Image Segmentation for Autonomous Driving Applications

Aristos Athens
Stanford University
Stanford, CA
aristos@stanford.edu

Amelia (Qingyun) Bian
Stanford University
Stanford, CA
qb1@stanford.edu

Alice Li
Stanford University
Stanford, CA
ali2@stanford.edu

Abstract

Image segmentation is an exciting field that has wide applications in the field of computer vision. In autonomous driving, specifically, instance segmentation is especially important in identifying obstacles and moving objects in order to ensure vehicular safety.

In this project, we performed image segmentation on movable objects using a dataset provided by Kaggle. We developed three UNet based architectures and a novel way to weight important pixels to counter class imbalance. Our implementation of bounding box loss weighting and SPP-Unet has resulted in an improvement in our model evaluated by the pixel classification accuracy. Our best performance model ResNet50-UNet achieved a pixel-wise accuracy of 98.0% on the test set.

1. Introduction

One of the biggest challenges in autonomous driving is recognizing important objects in the vicinity of the car. Video cameras can easily capture visual data in real time, but any autonomous vehicle needs to process this information quickly and efficiently to avoid getting into crashes or performing illegal maneuvers. Autonomous driving has many potential benefits over human drivers, including: faster reaction time, higher predictability, and constant monitoring of the road. All of these factors contribute to greater safety, for both vehicle occupants as well as others on the road.

Baidu Inc. has recently released a dataset to Kaggle containing dashcam images with labels for any moving objects, such as other vehicles and pedestrians. A challenge hosted by the 2018 CVPR workshop for autonomous driving asked participants to segment and label all movable objects within the images. This challenge focused on safety in relation to responding to uncertainty while driving rather than safety regarding ability to follow the rules of the road. While it is extremely important to recognize and follow road signs and markers, it is arguably more important to be able to recognize and react quickly to the more unpredictable elements present on the road, including humans and other vehicles.

The inputs for this project were RBG image frames captured from dashcam videos on cars. These RBG images were passed through a convolutional neural network to output an array that can predict the category of each pixel. We looked specifically for pixels that make up vehicles, including cars and motorcycles, and pixels that make up people.

For the convolutional neural network, we tried different approaches such as DownUp sampling, UNet, and spatial pyramid pooling. In this report, we propose new methods of weighting the different categories and pixels in loss calculation in order to mitigate the class imbalance in the dataset.

2. Related Work

Image segmentation is an active field of research. In 2014, the Fully Convolutional Networks (FCN) developed by [11] introduced upsampling, which allowed for semantic segmentation on images of any size.

Many others have since built upon this work to develop different models which have produced significant improvements in classification speed and accuracy. Segnet, proposed by [1], is similar to FCN, but does not copy the encoder features from FCN, and instead copies the indices during the maxpooling layers. This method increases the efficiency of the network.

Deeplab v3 is another segmentation network proposed by [5]. This work improved upon prior implementations of atrous spatial pyramid pooling (ASPP) and outperformed previous Deeplab versions ([6] [3]).

The Dilated Residual Networks developed by [16] addresses the issue of loss of spatial acuity by implementing dilation. Their results have outperformed non-dilated counterparts.

The recent work of He et al.[9] from Facebook Research introduced Mask RCNN, which improved upon a previous model, Faster RCNN ([13]), has received some of the most promising results in the field today.

There have also been recent papers that focus on changing the loss function. Luc et al. deployed an adversarial net-

