Convolutional Networks for Semantic Segmentation

Keshav Rungta
Dept. of Electrical & Computer Engineering
University of California, San Diego
San Diego, CA 92092
krungta@ucsd.edu

Geeling Chau
Dept. of Electrical & Computer Engineering
University of California, San Diego
San Diego, CA 92092
gchau@ucsd.edu

Anshuman Dewangan
Dept. of Computer Science
University of California, San Diego
San Diego, CA 92092
adewanga@ucsd.edu

Margot Wagner
Dept. of Bioengineering
University of California, San Diego
San Diego, CA 92092
mwagner@ucsd.edu

Jin-Long Huang
Dept. of Physics
University of California, San Diego
San Diego, CA 92092
jih002@ucsd.edu

Abstract

In this paper, our goal is to apply an encoder-decoder network for the problem of semantic segmentation on the India Driving Dataset. Semantic segmentation belongs to the field of persistent problems in computer vision that requires high accuracy and efficiency, with applications in autonomous driving, robotics, and e-commerce. Our baseline model achieved a pixel accuracy of **0.699** and an Intersection over Union (IoU) of **0.159**. We worked to improve the results from a fully convolutional encoder-decoder network using various techniques including augmenting the dataset and addressing class imbalance, with the combination of techniques resulting in an accuracy of **0.691** and IoU of **0.144**. We further experimented to optimize the model by utilizing different architectures including changes to the activation function and number of layers, transfer learning, and U-Net implementation. Our best model, using transfer learning, sees a test accuracy of **0.723**, and a test IoU of **0.149**.

1 Introduction

As compute power increases, we see a steady jump in performance in the traditional computer vision problems of object detection, semantic and instance segmentation, image reconstruction and clustering. This improvement in performance was even more pronounced with the advent of neural networks and especially Convolutional Neural Networks (CNNs) and Deep Learning. In this paper, we specifically look at the problem of Semantic Segmentation and analyse different architectures that were important milestones for this problem.

We start this analysis on a baseline model which is a simple encoder-decoder network. More details on this network can be found in **Section 3.3.1**. We then work to improve this architecture using data-augmentation and weighted loss functions. With a baseline in place, we then analyse custom architectures, transfer learning, and U-Net, comparing each of these different models against each

other. All of these models are trained and tested on the India Driving Dataset **Section 3.1**. Finally we evaluate our models on two common metrics in the Semantic Segmentation domain, Pixel Accuracy and Jaccard Index or IoU (Intersection over Union). We also try to negate the effects of class imbalance in the dataset using a weighted loss version of Cross Entropy Loss. Our results can be found in **Section 4**.

2 Related Works

Convolutional Neural Networks have been ever-growing with applications in many domains. As a result there are numerous different architectures that have developed in order to address differing tasks. In the domain of semantic segmentation and object detection, there are similarly a variety of architectures suited for different images classification tasks. Some commonly used architectures include U-Net, Feature Pyramid Networks and Fully Connected Networks **[11, 7]**. They all rely on the basic encoder-decoder network for their inspiration.

Similar to CNNs, the field of Autonomous Driving is currently booming and as a result there are numerous different datasets that have been curated for various purposes. Datasets like KITTI, NuScenes, Astyx, India Driving Dataset (IDD) are the current fore-runners **[3, 1, 10, 12]**. However when transfer learning is applied on a proposed solution, datasets like COCO, MS-COCO and ImageNet become integral as these are really big datasets with labeled images **[8, 2]**. For the purposes of Semantic Segmentation, however, the labels need to be very specific and as a result we are using IDD.

In the evaluation of the various architectures, there are numerous different metrics that have been used. Some of the traditional ones include Mean Squared Error and Pixel Accuracy. But with the negative aspects of the metrics affecting the quality of the segmentation, we will be using IoU as a backup metric to also compare our models.

3 Methods

3.1 IDD Dataset

For this work, we are using the India Driving Dataset (IDD) **[12]** which has 27 object categories, annotated by pixels. We are using an image subset of approximately 7,000 images. The images are taken from a front facing car camera driving around the cities of Hyderabad and Bangalore, India. The categories include things like car, person, road, sky, and tunnel. It is an imbalanced dataset with fewer than 10^5 total images of the least populated category of "tunnel" and over 10^9 images in the most popular category, "road." We split the dataset according to 4:2:1 train, validation, and test.

As we can see in the sample images and the labels in **Fig. 1**, both the image and the corresponding label is of size $1080\times1920\times3$. As a result, we had to crop them to make our model more memory efficient and increase the batch size. For this we used a crop size of $256\times256\times3$. We took this opportunity to augment our dataset and extracted 5 sub images from each image where each cropped image is of shape $256\times256\times3$. We applied the same crop to the labels as well to train the model on the proper corresponding regions. More analysis on this augmentation can be seen in **Section 4**.

