Convolutional Neural Networks for Semantic Segmentation with the India Driving Dataset

Yunhai Han
Department of Mechanical and Aerospace Engineering
University of California, San Diego
La Jolla, CA 92037
y8han@eng.ucsd.edu

Hanyang Xu
ECE
University of California, San Diego
La Jolla, CA 92037
hax032@ucsd.edu

Xinyuan Lu
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92037
xil069@eng.ucsd.edu

Zhenzhen Zhu
Department of Mathematics
University of California, San Diego
La Jolla, CA 92037
zhz065@ucsd.edu

Abstract

This paper aims to perform semantic segmentation task that recognizes objects from multi-classes in realistic scenes. We built full convolutional neural networks and trained them from scratch to predict the segmentation masks of the images using the India Driving Dataset. Our baseline model with 10 layers and ReLU activation achieved an average pixel accuracy of 0.79 and an average IoU of 0.26. The data augmented model that mirrored, rotated, and cropped images had an average pixel accuracy of 0.69 and an average IoU of 0.18. The third model addressed the imbalanced class problem by implementing weighted loss, yielding an average pixel accuracy of 0.74 and an average IoU of 0.20. We then customized the baseline model by reducing two layers and reordering batch normalization, and obtained an average pixel accuracy of 0.76 and an average IoU of 0.22. The transfer learning model replaced the encoder part of the baseline model with the pre-trained architecture *Resnet-18*, resulting in an average pixel accuracy of 0.76 and an average IoU of 0.22. We also implemented the U-net architecture, which achieved an average pixel accuracy of 0.67 and an average IoU of 0.16. Our best model is the baseline model in terms of average pixel accuracy and average IoU. We found that weighted loss is the best practice of addressing imbalanced class problem as only the FCN network trained with weighted loss was able to recognize the classes of sidewalk and billboard. We observed no significant improvement in the performance of the models when using techniques such as data augmentation, transfer learning, or U-net architecture, compared with the baseline model.

1 Introduction

In recent years deep Convolutional Neural Networks (CNNs) have emerged as the leading tools for object recognition and segmentation. In particular, deep CNNs are crucial tools for the task of semantic segmentation, which classifies each pixel of a given image to some pre-determined category. Semantic segmentation is useful in a broad range of Computer Vision problems such as autonomous driving, satellite image processing, and medical image diagnostics. In such tasks it is important for the models to understand the context in the environment in which they are operating.
This paper explored semantic segmentation using a subset of the India Driving Dataset, with a goal of recognizing objects from the 27 annotated object classes (including road, sidewalk, car, billboard,

sky, and etc.) in realistic scenes. This subset of the dataset contains about 7000 images with a 4:2:1 split in training, validation, and test data.
One common challenge facing image segmentation tasks is rear or imbalanced class. Imbalanced class problem occurs when the number of observations belonging to one class is significantly lower than those belonging to the other classes. In this situation, the predictive model developed using CNN algorithms could be biased and inaccurate. In our dataset, it happens that some classes such as sidewalk and billboard are less frequent than other classes such as sky and road. We addressed the imbalanced class problem by using the multiclass cross-entropy loss and/or by implementing weighted loss algorithm that weights the infrequent classes more.
Training deep CNNs is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. To address that, we implemented batch normalization. Batch normalization is a technique that standardizes the inputs to a layer of each mini-batch through re-centering and re-scaling. It scales and shifts the normalized variable to a best suited range through learnable parameters γ and β, which stabilizes and accelerates the learning process of deep CNNs.
We used Xavier weight initialization to prevent layer activation outputs from exploding or vanishing during the course of a forward pass. The math required to complete a forward pass entails a succession of matrix multiplications between inputs and weights. Xavier initialization sets a layer's weights to values chosen from a random uniform distribution bounded between $\pm\sqrt{6}/\sqrt{n_i + n_{i+1}}$, where n_i is the fan-in to the layer and n_{i+1} is the fan-out from that layer[1]. Xavier weight initialization is useful as it maintains near identical variances of its weight gradients across layers.