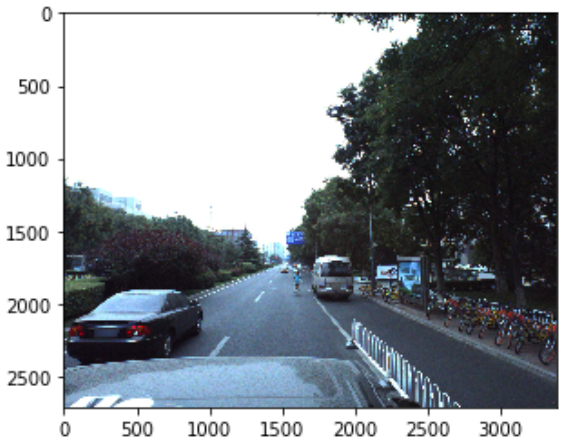

Figure 1. Example of an input image

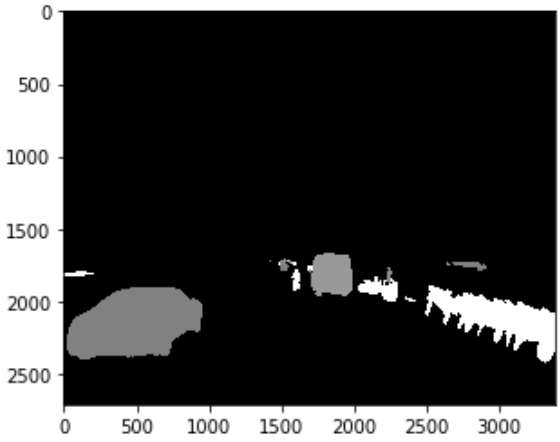

Figure 2. The label output image corresponding to Fig. 1

work and constructed a hybrid loss term that combined the adversarial loss with a multi-class cross entropy loss.[12] Bulo, Neuhold and Kontschieder employed a max-pooling concept that re-weights the contributions of pixels based on their losses to target under-represented classes.[2] We took inspiration from Bulo et al in formulating our bounding box loss weighting.

3. Dataset

For this project, we used a dataset provided by Baidu Inc., a Chinese technology company. Baidu's Apollo Project is one of world's leading autonomous driving programs. They hosted a Kaggle competition for segmenting frames from dashcam videos and provided a large dataset.[7] While we did not participate in the competition, we decided to use this dataset because we were interested in CNN applications to autonomous driving.

The dataset contains a total of 39222 RGB images in jpeg form (Figure 1) and the corresponding label in png form (Figure 2). Each input image has 2710×3384 pixels with 3 channels for RGB. Each label image is a one channel gray-scale image of the same size and each pixel is an integer number corresponding to the segmented object type and instance. There are 35 different classes, such as car, motorcycle, person, building, tunnel, traffic light, etc. In practice, however, we saw only non-background classes pertaining to vehicles and people. We split the dataset into 80% train, 10% validation and 10% test.

To change the problem from instance segmentation to regular segmentation, we performed processing on each label such that all instances of the same object were reported as the same category. Previously, Car #1 would have the label 33001 and Car #2 would have 33002. Post-processing, Car #1 and #2 would both have the label 33.

In addition, the original pictures were too big, which led to memory issues, so we cropped each picture and corresponding label once into 384×384 pixel images. We made sure to crop such that there would be a non-background object in each cropped image. During training, we normalized the input image before feeding it through the forward pass.

4. Methods

4.1. Loss

For multi-class classification, we primarily used cross entropy loss. This loss function applies a softmax layer to the end of the network and computes the negative log likelihood loss. The softmax layer outputs a per-pixel score of

$$S(f_{yi}) = \frac{e^{f_{yi}}}{\sum_j e^{f_j}}$$

where f_{yi} is the score of the correct class, normalized by the sum of all the class scores, correct or incorrect.

The negative log likelihood (NLL) takes the negative logarithm of the softmax score such that correctly predicted pixels have a high softmax score and a low NLL loss term. The final loss was the average loss of all the pixels in a batch. We used the Adam optimizer to minimize the loss function.

4.2. Class Imbalance

The dataset had labels such that the majority of pixels were background pixels, and only $\sim$10% of pixels on average were of cars, people, bicycles, etc. This meant that we had a class imbalance of around a factor of 10 between background and everything else.

Due to this fact, our models had a tendency to predict that all pixels were background pixels, resulting in all-black prediction images. To counter this, we devised two ways of weighting down the background pixels such that the model would be able to learn patterns for non-background pixels.