3.2 Weights Initialization & Metrics

We saw in the last couple of assignments that proper initialisation of weights has a huge impact on the performance of the network. This is because poor initialization can cause the weights to either explode or the gradients to vanish both of which are detrimental. In our models, we utilized Xavier weight initialization **[4]**. This strategy initializes weights by sampling a random uniform distribution bounded by

$$\pm\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{1}$$

where n_i is the number of connections into a given layer and n_{i+1} is the number of connections outgoing from a layer. This is to avoid the problem of vanishing gradients for early layers when the

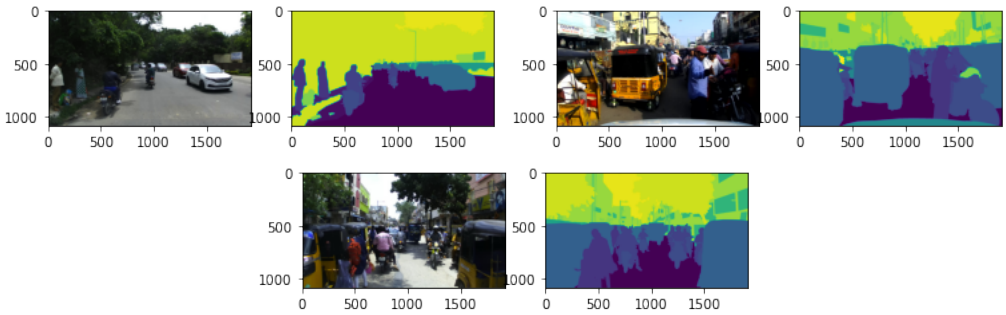


Figure 1: Some sample images in the dataset and the corresponding lables

fan-in size is large. When it was discovered, it was found to maintain gradient variances across all layers of the network leading to faster convergence and better accuracy.

In order to measure accuracy we are using two metrics: Pixel Accuracy and Intersection over Union (IoU). We trained our model with Cross Entropy Loss, which is given by:

$$E = -\sum t \log(y) \tag{2}$$

Here the target, t, is one hot encoded to only use the predictions corresponding to the specific class that is of concern to the model. This is a very flexible loss function in any classification associated task and has proven to give accurate results. The Pixel Accuracy metric is a very simple one. We compute the following for each image:

$$acc = \frac{\#correct}{\#pixels} \tag{3}$$

and average this over each image in the batch. The downside of this is that since all the classes are mixed into one calculation we have a metric that could give us good results even if the segmentation qualitatively is not good. An example of this can be seen when one class clearly outnumbers all other classes in the dataset and as a result the network could simply learn to predict that one class. In order to get around this, we also use the class-wise IoU metric, given by:

$$IoU = \frac{Intersection}{Union} = \frac{TP}{TP + FP + FN} \tag{4}$$

Here we basically compute an interesection and a union for each class in each image. This gives us a class wise IoU for each image. Then we average over the classes and the images to log the IoU.

3.3 Model

3.3.1 Baseline

Implementation

The baseline model for our semantic segmentation task was a fully convolutional network (FCN) which utilizes a convolutional neural network (CNN) to predict the category of each pixel in the image. The general structure of the network can be seen as an encoder-decoder network, where we first use a CNN to obtain features from the input image at the different resolutions in an encoding stage followed by a decoding stage, where features are mapped back to the dimensions of the input image using transposed convolution layers. Finally we use a Softmax layer in order to get the prediction for each pixel for each class.

For training, we utilized the Adam gradient descent optimizer provided in PyTorch as well as early stopping in order to use the best model to evaluate against the test set. Our batch size was 64, and used a learning rate of 5e-3, and early stopping was used to save the weights from the iteration of the model that performed best on the validation set. The Adam optimiser implements a weight decay

on the weights. However, we did not utilise this parameter in our training. If we would have used it, the weight update equation would have become **[9]**:

$$w_i \leftarrow (1 - \lambda)w_i - \alpha \nabla L \tag{5}$$

where λ is the weight decay factor and α is the learning rate. This might have helped with training but we were unable to experiment with it.

We also used batch normalization which normalizes the layer inputs in order to speed up and stabilize training by avoiding internal covariate shift **[6]**. First, this z-scores every variable at every layer of the network over the mini-batch using the equation:

$$\hat{x}_k = \frac{x_k - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} \tag{6}$$

Batch normalization then assigns adaptable parameters according to the following equation:

$$y_k = \gamma_k \hat{x}_k + \beta_k \tag{7}$$

so that the network can undo the batch normalization to get original inputs if necessary.

Each Convolution Layer in the encoder uses a kernel of size 3 and stride of 2 with a zero padding of 1. Between each convolution layer, we use a ReLU activation function followed by a batch norm. The decoder is a little different. We use a kernel of size 3 with stride 2 zero padding and zero output padding of 1. But the difference is that these convolutions are dilated convolutions of size 1. What that means is that the kernel has a padding of zeros between each column and row in the kernel. The last convolution layer is a 1×1 convolution layer which is used to map the features to each class in the dataset. The high-level architecture of this baseline model can be seen in **Fig. 2**.

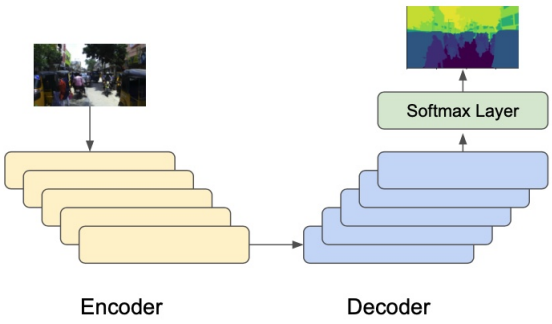


Figure 2: High-level architecture of baseline model.

