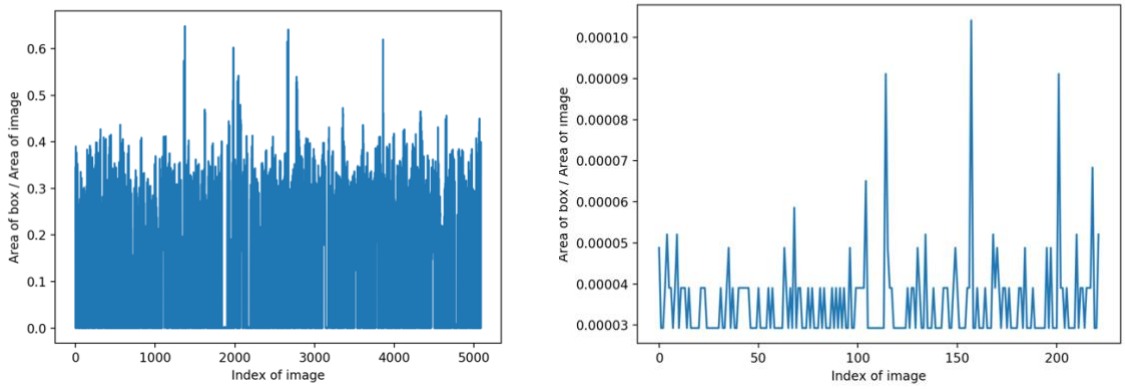


Fig 3: Example of an enclosing box around segmentation mask of car

The graph plotted from calculating these ratios for each image is shown in figure 4(a). Its statistics are summarized in Table 1. A similar process was performed for the TU Graz-2 database images. Its graph and statistics are provided in figure 4(b) and Table 1.



(a) *(b)*

Fig. 2: Ratio of car area for (a) Carvana images and (b) TU Graz-2 images

	Carvana Dataset	TU Graz Dataset
Mean	0.162839414326	0.000035411387
Standard Deviation	0.147727543204	0.000010134324
Median	0.196566247393	0.000029296875

Table 1: Summarizing ratio of car area to image area for both datasets

As can be clearly observed, the scale of car images in Caravana dataset is larger than that compared to the TU Graz dataset. The TU Graz dataset can thus be used to evaluate if the CNN model trained on Caravana Dataset is able to learn scale invariable features.

Algorithms and Techniques

Convolutional Neural Networks have been consistently producing state of the art results for many types of computer vision tasks like object recognition, detection and scene segmentation. CNNs gained popularity with the advent of GPUs and the success of AlexNet[20] in ILSVRC 2012. Most CNN models developed initially were designed for the task of classifying an image. CNNs for classification tasks are usually made up of a number of convolution and pooling layers which are responsible for extracting local and global features, and fully connected layers at the end, responsible for combining features. The final output is a 1-dimensional vector containing the probability of the image belonging to a particular class.

The "fully convolutional network" [21] changes this conventional design to produce a coarse heatmap(for each class) as the final output. This heatmap contains spatial information of the object in the image. This is achieved by treating the fully connected layers at the end as convolutional layers. They are viewed as convolutions with kernels that cover the entire input region. This makes them robust to changes in input dimensions. For instance, in the classical AlexNet CNN model, which takes an image of dimensions 227 X 227 X 3 as the input, the last convolutional layer outputs an array of dimensions 13 X 13 X 256. This is the input for the first fully connected layer. In the classical case, if the size of the input for fully connected layer is increased (by increasing the input image's dimensions), there will be an error, as the fully connected layer will not be able to process the increased dimensions. However, if the layer is treated like a convolutional layer with a kernel size of 13 X 13 X 256, it can process inputs of higher dimensions by sliding the kernel and producing a 2-dimensional input. An example is shown in Figure 3.