4.2.1 Class Weight Scheduling

The first approach was to calculate weights for each individual class. The default weight was 1, and we set the weight of each class to be inversely proportional to the prevalence of the class within each batch. This method helped produce correct predictions, but was quite slow, generally taking about 40 epochs before predictions changed from all background. Thus we set a hyperparameter to further down-weight background pixels. The weight of background pixels was given as

$$W_{background} = \alpha_{bg} * \frac{N_{bg\ pixels}}{N_{total\ pixels}}$$

where α_{bg} was the hyperparameter that we tuned between 0.01 and 0.5. We updated W_{bg} for every batch. This set of weights was then fed into our loss function.

With this approach our model was able to accurately predict object class and location, but the predicted shapes continued to be poor. We believe that by so heavily down-weighting the background we caused the loss function to not penalize false-positives sufficiently, so the model was able to predict segments with a larger profile and smoother edges than the ground truth label. We addressed this issue by using a bounding box to weight the loss.

4.2.2 Bounding Box Loss Weighting

The second approach, which we termed bounding box loss weighting, required processing after the negative log likelihood step and before averaging the per-pixel loss. First, as stated in the Dataset section, we cropped each image to a uniform size of 384 $\times$ 384 using the ground truth label images to find the portion of each image that had the largest number of non-background pixels.

Then, in addition to normal class weighting, we weighted the loss per-pixel by finding a rectangular bounding box that encapsulated all of the non-background pixels in that image. Different images in a batch would have different bounding boxes. We then multiplied the loss of the pixels outside of the bounding boxes by a hyperparameter β_{bg} tuned between 0.01 and 0.2. That way, the network would know to focus on the pixels in the box bounding all of the non-background pixels in each image.

This technique enabled the model to focus on correctly predicting background pixels near the objects without increasing the weight of background pixels in the rest of the image. We used this strategy instead of simply cropping to the smallest possible size in order to ensure that all input images were of uniform size. With the bounding-box method we saw substantial improvement in prediction granularity.

4.3. Transfer Learning & Model Ensembling

In order to speed up training and improve performance, we made use of publicly available object classification models for the downsampling portion of our models. After our initial, custom down-up model, we used transfer learning on all subsequent models. As discussed in later sections, we tested a variety of models. We produced three models capable of predicting labels with greater than 90% accuracy. To produce predictions with high spatial resolution, we used the Model Ensembling technique, averaging the outputs of all three models.

4.4. Models

4.4.1 DownUp Sampling

The first architecture that we employed was a custom downup sampling architecture. The downsampling half of our network was composed of

In $\rightarrow$ 2$\times$[Conv - ReLU] $\rightarrow$ 3$\times$[Pool $\rightarrow$ 3$\times$[Conv - ReLU]]

where we used PyTorch's Conv2d layers with a kernel size of 3 and padding of 1 to preserve the spatial dimensions. The pooling layers were MaxPool2d layers with stride 2. The downsampled batch dimension was N$\times$212$\times$48$\times$48.

The upsampling half of our network was an almost direct inverse of the downsampling half. We substituted ConvTranspose2d layers for the Conv2d layers (preserving kernel size and padding), and substituted Upsample layers for the MaxPool2d layers. Because we were classifying multiple classes, the final convolutional layer had an output channel size of 35 to match the number of classes. The output batch dimension was N$\times$35$\times$H$\times$W. This model was able to predict classes, albeit inconsistently, but was unable to produce correct object shape or location. In the end, we chose not to pursue this architecture further.

4.4.2 UNet (ResNet50)

Our custom models used a backbone of ResNet50, described by He, et al[8] and used an architecture based on UNet devised by Ronneberger, et al[14]. For our downsampling layers we took the first three blocks from the ResNet50 model, and froze the weights. Each of these blocks consisted of three convolutional layers with batchnorm and ReLU activation functions in between, followed by a downsampling layer. We used the fourth block with unfrozen weights. The upsampling path was the inverse of the downsampling path, but with deeper layers. Between each block in the upsampling path we added the corresponding output from the upsampling path. After the final upsampling step, we used a 1$\times$1 convolution to map the final number of channels to the number of classes. Thus the final output was a tensor of size N$\times$35$\times$384$\times$384, where N is the