Improvements: Dataset Augmentation and Weight Balancing

To improve upon the baseline, we augmented the dataset by cropping the images at the corners in addition to the center crop, since more training data can allow the model to learn better representations of the original data. This provided us with with 5x more data. We maintained each crop size to be the same size as the original baseline crop size of 256x256 to perfectly provide an augmentation of the same sized inputs.

Additionally, the dataset is highly imbalanced, as seen from a sample of pixel label counts in 200 training images below (**Table 1**). To address this, we implemented a weighted loss according to a sub-sample of cropped images. This sub-sample was dynamically generated by sampling 200 more images than when we observed all the pixel classes. In this random sub-sample, we counted the pixels in each class, then found the relative percentages by dividing by the total sum of all pixels in the sub sample. Then, we obtained an inverse of the percentage by subtracting it from 1. To exaggerate

the weight differences, we also multiplied the original percentages by 2 before subtracting it from 1. This will down weight the more frequent labels more. These weight adjustments are important for the network to learn more from the small sample of rare classes by adjusting weights relatively more aggressively when processing the error for these classes.

Class Name (label ID)	Num Pixels	Pixel Percentage	Weight
Road (0)	3285985	0.250	0.501
Sidewalk (2)	18068	0.001	0.997
Rider (5)	438121	0.033	0.933
Motorcycle (6)	255486	0.019	0.961
Car (9)	698914	0.053	0.894
Bus (11)	202977	0.015	0.969
Billboard (17)	154602	0.012	0.977
Building (22)	746465	0.057	0.887
Sky (25)	1380081	0.105	0.790
Unlabeled (26)	4804	0.0004	0.999

Table 1: Pixel count and percentages for a subset of class labels in 200 center cropped images in train dataset.

3.3.2 Experimentation

Custom Architecture

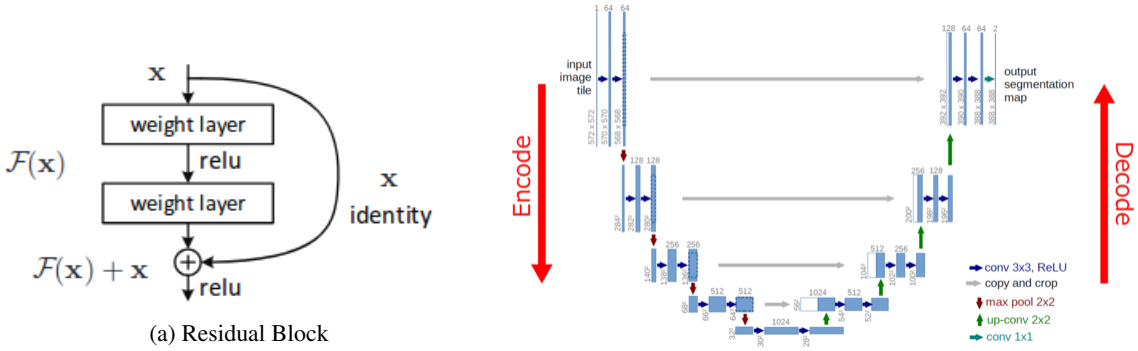
Having taken care of the defects in the dataset, we now want to evaluate the quality of the baseline. To that effect, we experimented with different architectures. In our first custom architecture, we changed the activation function in the intermediary encoding and decoding steps from ReLU to leaky ReLU. The leaky ReLU activation function modifies ReLU to allow for small negative values when the input is less than zero. This small negative value would allow the removal of vanishing gradients as the activation is not 0 in the negative half of the domain. In our second custom architecture, in addition to using leaky ReLU activation functions, we reduced the number of layers from five to two for both the encoder and decoder parts of the model. We hypothesized that a smaller architecture can speed up training times and reduce any potential overfitting.

Transfer Learning

Another experiment was to apply transfer learning to the model. This was done by initializing the model to a ResNet-18 model **[5]** pre-trained using the ImageNet dataset. This is a built-in PyTorch architecture. Because this is a semantic segmentation task, we additionally include the transformation through a 1×1 convolutional layer as well as re-scaling the predictions to the initial shape of the input images. Training was implemented using the same methods as the baseline model where, instead of a basic FCN model with Xavier weight initialisation, we use the weights learnt by ResNet-18 model. The way ResNet works is that it is made up of what is called residual blocks. A sample residual block can be seen in **Fig. 3a**. Here the identity mappings in each block are essential as they help solve the vanishing gradient problem. This allows us to use really dense networks trained on many more training examples than we had access to. As a result we are using a network with 18 layers in it.