2 Related Work

This project built upon many aspects of previous Deep Learning research works. Specifically, we implemented and built upon some previous Deep Learning Architectures such as FCN[2], U-Net[3], ResNet[4]. We also improved our network with many popular optimization techniques such as Xavier Weight Initialization[1], Data Augmentation[5], Weighted Loss Function[6], Adam Optimizer[7].

3 Methods

3.1 Baseline Model

Our baseline model used ReLU as the activation function for the hidden layers since ReLU overcomes the vanishing gradient problem, allowing the model to learn faster and perform better. Therefore, ReLU is good to use in our CNN. The output layer uses Softmax to compute the final score for each class.
We were determined to optimize the CrossEntropyLoss criterion because it is useful to train a classification problem with multiple classes (in our case, 27 classes). The multiclass cross-entropy loss is particularly useful when the training set is imbalanced, which is just our case. We drew this insight from the TORCH.NN.MODULES.LOSS source code.
We used the Adam gradient descent optimizer from the *torch.optim* package. We also enabled early stopping to train the model – the training stops when 3 successive validation loss go up.
Other important configurations of our baseline model included input image size 512×960, learning rate of 0.01, batch size of 4, and 40 epochs.

3.2 Improving Baseline Model

3.2.1 Data Augmentation

Early Deep Neutral Networks rely heavily on big data to avoid over-fitting. However, the amount of data that areas such as medical image analysis have access is not large enough to address the over-fitting naturally through training. Data Augmentation techniques inflates the training data size and artificially introduces variability into the training data-set by either image manipulation or using deep learning approaches. We implemented some image manipulation approaches include: Random crop, Rotation, and Mirror Flip. The specific effect of each operation is as follows:

- Random Crop: This operation crops the original input images to a specific height and width. The cropped portion location is randomly generated but is guaranteed to be within the original image and satisfied the height and width constraints.
- Rotation: This operation rotates the images by 3 to -3 degree randomly.
- Mirror Flip: This operation flips the image along the y-axis.
- Central crop: Similar to that of the random crop, but it is guaranteed that the image is cropped from the center with a specific size

All the above operations are set up to have 50% chance of being applied to each input image. These operation are integrated into PyTorch's Transforms class and Dataloader class which apply these data augmentations automatically to the mini-batch data and sent them to the training algorithm.

3.2.2 Imbalanced Class Problem

Imbalanced Class Problem refers to the imbalance of the total appearance of different classes in the training dataset. Some class appears less than other classes for many reasons such as the difference of size and difference of appearance frequency. For example, in the India Driving Dataset, the sky class has large label count since the sky always occupies a large of portion in the image. The fence, on the other hand, is more rear to appear in the image and occupies less portion of the image when it does appear. We mitigated the Imbalanced Class Problem with the weighted loss approach. We first analyzed the overall proportion of each class from the entire dataset when we first loaded the dataset into the data loader. Then, we weighted the update for a specific class by the inverse of its proportion calculated. In this way, the rare class updates will have a larger impact on the overall model than the familiar class updates.

3.3 Experimentation

The customized baseline architecture is shown below:

Table 1: Customized baseline architecture

Layer Index	Layer Type	Layer Dimension (In/Out/kernal/stride/actication)
1	Convolutional(Encoder)	3/32/3/2/ReLu
2	Convolutional(Encoder)	32/128/3/2/ReLu
3	Convolutional(Encoder)	128/256/3/2/ReLu
4	Convolutional(Encoder)	256/512/3/2/ReLu
5	Trans-Convolutional(Decoder)	512/512/3/2/ReLu
6	Trans-Convolutional(Decoder)	512/256/3/2/ReLu
7	Trans-Convolutional(Decoder)	256/128/3/2/ReLu
8	Trans-Convolutional(Decoder)	128/32/3/2/ReLu
9	Classifier	32/27/1/1/Softmax