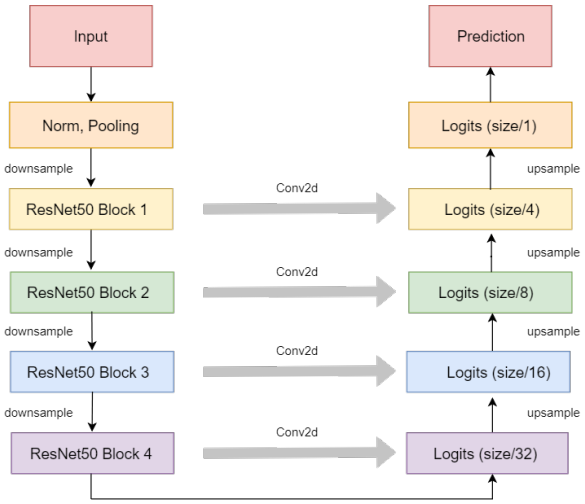

Figure 3. ResNet50-UNet Architecture

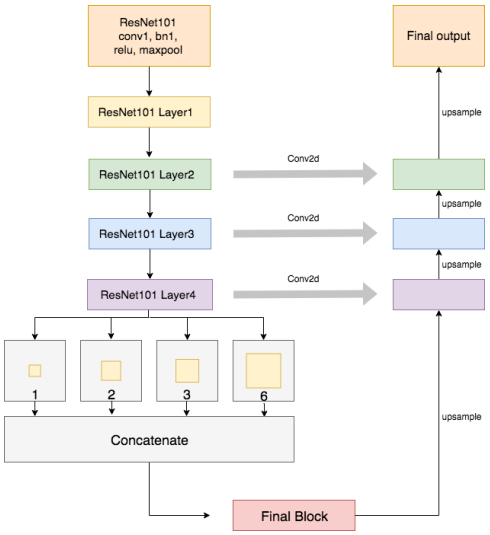

Figure 4. SPP-UNet Architecture

batch size, 35 is the number of classes, and 384×384 is the size of each prediction.

4.4.3 Spatial Pyramid Pooling

A characteristic of our dataset is that the features of objects had a large variation of sizes, with background pixels (label 0) accounting for up to anywhere between 50% to 98%. To improve our model's ability to recognize features in a variety of scales, we adopted the spatial pyramid pooling (SPP) model from [10]. The SPP model employs a variety of pooling rates to achieve a variety of effective field-of-views, and can thus learn features of the original images at different scales. According to the authors of the original work, the SPP model is able to train on images of various input sizes, which allows for cropping data augmentation to generate a more robust model. The author claims that this model improves accuracy in general and is less susceptible to deformation. Our implementation of the model used ResNet101 for downsampling, pooling at rates of 1, 2, 3, and 6, and performed a single upsampling after the final block.

We were motivated to improve upon the model by replacing the single upsampling with the upsampling technique employed in UNet structures. According to [14], the UNet allows the network to propagate context information to higher resolution layers because of the large number of channels in upsampling layers. Therefore, we modified the SPP single upsample model with a SPP and UNet upsampling architecture. Figure 4 shows the architecture of the SPPNet with UNet upsampling.

5. Results & Discussion

As described in the Models section, we experimented with various models. We produced three final models with high accuracies. Two followed the ResNet50-UNet like architecture, and one followed the SPP-UNet like architecture. We describe the final results for both types in the following sections.

For this project, we used PyTorch 0.4.0 for coding and Google Cloud for computing. We started by using 16 CPU's to train on small batch sizes and moved to 4 NVidia K-80 GPUs to train on the entire dataset.

5.1. ResNet50-UNet with Bounding Box Loss

For the ResNet50-UNet architecture with the bounding box loss, we used an Adam optimizer with learning rate of 4e-5. Our batch size was 50 because that was near the limit of memory that the GPU could handle. We chose $\beta_{bg} = 0.05$ for the bounding box loss weighting term because we found that anything greater than that would cause the training accuracy to stall at a low value for a long time before increasing.