U-Net

The last experiment was to implement U-Net architecture, which functions using a contracting encoding path and a expansive decoding path, shaped like a U. This can be seen in **Fig. 3b**. At each step, the encoding doubles the number of channels and halves the spatial dimension. The decoder does the opposite, halving the number of channels and doubling the spatial dimension at each step. Encoding function by repeated applying two 3×3 un-padded convolutions each with a ReLU activation afterwards. Lastly, it is downsampled to double the number of feature channels. This is done with a 2×2 max pooling operation with stride 2. In the decoder, the feature map is upsampled and a 2×2 convolution is applied. Next, it is concatenated with a cropped feature map from the corresponding step in the encoding function. Lastly, two 3×3 unpadded convolutions followed



(a) Residual Block

(b) U-Net Architecture

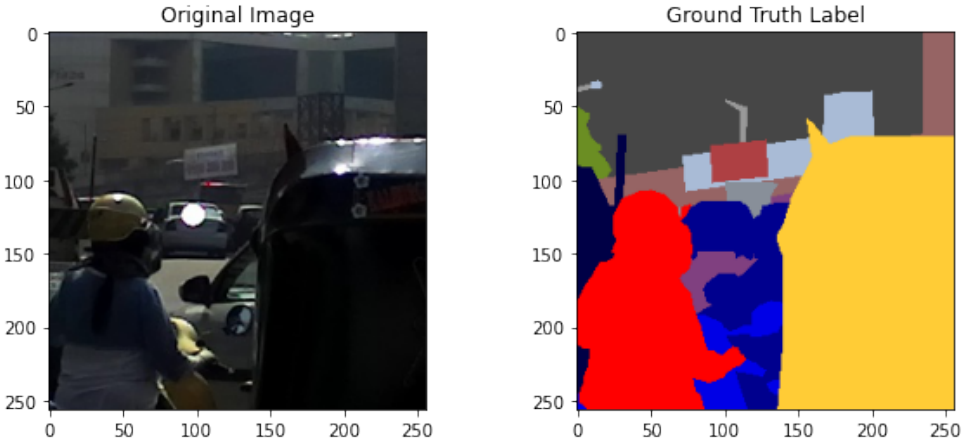
Figure 3: Other architectures we used to improve the baseline

by a ReLU are applied. And finally we use a 1×1 convolution to get the network to output 27 channels corresponding to the number of classes. U-Net is supposed to be optimized for semantic segmentation tasks and should theoretically improve our accuracy drastically.

A breakdown of each of the architectures we used can be seen in **Appendix Tables 4, 6, 7, 8**.

4 Results & Discussion

Below we present the results of all our models that we ran on the test set and some sample outputs we get from them after training. **Fig. 4** shows us what the output should look like and what the original image we passed into the network was.



(a) Ground truth labels

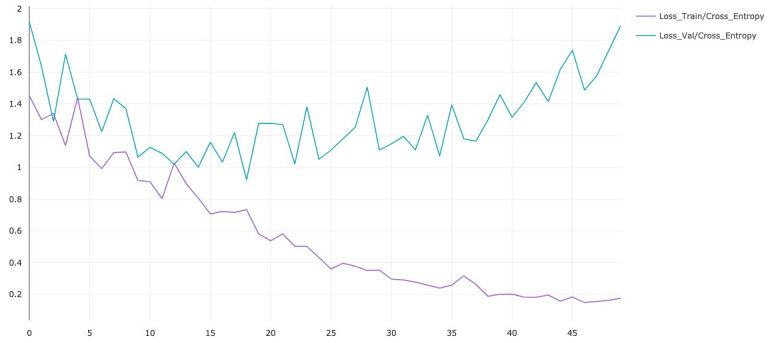
(b) Original input image

Figure 4: (a) Ground truth labels and input image for first image in the test set. (b) We can see that the top of the image is a building (dark grey), the bottom left contains a rider(red), and bottom right is a bus (yellow). There are cars parked in the back (dark blue), and the rider is riding a motorcycle (blue).

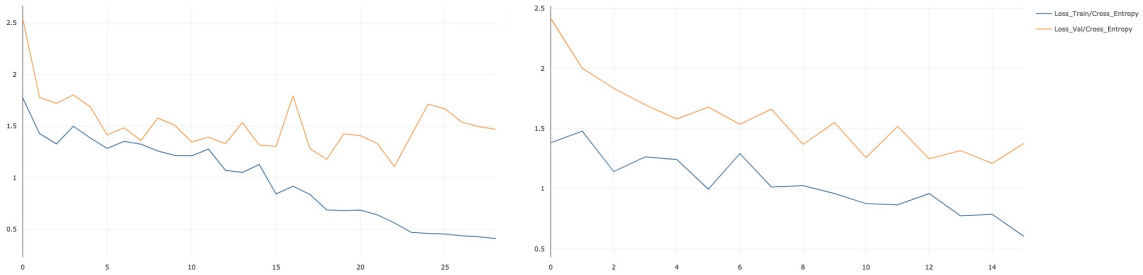
4.1 Baseline Model: Results

Below we have summarized the results from the baseline model and its improvements that we implemented. In general, the pixel accuracy and IoUs appear not to have improved but this may be due to the early-termination of the improved models. These metrics can be found in **Table 2**. The

training and validation losses are in **Figure 5**. The visualization of the predictions on the first image in the test set are in **Figure 6**.