The main differences are the layer deduction (remove one layer for each encoder and decoder) and the order of batch normalization and ReLU activation function. It is still an open question that whether normalization comes first and then follow by activation or vice verse. Thus, it is interesting to conduct such experiment to compare the model performances. In the **raw baseline model**, ReLu is first applied on the layer outputs and then get normalized. Intuitively, this could allow more non-zero values. It is possible that by doing so, more information is carried into the next layer, which could provide better performances. The **customized baseline model** takes the inverse order and the experiment results show clear differences. Also, as professor said in the lectures, the more number of layers, the better performance we could expect. The other parts, including parameter initialization methods and gradient descent optimization, are the same for these two models.

The transfer learning model architecture is shown below:

Table 2: Transfer learning model architecture

Layer Index	Layer Type	Layer Dimension (In/Out/kernal)
1	Convolutional(Encoder)	3/64/7
2	Convolutional(Encoder)	64/64/3
3	Convolutional(Encoder)	64/128/3
4	Convolutional(Encoder)	128/128/3(downsample+upsample)
5	Convolutional(Encoder)	128/256/3
6	Convolutional(Encoder)	256/256/3(downsample+upsample)
7	Convolutional(Encoder)	256/512/3
8	Convolutional(Encoder)	512/512/3(downsample+upsample)
9	Trans-Convolutional(Decoder)	512/512/3/2/ReLu
10	Trans-Convolutional(Decoder)	512/256/3/2/ReLu
11	Trans-Convolutional(Decoder)	256/128/3/2/ReLu
12	Trans-Convolutional(Decoder)	128/32/3/2/ReLu
13	Classifier	32/27/1/1/Softmax

We implemented CovNet as an initialization[4]. The encoder layers were replaced by the pre-trained layers from **Resnet-18** (with the full-connected layer removed). Here, the weights for all of the encoder layers are initialized using the pre-trained values. The decoders are the same as the baseline model. Since the wights and biases of encoders layers are pre-trained, it require less data for training on a new task. The network architecture was obtained by the return value of *models.resnet18.named_parameters()*. Both encoder and decoder were trained at learning rate 0.005, with 16 batches, 40 training epochs, and early stopping in the case of 3 consecutive validation loss going up.
The U-net architecture is shown below:

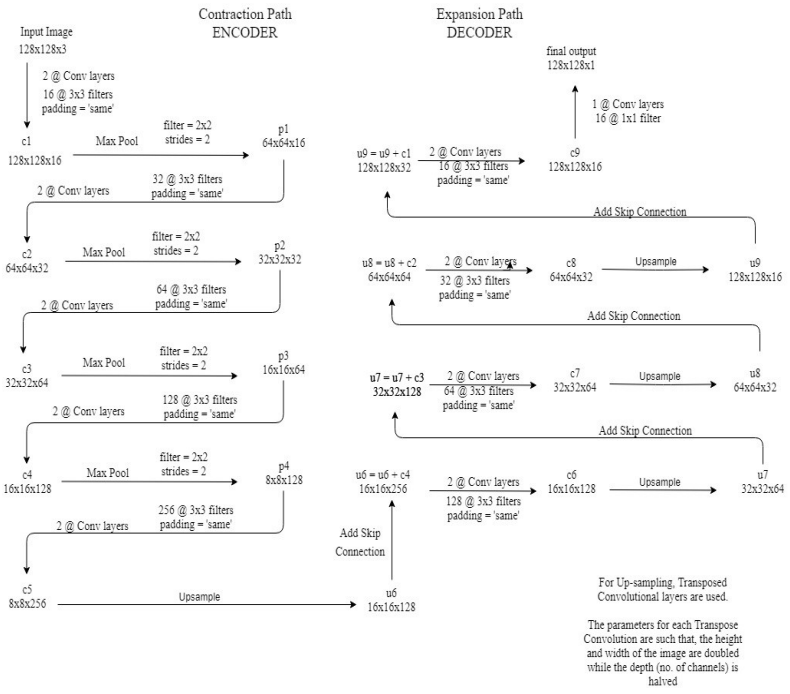

Figure 1: U-net model: Basic design of the U-net.