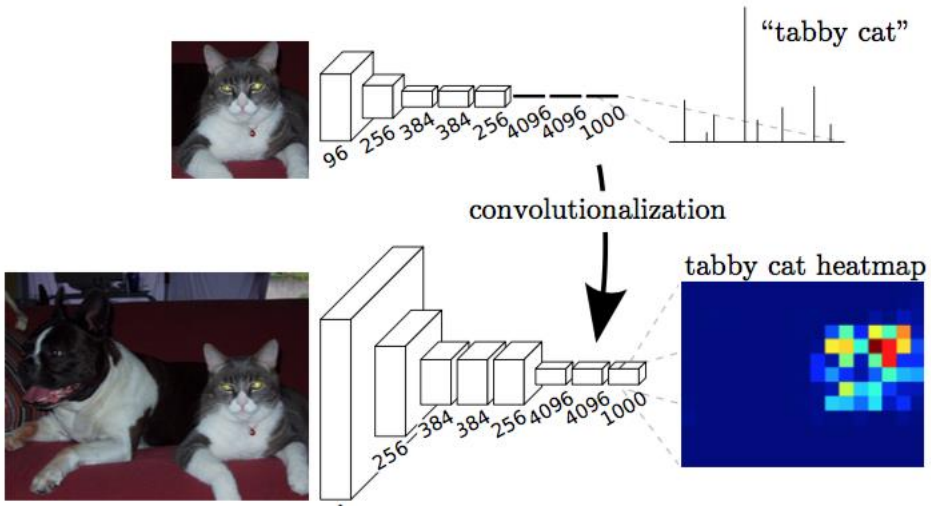


Fig 3: Converting fully connected layers to convolutional layers. The first image has dimensions 227 X 227 whereas the second image has dimensions 500 X 500 [20]

Such models are called fully convolutional models as they are made up only of convolutional layers.
The models used in this project for solving the segmentation task are called U-nets [22]. The U-net builds upon the fully convolutional network and modifies its architecture in such a way that very few training images are required for precise segmentation results. In fully convolutional networks, upsampling operators are used to improve the resolution of the coarse result. Upsampling layers are just reverse pooling layers which increase the dimensions of the output. The U-net architecture can be divided into 2 paths – the contracting path and the expanding path.

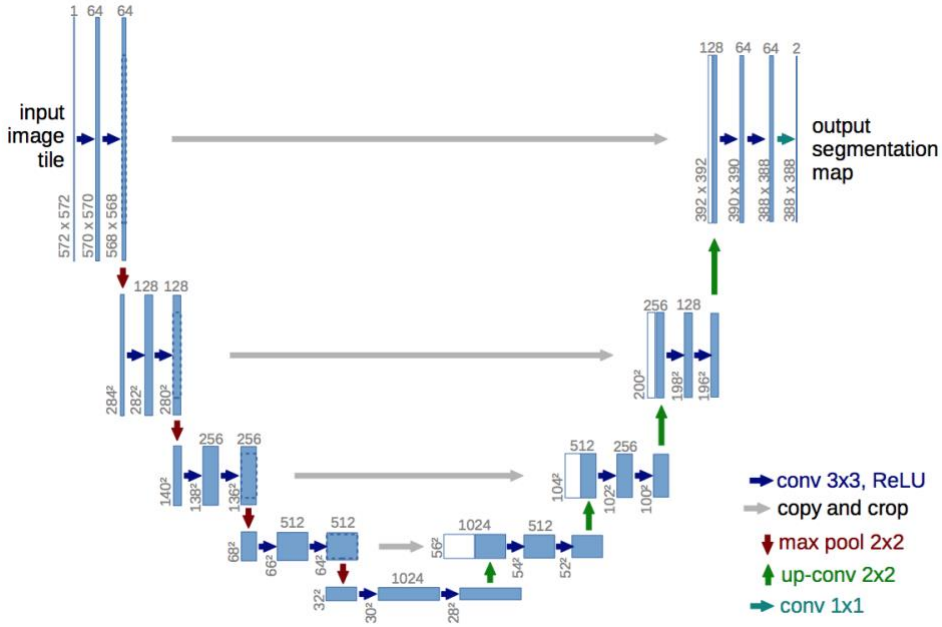


Fig 4: U-net architecture [22]

The contracting path is similar to classical CNN architectures, where each layer attempts to learn high level features from the previous layer and pooling operations decrease the dimensions of the output. The expanding path uses upsampling operations followed by convolutional layers. This path contains a large number of features, similar to the contracting path to propagate context information to higher resolution layers. High resolution features from the contracting path are also combined with the upsampled layers for the purpose of localization.
The final layer is a 1 X 1 convolutional layer which produces the segmentation mask for different classes.

Benchmark Model

The benchmark model used in this project is a fully convolutional VGG-16 CNN[18]. The fully connected layers at the top are simply converted to convolutional layers. The performance of U-nets is tested against this classical fully convolutional network trained from scratch on the Carvana images dataset. This will help in comparing the efficiency of training U-nets on smaller datasets, as compared to bigger fully convolutional networks.