Figure 5. ResNet50-UNet Validation Example: input image (left), predicted label (center) and ground truth label (right)

For training, we first attempted to overfit on 10 batches of size 10. We were able to achieve 99.7% accuracy on training data and 94.2% on validation. After tuning hyperparameters, we trained on the full dataset for our final model. We achieved peak values of 98.8% accuracy on training data,

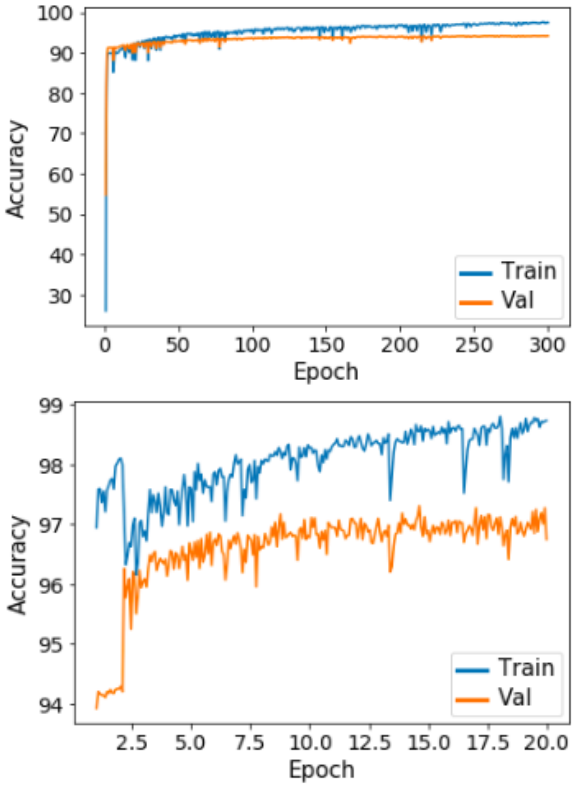

Figure 6. ResNet50-UNet Accuracy: overfitting on 2 batches (top) and switching to the entire dataset (bottom)

97.3% accuracy on validation data, and 98.0% final test accuracy.1 An example of a predicted label from the validation dataset is shown in Figure 5. As we can see, the model was able to distinguish objects in the background as well as the protruding edge of the taxi's sign on top.

	ResNet50-UNet
Overfit Training Accuracy	99.7%
Overfit Validation Accuracy	94.2%
Final Training Accuracy	98.8%
Final Validation Accuracy	97.3%
Final Test Accuracy	98.0%

Table 1. ResNet50-UNet Accuracy Results

This ended up being our best-performing architecture. In the end, both the training accuracy and the gap between training and validation accuracy increased, as seen in Fig 13. This suggests that there still existed some overfitting in the model. In the future, we would add a regularization term to our loss to bound the magnitudes of our weights.

5.2. SPP-UNet

In order to see whether using a UNet-like upsampling technique with SPP would improve the performance of the

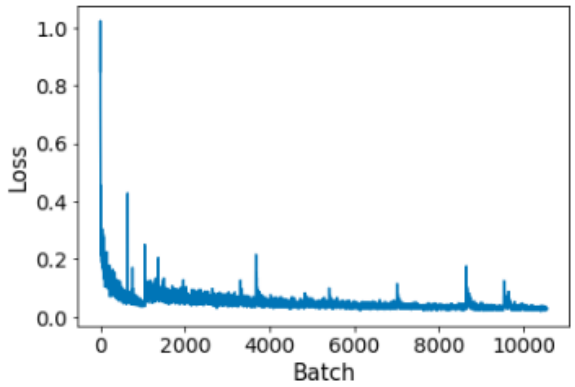

Figure 7. ResNet50-UNet Loss: around Batch #1000 we switched from overfitting on a few batches to training over the whole dataset

model, we trained both SPP with simple upsampling and SPP-UNet on the same set of parameters and data. During training on both models, we used Adam as our optimizer with a learning rate of 7e-5, and trained on 20 batches of batch size 12 over 100 epochs. The results are shown in Figure 8. As shown in the figure, both training accuracy and validation accuracies of SPP-UNet generally stayed higher than that of SPP. The highest accuracies as well as final accuracies achieved by SPP-UNet also exceeded that of SPP, as shown in Table 2