The architectural details of U-net is shown in the graph above. The network consist of 5 layers of down convolution and 5 layers of up convolutions. Each layer consist of 2 convolutions and 2 ReLU activations. The up layers will concatenate the layers from the output from the previous down layers. The convolution layer has 3×3 kernel.

4 Results

4.1 Pixel accuracy and IoU

Table 3: Validation set pixel accuracy, average IoU and IoU of the classes mentioned

Model	**Accu**	**IoU**	**IoU of particular classes**				
			road(0)	sidewalk(2)	car(9)	billboard(17)	sky(25)
Baseline model	0.79	0.26	0.87	0	0.46	0	0.90
Data augmented model	0.69	0.18	0.73	0	0.34	0	0.82
Weighted loss model	0.74	0.20	0.79	0.25	0.42	0.18	0.84
Customized model	0.76	0.22	0.85	0	0.36	0	0.89
Transfer learning	0.76	0.22	0.86	0	0.36	0	0.88
U-net	0.67	0.16	0.78	0	0.05	0.01	0.89

4.2 Baseline model

Our baseline model created adequate loss curves on the training and validation sets, as shown below in Fig 2. The loss curves decay exponentially without increasing across training epochs, which implies that there is no overfitting.

Figure 2: Baseline model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the baseline model are shown below in Fig 3 and Fig 4, respectively.

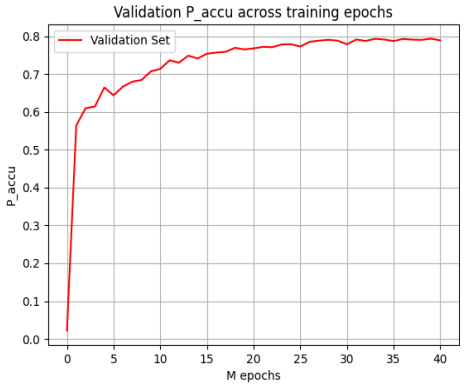

Figure 3: Baseline model average pixel accuracy on validation set

Figure 4: Baseline model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 5 shows the segmented output for the first image int the *test.csv* overlaid on the image.

Figure 5: Baseline model: Segmented output for the first image in *test.csv* overlaid on the image

4.3 Improving baseline model: Data Augmentation

Fig 6 shows the training and validation loss curves obtained by the data augmented model.

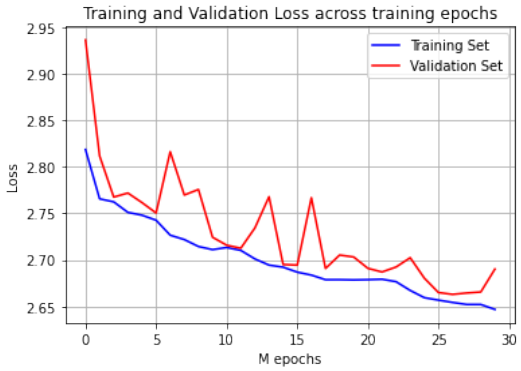

Figure 6: Data augmented model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the data augmented model are shown below in Fig 7 and Fig 8, respectively.

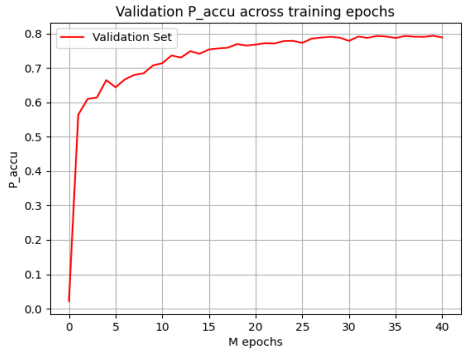

Figure 7: Baseline model average pixel accuracy on validation set

Figure 8: Data augmented model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 9 shows the segmented output for the first image int the *test.csv* overlaid on the image.