(a) Baseline



(b) Baseline + Weighted

(c) Baseline + Weighted + Augmented

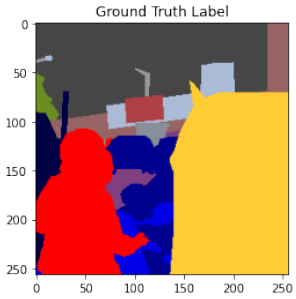
Figure 5: Training and Validation losses. a) The baseline model was able to train for the full 50 epochs, and we can see that it begins to over fit at around epoch 20. b) The baseline + weighted model terminated at 28 and we can see it may not have completed training. c) The baseline weighted augmented experiment also terminated early at epoch 15, so it may also have not finished training.

Metrics	B (50 epochs)	B + W (28 epochs)	B + W + A (15 epochs)
Pixel Accuracy	0.699	0.708	0.691
IoU	**0.159**	0.153	0.144
IoU - 0	0.723	0.717	0.744
IoU - 2	**0.0638**	0	0
IoU - 9	0.236	0.213	0.203
IoU - 17	0.0513	0.0126	0.0386
IoU - 25	**0.598**	0.586	0.572

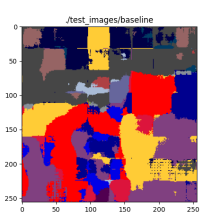
Table 2: Baseline model results on test set. B: Baseline. W: Weighted Loss, A: Augmented Training Data

4.2 Baseline Model: Discussion

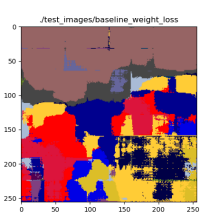
For the baseline model, we can observe that training for 50 epochs was more than enough for this model as the validation loss appears to increase after 20 epochs of training, suggesting that the model started to overfit on the training data after this point. The baseline pixel accuracy of 0.699 and IoU of 0.159 show promise and the segmentation image also appears as expected with distinct patches of objects being segmented.



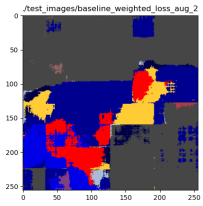
(a) Ground Truth Labels



(b) Baseline



(c) Baseline + Weighted Loss



(d) Baseline + Weighted loss + Augmentation

Figure 6: Outputs from Baseline model and its variations visualization of semantic segmentation on first image of test set. a) Ground truth labels b) We can see that our baseline already favors labeling in patches and is beginning to identify ”person,” ”vehicle,” and ”building.” c) We can observe that there is more ”unlabeled” category (black) present in this image due to the higher weight of learning unlabeled data. d) We can observe a higher prediction rate for the buildings color (dark grey) which may be due to the increased number of corner crops which would have picked up on more buildings.

To improve the model based on the training set label distribution, we tested the barebones baseline with weight adjustments based on a sample set of images. Imbalanced classes impair model learning assumptions about equally distributed examples resulting in poorer performance for the minority class or classes. Therefore, addressing this imbalance can improve model performance accuracy and should be included as best practice in all classification tasks. The expected result was to see an increase in overall and class-wise IoU, with a roughly similar accuracy. This is because the weight adjustments would cause the model to learn more from the small sample of rare classes by adjusting weights relatively more aggressively when processing the error for these classes. However, we observed that overall pixel accuracy of 0.708 and IoU of 0.153 were relatively the same compared to that of the baseline (0.699, 0.159 respectively), while class-wise IoU was slightly smaller for all classes. One explanation for this result is that the number of train epochs is significantly smaller than baseline model, and it is unclear if the validation loss is still decreasing. However, we can see the weighted loss in action with the differing amount of ”unlabeled” pixels compared to the barebones baseline.

We then tested the training data augmentation with the weighted loss improvement to obtain these results. Augmenting data should improve performance because it artificially expands the training dataset, which allows the model to generalize to different variations of images. However, this will also increase the training time per epoch as there are many more data samples to train on (in our case, 5 times more!). We expected to see that the pixel accuracies and IOUs improve with this larger dataset for training due to this. However, we observed from the 15 epochs that was trained, that the pixel accuracy of 0.691 and IOU of 0.144 stayed relatively the same as baseline (0.699, 0.159 respectively). We think this may be due to the fact that the model had not completed training in the 15 epochs that it ran for. We can observe that the validation loss is still on a downward trend, suggesting that the improvements are still yet to be seen given a longer training time. One interesting feature we see in the sample prediction is that we see more ’building’ label being predicted, especially where the label was supposed to be ’bus’ in the lower right corner. This makes sense because the addition of corner crops from data augmentation likely introduced a larger sample of buildings into

the dataset. This inherently increases the prior of seeing a building, thus making it more likely for the model to predict the output to be a building. We would expect that this augmentation technique would perform better if the validation and test sets were similarly augmented as well. In the future, we may consider flipping and rotating the center crops rather than using corner crops to address this prior balancing issue.

4.3 Alternate Model: Results

In this section, we cover the results from all the alternate models we ran, in order to try and improve the results from the baseline model. In general, the pixel accuracy and IoUs appear not to have improved but this may be due to the early-termination of the improved models. These metrics can be found in **Table 3**. The training and validation losses are in **Figure 7**. The visualization of the predictions on the first image in the test set are in **Figure 8**.