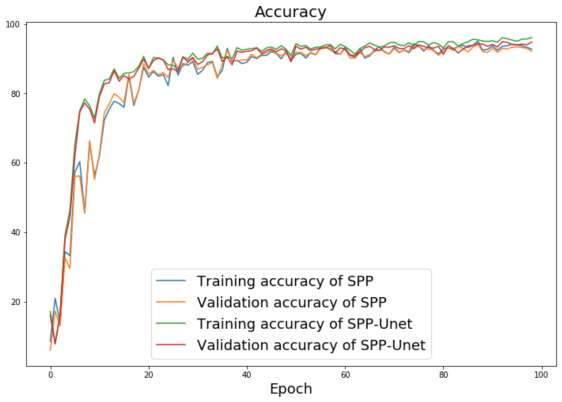

Figure 8. SPP-UNet and SPP Accuracy

	SPP	SPP-UNet
Maximum Training Accuracy	95.1%	96.1%
Maximum Validation Accuracy	94.2%	94.9%
Final Training Accuracy	92.8%	96.1%
Final Validation Accuracy	92.2%	94.8%

Table 2. SPP and SPP-UNet Accuracy Results

We then increased the training data to improve the accuracy of SPP-UNet. Training on 200 batches of size 20, we were able to see significant accuracy improvement. Al-

	SPP-UNet
Final Training Accuracy	98.1%
Final Validation Accuracy	97.0%
Final Test Accuracy	97.9%

Table 3. SPP-UNet Accuracy Results

though the entire training set has a total of 33000 images, we only trained on 4000 (200 x 2) images over 160 epochs due to time limitations. The accuracy and loss curves are shown in Figure 9

The final validation accuracy was 97.0%, and final training accuracy was 98.1% (Table 4). Both accuracies were also the highest achieved during training, which indicates that the accuracies were still increasing. The gap between validation and training accuracies indicates overfitting, which can be reduced by increasing the training dataset. Based on the results, we believe that if we train on a larger portion of the data and for more epochs, we would be able to achieve even higher accuracies.

During training, we noticed that the accuracy curves sometimes sharply decreased in Figure 9. This is due to the way Adam updates parameters internally. The Adam algorithm is as follows:

$$t \leftarrow t + 1$$

$$lr_t \leftarrow lr * \sqrt{\frac{1 - \beta_2^t}{1 - \beta_1^t}}$$

$$m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g$$

$$v_t \leftarrow \beta_2 * v_{t-1} + (1 - \beta_2) * g^2$$

$$var \leftarrow var - lr_t * \frac{m_t}{\sqrt{v_t} + \epsilon}$$

The Adam optimizer keeps track of m, an exponential moving average of the mean gradient, and v, an exponential moving average of the squares of the gradients. It then divides update parameter (var) by $v + \epsilon$, and multiplies the learning rate by the update parameter. After training for a long time and approaching optimal outputs, the value of v can become quite small. If we then reach a situation where the gradient starts increasing again, the update parameter will be divided by a very small number and explode. This will cause the learning rate to be huge, and cause us to step out of a local minimum. We used the default values of Adam, which are β = (0.9, 0.999) and $\epsilon = 10^{-8}$. The value of m changes more quickly than v, and when v is much smaller compared to m, a small ϵ value would lead to a large update. This problem can be remedied by using a larger epsilon value.