Figure 9: Data augmented model: Segmented output for the first image in *test.csv* overlaid on the image

4.4 Improving baseline model: Weighted Loss

Fig 10 shows the training and validation loss curves obtained by the model using weighted loss.

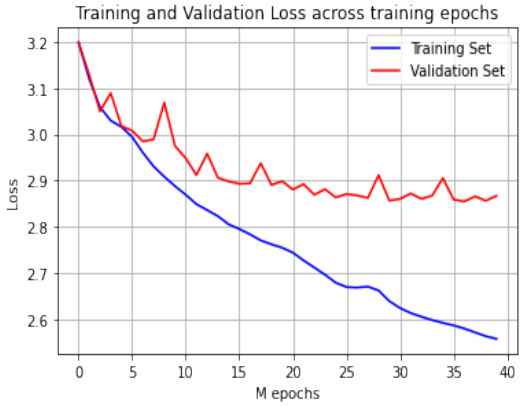

Figure 10: Weighted loss model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the weighted loss model are shown below in Fig 11 and Fig 12, respectively.

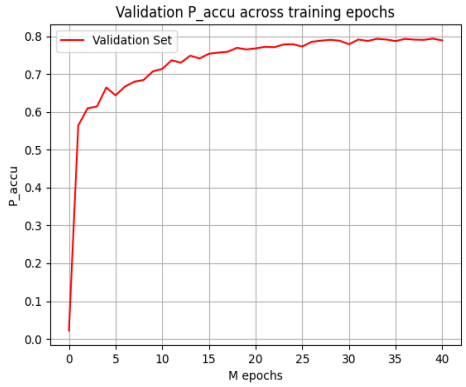

Figure 11: Baseline model average pixel accuracy on validation set

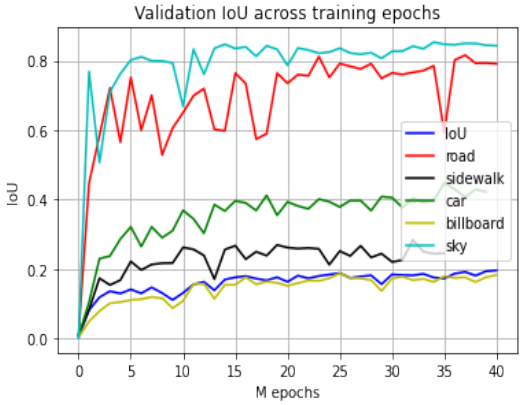

Figure 12: Weighted loss model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 13 shows the segmented output for the first image int the *test.csv* overlaid on the image.

Figure 13: Weighted loss model: Segmented output for the first image in *test.csv* overlaid on the image

4.5 Customized architecture

As described in Table 1, we modified the baseline by removing two layers, reordering batch normalization, and using ReLU activation. Then this customized model has the following training and validation loss curves as shown in Fig 14.

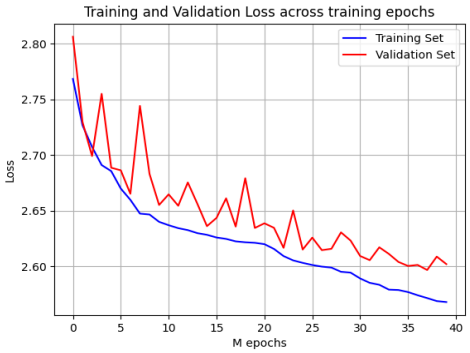

Figure 14: Customized model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the customized model are shown below in Fig 15 and Fig 16, respectively.