(a) Custom leakyReLU Model

(b) Custom leakyReLU Model with Reduced Layers

(c) Transfer Learning Model

(d) U-net Model

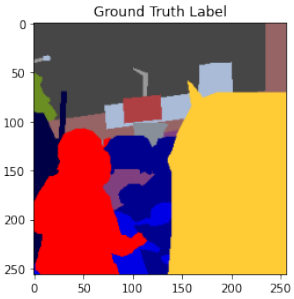
Figure 7: Training and validation losses on alternate architectures.

Metrics	C1 + W + A (25 epochs)	C2 + W + A (25 epochs)	T + W + A (19 epochs)	U + W + A (15 epochs)
Pixel Accuracy	0.679	0.582	**0.723**	0.41
IoU	0.139	0.0925	0.146	0.012
IoU - 0	0.691	0.633	**0.792**	0
IoU - 2	0	0.000371	0.049	0
IoU - 9	0.218	0.0664	**0.326**	0
IoU - 17	0.032	0.0150	0.032	0
IoU - 25	0.566	0.528	**0.744**	0

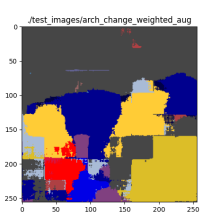
Table 3: Alternate model results on test set. W: Weighted loss. A: Augmented training data. C1: Custom leakyReLU model. C2: Custom leakyReLU model with reduced layers. T: Transfer learning model. U: U-Net model.

4.4 Alternate Model: Discussion

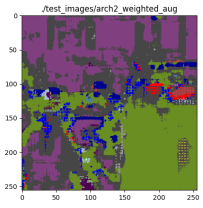
In our first custom architecture, we replaced ReLU activation functions with leakyReLU activation functions, which would allow small negative values and help prevent vanishing gradients. That said,



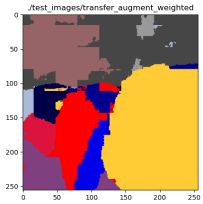
(a) Ground Truth Labels



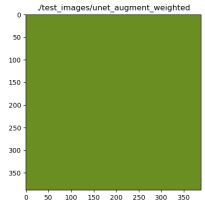
(b) Custom leakyReLU



(c) Custom leakyReLU with reduced layers



(d) Transfer Learning



(e) U-Net

Figure 8: Alternate model predictions on first test image

the original paper on leakyReLU did not demonstrate a significant performance improvement over ReLU **[13]**. Consequently, we anticipated that the model might train faster and gets comparable performance to that of the baseline. However, baseline + weighted loss + augmentation resulted in a pixel accuracy of 0.691 and IoU of 0.144 while our custom architecture resulted in a lower accuracy of 0.679 and IoU of 0.139. Additionally, our custom architecture trained for 13 hours but could only complete 25 epochs; the validation loss did not appear to have plateaued. Visually, the custom architecture did seem to predict the vehicle shape better in the bottom-right corner of the test image, but otherwise predicted roughly the same. We're not too surprised from the lack of improvement of leakyReLU on top of the baseline and conclude that leakyReLU may not be well-suited for this problem.

In our second custom architecture, in addition to replacing ReLU activation functions with leakyReLU activation functions, we reduced the number of layers in both the encoder and decoder parts of the model from 5 to 3. Simplifying the model architecture by reducing the number of layers could increase training speed and reduce overfitting, which can be seen by the 1.0 difference between train and validation loss curves in the baseline + weighted loss + augmentation model. However, we expected that such a large decrease in the number of layers should result in a poorly performing model. As suspected, the resulting model's pixel accuracy was 0.582 and IoU was 0.0925, which was much lower than the baseline + weighted loss + augmentation model (0.691, 0.144 respectively). We were correct in our hypothesis that this smaller architecture model would not learn the representations necessary to solve the task. This is further highlighted by the grossly incorrect predictions on the test image, where "vehicle" was classified as "vegetation," and "building" was classified as "road."

We then experimented with transfer learning, using the ResNet-18 model **[5]** pre-trained using the ImageNet dataset. The benefit of transfer learning is to take advantage of deep networks pre-trained on large datasets with similar goals to our problem. The goal of the model was to "classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals." Consequently, we hypothesize that the model would improve our baseline performance noticeably, as the saved weights from the ResNet-18 model should have learned representations of the shapes for the objects we're trying to segment. Where the model might falter is in actually segmenting the outlines for each image, as the goal of ImageNet was just to classify the whole image into a particular category. Our hypothesis was correct: our transfer learning model ended up with the highest performance of all of our experiments, with a pixel accuracy of 0.723, an IoU of 0.146, and the highest class-wise IoU

for each category. The validation loss had an interesting phenomenon in which the validation loss spiked up for 2-3 epochs twice during training; however, the model was able to correct itself for an overall low loss. The visualization of the test image was most accurate, with clear attempts to define "vehicle," "person," and "motorcycle." The model did not do as well defining the "building" at the top of the image; this may represent the ImageNet dataset's bias towards smaller, more well-defined objects over large structures that take up the whole image.