Methodology

Data Preprocessing

For this project, 3 preprocessing techniques are used to augment training data and prevent the model from overfitting.
The simplest of these is Random Horizontal Flipping, where the training image is flipped through the horizontal axis with a pre-decided probability (0.5 for this project).

Fig 5: Horizontally Flipped Images

The next augmentation technique applied to the training images was a group of perspective transforms including shifting the image, rotating it, and scaling it up or down with a pre-decided probability.

Fig 6: Perspective transforms applied to image

The last augmentation technique converts the RGB image to an alternative color space – HSV or Hue, Saturation and Value, and shifts the HSV values randomly.

Fig 7: HSV shift

The horizontal flipping technique simply helps in increasing the number of training images. The Perspective transforms help the model achieve invariance with respect to shift, scaling and rotation operations. The HSV value shift helps in emulating different lighting conditions and helps achieve color and light invariance. Applying all these techniques resulted in increasing the accuracy of the best U-net model from 97.7% to 98.5%. Some additional augmentation techniques that were applied resulted in a decrease in accuracy and were removed.

Implementation

This project was implemented using Python 2.7, since many popular third-party machine learning libraries have python front-ends, like TensorFlow and Caffe.

Initially, it was planned to train the models on a Jetson TX-1 device and use Caffe to build the models. However, after repeated experiments and despite optimizations, the 2 GB RAM on the TX-1 proved to be insufficient for finishing all experiments. TX-1 comes with a pre-installed Ubuntu operating system. This system does not allow the creation of swap space, which leaves the

user stuck with the limited 2 GB onboard memory. The only option is to re-compile the entire linux kernel (after making some minor changes) to switch on swap space creation and then install the kernel on TX-1. However, this process does not guarantee a good performance, unless a solid state drive is used for swap space. Even then, the drive may wear down early due to high number of reads and writes.

Finally, the model was trained on Google Cloud Platform (GCP). GCP has a number of cloud resources related to Machine Learning. For this project, 3 services provided by GCP were used predominantly – Google Cloud Machine Learning Engine (ML Engine), Google Compute Engine and Google Cloud Storage (GCS).
The Google Cloud Storage is an online storage bucket. It was used to store the training, validation and test images. It was also used to store the generated model files containing weights for CNN models. Lastly, it was used to store logs generated during training.
Google Compute Engine was used to initialize jobs and run code. It had python and many other third party libraries like PIL and OpenCV pre-installed.
The Google Cloud Machine Learning Engine provides many APIs related to training, testing and using machine learning models for various tasks such as inference and object detection. It also provides GPU support for training models.

All 3 of these services use TensorFlow for various operations (even mundane ones, like reading a file from GCS). Hence, this project uses Keras (a TensorFlow wrapper library) to build and train CNN models. Other major libraries used include OpenCV, numpy for image preprocessing and scikit-learn, pandas for managing training data.

There are many implementations of U-nets available online from different libraries. The boilerplate code for the project for implementing U-nets was taken from Peter Giannakopoulos's github [23]. The project code was built on top of this code.

The dice coefficient is implemented as the metric against which the validation data is tested during training. Initially binary crossentropy was used as the loss function (for backpropogation). However, after running a few tests, it was determined that using a combination of binary crossentropy and dice loss produced better results. A few details about the training parameters –

- The training used RMSProp to optimize gradient descent with a learning rate of 0.0005.
- The learning rate was reduced by a factor of 0.1 whenever validation loss stopped improving by 0.0001.
- Early stopping was used to stop the training when a monitored quantity (validation loss) stopped improving by a threshold amount (0.0001) since a pre-determined number of epochs (3).
- A batch size of 8 was used to train for about 822 epochs.

During testing, a multithreaded approach was used where threads were dispatched with various test images and a thread safe queue was used to keep track. The resulting masks were compressed using run length encoding as the Kaggle competition accepts rle strings only.
To run python code on the Google Compute Engine, it is required to package the code in a specific manner and write an installation script (setup.py). Each package is run as part of a "job", where each job is assigned some specific resources, e.g. 10 GB RAM, 1 GPU, 20 GB storage space on GCS etc. These resources can be dynamically increased during job execution if the need arises.
To assign GPUs to a task, some parameters need to be specified (cloudml-gpu.yaml). A shell script is written to combine all these command line arguments and execute the job on GCP(start.sh).