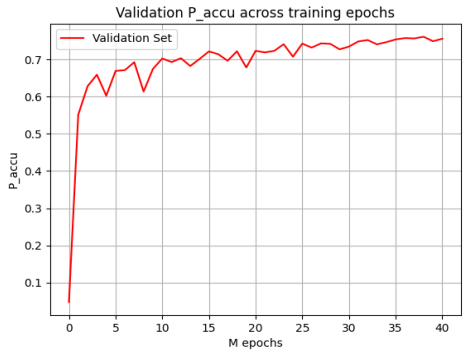

Figure 15: Customized model average pixel accuracy on validation set

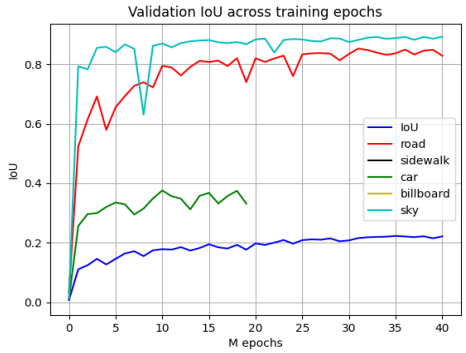

Figure 16: Customized model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 17 shows the segmented output for the first image int the *test.csv* overlaid on the image.

Figure 17: Customized model: Segmented output for the first image in *test.csv* overlaid on the image

4.6 Transfer learning

As described in Table 2, we replaced the encoder part in baseline model with the pre-trained **Resnet-18** and removed the full-connected layer. Then this transfer learning model has the following training and validation loss curves as shown below in Fig **??**.

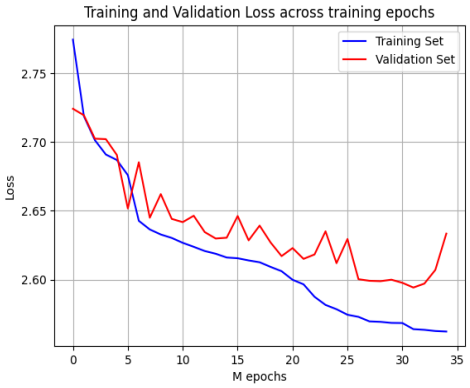

Figure 18: Transfer learning model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the transfer learning model are shown below in Fig 19 and Fig 20, respectively.

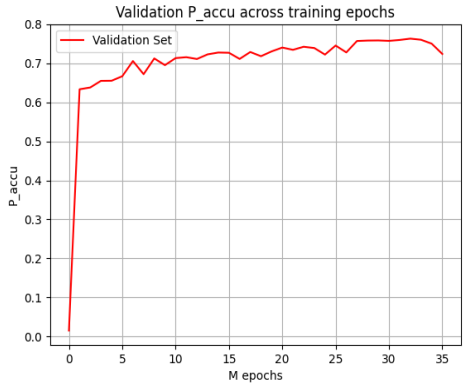

Figure 19: Transfer learning model average pixel accuracy on validation set

Figure 20: Transfer learning model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 21 shows the segmented output for the first image int the *test.csv* overlaid on the image.

Figure 21: Transfer learning model: Segmented output for the first image in *test.csv* overlaid on the image

4.7 U-net

With a basic architecture described in Fig 1, our U-net model has the following training and validation loss curves as shown below in Fig 22.

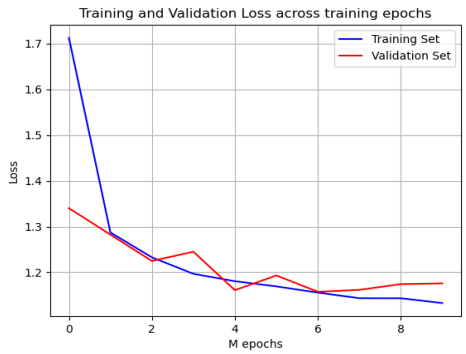

Figure 22: U-net model training and validation loss curve

The average pixel accuracy, average IoU and IoU of particular classes of the U-net model are shown below in Fig 23 and Fig 24, respectively.