Lastly, we attempted to recreate the U-Net model. We expected the U-Net model to result in a large improvement for a few important tricks they used: (1) introducing the decoding layers, which results in a somewhat symmetrical architecture, increasing the resolution in the output layer; (2) use long skip connection via concatenation, so that it helps with the vanishing gradient issue, and reuse features computed in encoding layer, giving better context for the pixels near the boundary; (3) pre-compute the weight map to address the class imbalance problem and force the network to learn the boundaries between different regions. Unfortunately, we were not able to get productive results from our implementation of the model. Even after training for over 24 hours, the model was only able to complete 15 epochs. For the most part, we saw that the all the metrics we are using to evaluate this network remain flat, oscillating about a certain point. When we try to look at the general trend of the loss plots, we see that the training curve is trying to go downwards (but ever so slightly) while the validation curve is going up from the get go. And the output segmentation mask is monotone which further bolsters the sign that the network has not learnt anything despite augmenting the dataset and using the weighted loss function and a 5 layer deep model with 3 convolutions in each layer. We hypothesize four areas where our implementation may have been incorrect. (1) missing batch norm layers and the way we initialised our weights. (2) in order to match the size of the outputs, we center-cropped our labels to 388×388, but this might have resulted in a mismatch in the information contained from the outputs from the labels. Instead, we should have *resized* the labels down to 388×388 so that it represents the same information as the outputs. (3) we didn't pre-compute the weight map to compensate the class imbalance, and didn't ask our model to focus on the boundaries. (4) we could have tried a smaller learning rate to prevent oscillations when optimizing the loss function. These ideas to fix the model can be implemented in future work.

5 Conclusion

We were successful in implementing a convolutional neural network to solve the problem of semantic segmentation in the India Driving Dataset, with a baseline pixel accuracy of 0.699 and IoU of 0.159. However, we saw mixed results from our attempts to improve the baseline with weighted loss and augmented training data, as well as experimenting with model architecture, transfer learning, and the U-Net model. For future work, we plan to allow our models to train for more epochs to ensure that they aren't underfitting and look to address implementation errors in our U-Net model.

6 Individual Contributions

We all contributed in equal amounts to each of the different stages of the project. Every line of code was essentially reviewed and helped in debugging by every person (with logic) and the reports were also brainstormed as a group.

Geeling Chau

- Implemented and ran data augmentation and weighted baseline models.
- Implemented prediction visualization code.
- Wrote up report for improved baseline models.
- Packaged up code for submission.

Keshav Rungta

- Set up Comet to log all our experiments
- Wrote the baseline model
- Debugged transformations in dataloader, metrics

- Debugged U-Net (or to whatever current state we could get it to)
- Wrote abstract, introduction, related works, dataset, Metrics, training details, some of the discussion in various sections

Anshuman Dewangan

- Implemented baseline train, val, and test functions (with early stopping), second custom architecture model
- Explored optimizing GPU memory... but failed and had to implement image cropping instead
- Helped debug pixel accuracy and iou
- Brought together all code and logging to prepare for experimentation
- Outlined framework for report; wrote discussion, some methods, results, abstract, and conclusion

Margot Wagner

- Wrote custom architecture model
- Implemented transfer learning model
- Implemented U-Net model
- Wrote methods, some discussion, and dataset information

Jin-Long Huang

- Implemented and tested Pixel accuracy function, overall IoU and class-wise IoU.
- Debugged and ran U-Net based on the architecture Margot wrote.
- Wrote general discussion for U-Net.

7 References

[1] CAESAR, H., BANKITI, V., LANG, A. H., VORA, S., LIONG, V. E., XU, Q., KRISHNAN, A., PAN, Y., BALDAN, G., AND BEIJBOM, O. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027* (2019).

[2] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).

[3] GEIGER, A., LENZ, P., STILLER, C., AND URTASUN, R. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)* (2013).

[4] GLOROT, X., AND BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), JMLR Workshop and Conference Proceedings, pp. 249–256.

[5] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[6] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[7] LIN, T., DOLLÁR, P., GIRSHICK, R., HE, K., HARIHARAN, B., AND BELONGIE, S. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 936–944.

[8] LIN, T.-Y., MAIRE, M., BELONGIE, S., BOURDEV, L., GIRSHICK, R., HAYS, J., PERONA, P., RAMANAN, D., ZITNICK, C. L., AND DOLLÁR, P. Microsoft coco: Common objects in context, 2015.

[9] LOSHCHILOV, I., AND HUTTER, F. Fixing weight decay regularization in adam. *CoRR abs/1711.05101* (2017).

[10] MEYER, M., AND KUSCHK, G. Automotive radar dataset for deep learning based 3d object detection. In *Proceedings of the 16th European Radar Conference* (2019), pp. 129–132.

[11] RONNEBERGER, O., FISCHER, P., AND BROX, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 Lecture Notes in Computer Science* (2015), Springer.

[12] VARMA, G., SUBRAMANIAN, A., NAMBOODIRI, A., CHANDRAKER, M., AND JAWAHAR, C. V. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *IEEE Winter Conf. on Applications of Computer Vision (WACV 2019)* (2019).