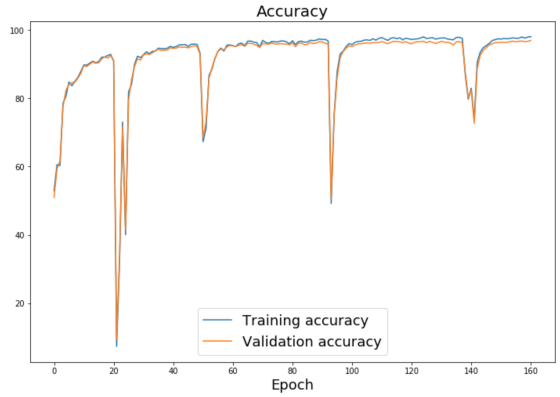

Figure 9. Accuracy Curves of SPP-UNet training on 200 batches of 20

5.3. Dilated Residual Network

The model introduced in [16] has improved the resolution of pixel prediction by implementing dilation. We built the DRN_A model based on ResNet18, but due to time limitations, we were not able to finish the training. The model is shown in Figure 10 from the original paper.

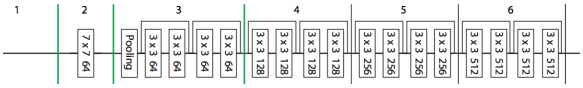

Figure 10. DRN_A-18 Architecture

5.4. Model Comparison

Between the simple upsampling and UNet upsampling SPP models, we saw that UNet upsampling improved the performance of the model. This is due to the extra channels of features during upsampling.

Both ResNet50-UNet and SPP-UNet achieved relatively high accuracies on the validation set. ResNet50-UNet required fewer parameters. We were therefore able to perform training with larger batches and using the entire training set. SPP-UNet required many more parameters than ResNet50-UNet, and achieved lower validation and test accuracies. The test accuracy of SPP-UNet is significantly lower than the validation set accuracy, which suggests that in tuning hyperparameters, we overfit to the validation set.

Although we expected SPP-UNet to perform better than ResNet50-Unet because it employs multi-rate pooling and a deeper network with ResNet101, we did not observe results that aligned with our expectations. This is potentially because we did not train for long enough on the entire dataset with the SPP model due to the significantly longer training time caused by more parameters. Overall, ResNet50-UNet was our best model.

Below are some prediction images evaluated using input images from the test set. As shown in Figure 11, we can see

	Test Accuracy
ResNet50-UNet	98.0%
ResNet50-UNet Modified	96.0%
SPP-UNet	97.9%
Ensembled Model	97.1%

Table 4. Test Accuracy Results

sharp, defined edges and shapes that greatly resemble the cars and buses in the input images. The set from ResNet50-UNet qualitatively seems slightly better than the SPP-UNet set, which matches the quantitative results from Table 4.

Figure 11. Final predictions from the test set: input image (left), predicted label (center) and ground truth label (right). The top set is from ResNet50-UNet, the bottom set is from SPP-UNet

5.5. Saliency Maps

We created saliency maps to visualize what our model was seeing, shown in Figure 12. In the examples, we see that the gradients are concentrated around the moving objects in the images, which makes sense. In addition, in both examples there are brighter clusters around the bottom sections of the vehicles. This suggests that the model is focusing on the edges of the vehicles, especially where there are sharper edges defined by wheels.

5.6. Other Attempts

Our first model, DownUp, was an entirely custom-made model with no transfer learning. While this provided great freedom in choosing which layers to put in, ultimately it could not make up for the lack of pre-trained layers from a model like ResNet50. It was able to correctly predict some categories, but the labeled shapes were too round to be able to discern the underlying object.

We also tried implementing the DRN_A dilated model introduced in [16]. Upon training on 10 batches of batch size 20, and using a learning rate 7e-5, we were able to see a gradual increase in both training accuracy and validation accuracy. However, the accuracy increase was relatively slow, and due to time limitations, we did not finish the training of this model.