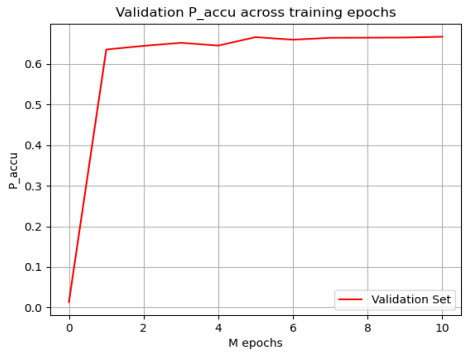

Figure 23: U-net model average pixel accuracy on validation set

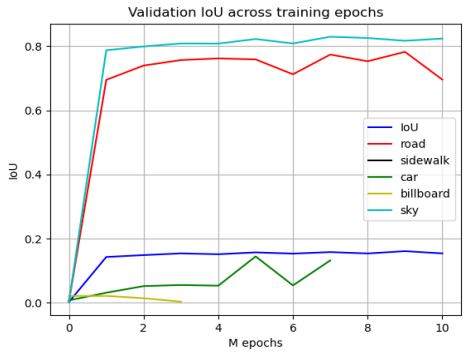

Figure 24: U-net model on validation set average IoU without the *unlabeled class (26)* and IoU of the particular classes *road(0), sidewalk(2), car(9), billboard(17), sky(25)*

Fig 25 shows the segmented output for an image int the *test.csv* overlaid on the image.

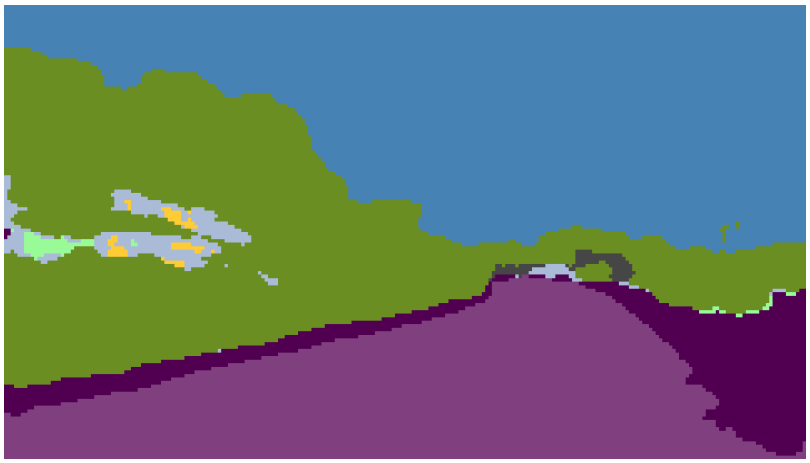

Figure 25: U-net model: Segmented output for the first image in *test.csv* overlaid on the image

We can see a very clear image of the freeway and the pole on the freeway.

5 Discussion

5.1 Baseline model

We observed from Table 3 that the baseline model reached an average IoU of 0.26 on validation dataset, which is lower than the 0.55 expected average IoU specified in the assignment. This is because the method we used to calculate the average IoU is firstly calculating a per-class intersection and a per-class union, then calculating a per-class IoU as (intersection of class C) / (union of class C) for each of the C classes, and finally averaging all of the class-wise IoUs. This way of calculation weights the all objects equally.
This baseline model, despite being simple and effective,has a number of drawbacks. One such drawback is poor resolution at the boundaries due to loss of information from the process of encoding.

5.2 Improving baseline model: Data Augmentation

We observed from Table 3 that the performance of FCN network trained on the data augmented dataset is all around worse than the baseline FCN network. This performance difference could be due to the fact that the dataset we trained our model on is already large and each class is well labeled. Therefore, the benefit of data augmentation will not be obvious even if there is any. Also, the image augmentation might prevent the network to learn some obvious patterns in the regular images which could be very useful when running on the test set.

5.3 Improving baseline model: Weighted Loss

We observed from Table 3 that the FCN network trained with the weighted loss function performs better at recognizing sidewalk and billboards. All the other networks could not recognize sidewalk and billboards due to their small image area and more rear appearance. The weighted loss model recognizing these under-represented objects proved that adding the weighted loss function achieved its goal. However, the overall performance is still worse than the baseline due to the learning from the large and frequently appeared objects is smaller than the learning from the baseline FCN, therefore, the average pixel accuracy will slightly decrease correspondingly.