Refinement

Three CNN models were trained on the training data – a U-net model which takes as input images of dimension 128 X 128, another U-net model with input dimensions 1024 X 1024 and a fully convolutional VGG-16 model (FCN-VGG16).
The U-net models are based on the design provided by Ronneberger et al [22]. The first net has input dimensions of 128 X 128. It is downsized 4 times (after passing through convolutional and pooling layers), to 8 X 8 and then upsized to 128 X 128. For input, the training image is downsized from 1918 X 1280 to 128 X 128. The output generated by the network is scaled up using bilinear interpolation.
For the second U-net, the input is again downsized to 8 X 8 after 7 sets of convolutional and pooling layers. It is symmetrically upsized to 1024 X 1024 and the output mask is interpolated.
Initially the batch size was low (in a range of 2 - 4) which resulted in an increase in the validation loss. Increasing the batch size resulted in smoothly decreasing validation loss.
Tuning other parameters such as learning rate and gradient descent optimizer also improved the loss and decreased training time.
As noted above, it was observed that a combination of binary cossentropy and dice loss presented a better result than simply using binary crossentropy. The models using the combination was able to converge faster to a better result.
Training the 1024 U-net takes ~20 hours and 128 U-net requires ~12 hours. FCN-VGG16 required ~30 hours for training. Testing and RLE conversion took ~1.5-3 hours in each case.

Results

Model Evaluation and Validation

The following table summarizes the performance of the 3 CNN models on the Carvana Image Dataset –

	Training	Validation	Test
FCN-VGG16	0.988	0.421	0.362

	Training	Validation	Test
128 U-net	0.991	0.954	0.957
1024 U-net	0.995	0.987	0.984

Table 2: Performance of CNN models on Carvana Image Dataset

It can be observed that the 1024 U-net delivers the best performance on the Carvana Image Dataset. The FCN-VGG16 model seems to overfit on the training data, as its training accuracy is high but validation and test accuracy are very low. The 128 U-net performs slightly worse than the 1024 U-net, probably because of the difference in input and output resolutions, as discussed later.

The following table summarizes the performance of these models on the TU-Graz dataset –

	TU-Graz
FCN-VGG16	0.144
128 U-net	0.452
1024 U-net	0.445

Table 3: Performance of CNN models on TU-Graz Image Dataset

The 128 U-net model performs best among the three models, although it can be said that all models perform rather poorly. This result proves that the U-net models, which were able to segment images in the Carvana image dataset are not robust enough to classify images in the TU-Graz dataset. There might be 2 possible reasons for this –

- The background in Carvana dataset images is uniform and the U-net models may end up learning the features of the background and identifying it as another class. Their learned definition of this "background" class is not robust enough to recognize the background pixels in TU-Graz dataset.
- As shown in Figure 2, the size of car with respect to the whole image in the TU-Graz image dataset is different from Carvana image dataset. The individual features that make up a car

and their relationship with each other (like the position of a tire with respect to the headlights) is not scale invariant. Also, each car in the Carvana dataset is captured only from 16 different viewpoints. The TU-Graz dataset cars captured from random viewpoints.

Justification

As summarized in Table 2, the CNN models based on U-net architecture perform better than the benchmark model – FCN-VGG16 on Carvana dataset. The performance of the 1024 U-net is 71% higher than the benchmark on test data. It can thus be stated that the U-net architecture is better at learning segmentation masks from small datasets as compared to a simple fully convolutional architecture.
The performance of 1024 U-net is better than that of the 128 U-net because of the difference in resolution. The output mask generated by the latter is 128 X 128 and has to be interpolated to 1918 X 1280 which means it loses a lot of information. The 1024 U-net on the other hand has a higher resolution for both input and output. It is also deeper and hence, might be able to connect local and global features in a better way.
The leaderboard [24] for the Carvana challenge displays a winning score of 0.997, which is 1.3% higher than the score achieved by the 1024 U-net. Since this is a small margin, it can be said that the U-net architecture is able to perform segmentation on the Carvana image dataset successfully.

Conclusion

(a) *(b)*

(c) *(d)*

Fig 8: Examples of incorrectly segmented images

The images in Figure 8 are examples where the model is unable to recognize and segment the car even partially. 8(a) and 8(b) have a top view of the cars which was not available to the model during training. 8(c) contains partial car image obscured by iron railings. The model is not robust against that kind of obstruction. In 8(d), the car is very far from the camera and the model is unable to recognize it due to the difference in scale.