[13] XU, B., WANG, N., CHEN, T., AND LI, M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

Appendix

Layer Breakdown for each Model

Layer	Input, Output, Kernel, Stride, Padding, Activation function
Conv2D	3, 32, 3, 2, 1, ReLU
Conv2D	32, 64, 3, 2, 1, ReLU
Conv2D	64, 128, 3, 2, 1, ReLU
Conv2D	128, 256, 3, 2, 1, ReLU
Conv2D	256, 512, 3, 2, 1, ReLU
ReLU	inplace
ConvTranspose2d	512, 512, 3, 2, 1, ReLU
ConvTranspose2d	512, 256, 3, 2, 1, ReLU
ConvTranspose2d	256, 128, 3, 2, 1, ReLU
ConvTranspose2d	128, 64, 3, 2, 1, ReLU
ConvTranspose2d	64, 32, 3, 2, 1, ReLU
ConvTranspose2d	32, 27, 3, 2, 1, ReLU
Softmax	

Table 4: Baseline model layers

Layer	Input, Output, Kernel, Stride, Padding, Activation function
Conv2D	3, 32, 3, 2, 1, leakyReLU
Conv2D	32, 64, 3, 2, 1, leakyReLU
Conv2D	64, 128, 3, 2, 1, leakyReLU
Conv2D	128, 256, 3, 2, 1, leakyReLU
Conv2D	256, 512, 3, 2, 1, leakyReLU
leakyReLU	inplace
ConvTranspose2d	512, 512, 3, 2, 1, leakyReLU
ConvTranspose2d	512, 256, 3, 2, 1, leakyReLU
ConvTranspose2d	256, 128, 3, 2, 1, leakyReLU
ConvTranspose2d	128, 64, 3, 2, 1, leakyReLU
ConvTranspose2d	64, 32, 3, 2, 1, leakyReLU
ConvTranspose2d	32, 27, 3, 2, 1, leakyReLU
Softmax	

Table 5: First experimental leaky ReLU model layers

Layer	Input, Output, Kernel, Stride, Padding, Activation function
Conv2D	3, 32, 3, 2, 1, leakyReLU
Conv2D	32, 64, 3, 2, 1, leakyReLU
leakyReLU	inplace
ConvTranspose2d	64, 64, 3, 2, 1, leakyReLU
ConvTranspose2d	64, 32, 3, 2, 1, leakyReLU
ConvTranspose2d	32, 27, 3, 2, 1, leakyReLU
Softmax	

Table 6: Second experimental leaky ReLU model with reduced layers

Layer	Input, Output, Kernel, Stride, Padding, Activation function
Conv2D	3, 64, 7, 2, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Conv2D	64, 128, 3, 1, 0, ReLU
Conv2D	128, 128, 3, 1, 0, ReLU
Conv2D	128, 128, 3, 1, 0, ReLU
Conv2D	128, 128, 3, 1, 0, ReLU
Conv2D	128, 256, 3, 1, 0, ReLU
Conv2D	256, 256, 3, 1, 0, ReLU
Conv2D	256, 256, 3, 1, 0, ReLU
Conv2D	256, 256, 3, 1, 0, ReLU
Conv2D	256, 512, 3, 1, 0, ReLU
Conv2D	512, 512, 3, 1, 0, ReLU
Conv2D	512, 512, 3, 1, 0, ReLU
Conv2D	512, 512, 3, 1, 0, ReLU
Linear	512, 512, –, –, –, –
Conv2D	512, 27, 1, 1, 0, –
ConvTranspose2D	27, 27, 64, 32, 16, –
Softmax	

Table 7: Transfer learning model layers (ResNet-18)

Layer	Input, Output, Kernel, Stride, Padding, Activation function
Conv2D	3, 64, 3, 1, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Conv2D	64, 128, 3, 1, 0, ReLU
Conv2D	128, 128, 3, 1, 0, ReLU
Conv2D	128, 256, 3, 1, 0, ReLU
Conv2D	256, 256, 3, 1, 0, ReLU
Conv2D	256, 512, 3, 1, 0, ReLU
Conv2D	512, 512, 3, 1, 0, ReLU
Conv2D	512, 1024, 3, 1, 0, ReLU
Conv2D	1024, 1024, 3, 1, 0, ReLU
ConvTranspose2d	1024, 512, 2, 2, 0, ReLU
Conv2D	1024, 512, 3, 1, 0, ReLU
Conv2D	512, 512, 3, 1, 0, ReLU
ConvTranspose2d	512, 256, 2, 2, 0, ReLU
Conv2D	512, 256, 3, 1, 0, ReLU
Conv2D	256, 256, 3, 1, 0, ReLU
ConvTranspose2d	256, 128, 2, 2, 0, ReLU
Conv2D	256, 128, 3, 1, 0, ReLU
Conv2D	128, 128, 3, 1, 0, ReLU
ConvTranspose2d	128, 64, 2, 2, 0, ReLU
Conv2D	128, 64, 3, 1, 0, ReLU
Conv2D	64, 64, 3, 1, 0, ReLU
Softmax	

Table 8: U-Net model Layers