5.4 Customized architecture

We observed from Table 3 that the performance of customized baseline architecture is worse than the baseline model(all of the accuracy of IoU values are smaller). The possible reason is that customized baseline architecture takes the inverse order(batchnormalization first and then ReLu), thus less information is carried into next layer. Also, since two layers are removed from customized baseline architecture, the network lost units that are useful for the task.

5.5 Transfer learning

In general, transfer learning is beneficial as we do not need to train an entire CNN from scratch with random initialization. Instead, we can rely on some ConvNet pre-trained on a large dataset like ImageNet, and therefore eliminate the issue of not having a dataset of sufficient size. However, we observed in Table 3 that our transfer learning model performed worse than the baseline model. The possible reason is that the learning rate for the pre-trained encoder layers is not appropriate. Unlike the training process for baseline, transfer learning should adopt two different set of learning set: one for encoder and the other for decoder. In this case, we may try several times to find the best combination. However, due to the heavy computation, it is hard for us to find it within limited time.

5.6 U-net

Surprisingly, the U-net performed not the best among all the models, as we seen the in the IoU graph (Fig 24), accuracy graph (Fig 23) and Loss graph (Fig 22)and the really nonsense annotation (Fig 25). There are several possible reasons for that. First of all, it might be a little bit under-fitting, since we might have a rather short early stopping interval, which made the model tend to stop at the local minimal. If we turned off the early stopping, we can see the loss will goes even further. From Fig 22 , we could see that both the training set and validation set have the tendency to go further down, considering the model has only been trained for 10 epochs and early stopped at the 7-th epoch. We could change the model to be further trained, but for the sake of the fairness, we decided to leave it as it is. Something worth mentioning here is that similar to all the other models, U-net without weighted training failed to recognize rather small objects that appeared far less in the training set. The model was L_2 regularized and we used Adam optimization to make sure that we would not be stuck at the local minimal for a long time.

6 Authors' Contributions and References

6.1 Yunhai Han

I created the codes for performance metrics, including pixel accuracy, average IoU, IoU of mentioned class. I also completed the baseline model architecture and customized model architecture and drafted the corresponding parts in the report.

6.2 Hanyang Xu

Xinyuan and I developed the data loader which includes data loading, data augmentation and data weighting. I tested Data Augmentation and Weighted Loss Training. Xinyuan and I implemented U net architecture. I also wrote the corresponding sections in the report that I was in charge of developing.

6.3 Xinyuan Lu

Hanyang and I developed the data loader which includes data loading, data augmentation and data weighting. We tested Data Augmentation and Weighted Loss Training, implemented U net architecture. I wrote the unet section of the report.

6.4 Zhenzhen Zhu

As of coding, I sketched part of utils.py and transfer learning. In report, I wrote Abstract, Intro, the parts about the baseline model, and References.

References

[1] James Dellinger. *Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming.* https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79

[2] Jonathan Long, Evan Shelhamer, Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation.* arXiv:1411.4038. [cs.CV]. 8 Mar 2015. https://arxiv.org/abs/1411.4038

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* arXiv: 1505.04597v1.[cs.CV]. 18 May 2015. https://arxiv.org/pdf/1505.04597.pdf

[4] Sasank Chilamkurthy. *Transfer Learning for Computer Vision Tutorial.* https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

[5] Shorten, C., Khoshgoftaar, T.M. *A survey on Image Data Augmentation for Deep Learning.* J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

[6] Source code for torch.nn.modules.loss – PyTorch 1.71 documentation. https://pytorch.org/docs/stable/_modules/torch/nn/modules/loss.html

[7] Torch.optim – PyTorch 1.71 documentation. https://pytorch.org/docs/stable/optim.html#torch.optim.Adam