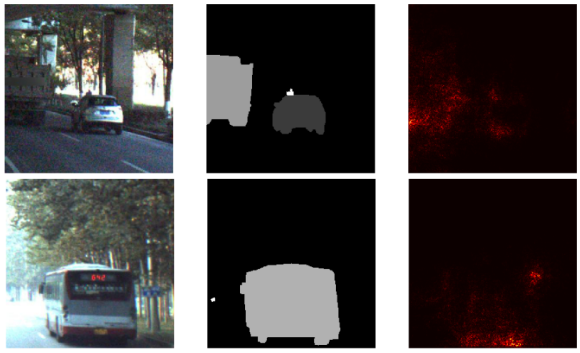

Figure 12. Saliency maps; input image (left), ground truth label (center) and saliency heat map (right)

5.7. Dataset Discussion

While the dataset we used was large, there were some limitations. To begin with, the ground-truth labels provided made no use of certain classes, namely vegetation, road, and sky. Instead the majority of each image was incorrectly labeled as ”background” or ”unlabeled”, which both map to black. This contributed to the large class imbalance we dealt with.

Additionally, there were some labels that were missing or incorrect. In 13 we can see an example of an image from the dataset that is missing a label for the car in the mid/foreground. Ignoring such prominent features can potentially harm our model during training by penalizing a false-positive (although in reality it is a correct positive).

An additional issue is image resolution. Although the original images are huge, the edges of objects are often blurry, either due to weather conditions or motion blur. On the other hand, the pictures in the dataset have a variety of lighting conditions and environments, which can allow us to train a more robust model.

Figure 13. Example of poor data: Dataset image (left) and ground-truth label (right). The label is missing the large car in the middle of the image.

6. Conclusion & Future Work

In this project, we explored several models and loss functions in performing semantic segmentation on a new Kaggle dataset, primarily created for autonomous driving.

Our highest performing architecture was the ResNet50-UNet with the bounding box loss with a test set accuracy of 98%. We believe that the bounding box loss weighting improved our performance by concentrating only on relevant pixels and classes. With more time, the SPP-UNet architecture may have exceeded the ResNet50-UNet performance, but we were not able to train on sufficient data given memory and time constraints.

In the future, we would like to explore techniques that focus more on learning the finer features in the picture. The work of [4] employed atrous layers to increase the field-of-view. We would also like to complete the training of our dilated neural network model. In addition, another possible future direction could include the implementation of an LSTM RNN model. During our project, we trained our models on random images, but because the dataset is created from videos, we expect that we can improve the general accuracy by training on a model that takes into account the continuity relationship of images in a time series.

7. Contributions & Acknowledgements

Amelia developed and trained on the SPP-UNet model. Aristos and Alice developed the bounding box loss and other weighting functions and trained on the ResNet50-UNet models.

We studied the GitHub repositories of [15] and [17] in creating our models.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: 39.12 (2017). DOI: 10.1109/tpami.2016.2644615.

[2] Samuel Rota Bulo, Gerhard Neuhold, and Peter Kontschieder. "Loss Max-Pooling for Semantic Image Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.749.

[3] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: (2016). DOI: 10.1109/tpami.2017.2699184.

[4] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: (2018). DOI: 10.1109/tpami.2017.2699184.

[5] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: (2017).

[6] Liang-Chieh Chen et al. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: (2014).

[7] *CVPR 2018 WAD Video Segmentation Challenge*. 2018. URL: https://www.kaggle.com/c/cvpr-2018-autonomous-driving.

[8] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (2016). DOI: 10.1109/cvpr.2016.90.

[9] Kaiming He et al. "Mask R-CNN". In: (2017). DOI: 10.1109/iccv.2017.322.

[10] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: (2014).

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: (2015). DOI: 10.1109/cvpr.2015.7298965.

[12] Chintala Luc Couprie and Verbeek. "Semantic Segmentation using Adversarial Networks". In: (2016).

[13] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: 6 (2017). DOI: 10.1109/tpami.2016.2577031.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). DOI: 10.1007/978-3-319-24574-4_28.

[15] Warmspringwinds. *warmspringwinds/pytorch-segmentation-detection*. URL: https://github.com/warmspringwinds/pytorch-segmentation-detection.

[16] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. "Dilated Residual Networks". In: (2017). DOI: 10.1109/cvpr.2017.75.

[17] Zijundeng. *zijundeng/pytorch-semantic-segmentation*. URL: https://github.com/zijundeng/pytorch-semantic-segmentation.