This project dealt with generating the segmentation mask for a single class of objects, namely cars, from a specified dataset. The images were captured in a controlled environment (a studio where random effects such as lighting, background etc. were controlled). The U-net architecture was identified to be a good fit for training on the given image dataset. Google Cloud Platform was used for training the models on the given data. The code for training and testing the model was packaged to run as a job on GCP and the data was uploaded online on Google Cloud Storage. Finally, parameters such as batch size and learning rate were tuned by performing repeated experiments. The performance of models was summarized and the best model for segmenting cars was determined.
The most interesting aspect of this project was the analysis of performance of the trained models on another dataset. It made me reexamine the properties of the training dataset and how they might affect what the model learns. The assumption made by neural networks (and other machine learning algorithms) that the testing data will follow the same distribution as the training data was reinforced. It helped in gaining a deeper insight into the workings of a CNN.
A significant, and rather obvious conclusion from this project is that good performance of an algorithm over one dataset does not imply robustness. An improvement can be made by creating a robust training datasets or mixing several datasets. Augmentation techniques may also be developed to make the model invariant to random features.

References

1. S. Maldonado-Bascon, S. Lafuente-Arroyo, P. GilJimenez, H. Gomez-Moreno, and F. LopezFerreras, “Road-sign detection and recognition based on support vector machines,”

Intelligent Transportation Systems, IEEE Transactions on, vol. 8, no. 2, pp. 264–278, Jun. 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4220659

2. Larie SM, Abukmeil SS. 1998. "Brain abnormality in schizophrenia: a systematic and quantitative review of volumetric magnetic resonance imaging studies." *J. Psychol.* 172:110–20
3. Worth AJ, Makris N, Caviness VS, Kennedy DN. 1997. Neuroanatomical segmentation in MRI: technological objectives. *Int. J. Pattern Recognit. Artif. Intell.* 11:1161–87
4. C. Huang, L. Davis, and J. Townshend, "An assessment of support vector machines for land cover classification," International Journal of remote sensing, vol. 23, no. 4, pp. 725–749, 2002
5. D. L. Pham, C. Xu, and J. L. Prince, "A survey of current methods in medical image segmentation," *Annual Review of Biomedical Engineering, vol. 2, no. 1, pp. 315–337, 2000, pMID: 11701515*. [Online]. Available: http:// dx.doi.org/10.1146/annurev.bioeng.2.1.315
6. Treml, Michael, et al. "Speeding up semantic segmentation for autonomous driving." *NIPSW* 1.7 (2016): 8.
7. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
8. Zhao, Feng et al. "A survey on deep learning-based fine-grained object classification and semantic segmentation." *International Journal of Automation and Computing 2017.*
9. Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross B. Girshick "Mask R-CNN." *Clinical Orthopaedics and Related Research, March 2017.*
10. V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv:1511.00561, 2015.*
11. Nebulaa Innovations, [Online] *http://www.nebulaa.in/*
12. Kaggle's Carvana Image Masking Challenge, [Online] *https://www.kaggle.com/c/carvana-image-masking-challenge*
13. Dice coefficient, [Online] https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
14. Kaggle's Carvana Image Masking Challenge Dataset, [Online] *https://www.kaggle.com/c/carvana-image-masking-challenge/data*
15. Pascal Object Recognition Database Collection, [Online] *http://host.robots.ox.ac.uk/pascal/VOC/databases.html*
16. TU Graz-02 Database, [Online] *http://www.emt.tugraz.at/~pinz/data/GRAZ_02/*
17. Compiled Car Images and Ground truths [Online] *https://drive.google.com/drive/folders/0B0Te0p-dLjuqc1VRZEp0d3k1Q3c?usp=sharing*

18. Long, J., Shelhamer, E., and Darrell, T. "Fully convolutional networks for semantic segmentation." *CoRR, abs/1411.4038, 2014*
19. F-1 score, *https://en.wikipedia.org/wiki/F1_score*
20. ImageNet classification with deep convolutional neural network. A Krizhevsky, I Sutskever, GE Hinton (NIPS 2012)
21. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation (2014), arXiv:1411.4038 [cs.CV]
22. U-Net: Convolutional Networks for Biomedical Image Segmentation. Olaf Ronneberger, Philipp Fischer, and Thomas Brox (ICMAI 2015)
23. Boilerplate code from Peter Giannakopoulos, https://github.com/petrosgk/Kaggle-Carvana-Image-Masking-Challenge
24. Kaggle Carvana Challenge leaderboard, https://www.kaggle.com/c/carvana-image-masking-challenge/leaderboard