CSE 253 Programming Assignment 3

Astuti Sharma
A53285392
asharma@eng.ucsd.edu

Divyanshu Mund
A53281339
dmund@eng.ucsd.edu

Kunal Jain
A53309158
kujain@eng.ucsd.edu

Ria Aggarwal
A53304030
r2aggarw@eng.ucsd.edu

Sai Akhil Suggu
A53284020
ssuggu@ucsd.edu

Abstract

Deep convolutional neural networks have been applied to a broad range of problems and tasks within Computer Vision. In this paper we implement multiple deep CNN architectures from scratch and train them to predict the segmentation masks of the images from Cityscapes dataset. We then experimented with transfer learning on pretrained VGG, Inception from Pytorch Library, also implemented U-Net architecture from scratch and reviewed their performance on semantic pixel-wise segmentation. All these networks downsample the image using an encoder made of Convolutions and Max pooling layers and upsample with a decoder using Deconvolution layers. We then explored different modifications of the traditional networks by changing number of layers and network design, showing the advantages and disadvantages of each ablation. These assessments show that our own network (Megatron) inspired from Inception gives the best performance on Cityscapes dataset

1 Introduction

To understand an image, we want to understand it's contents. To acheive this task , we need to segment the image into relevant parts, namely semantic segmentation. Semantic segmentation has numerous applications from scene understanding, image captioning to autonomous driving. Apart from recognising what the image contains, we also have to output the boundaries of each object. Therefore, unlike classification, we need to infer labels for each pixel, so that each label has one enclosed region. This task is although not new, we were unable to achieve satisfactory performance with traditional machine learning tasks. Prior to introdution of deep CNN's over computer vision, We used to have classifiers like TextonForest[1] and Random Forest based classifiers for semantic segmentation. As seen in image classification, convolutional neural networks (CNN) have been enormously successful on segmentation problems. With the recent advances in deep neural networks, deep Convolutional neural networks are shown to be much better at this task, even better than humans in some tasks.

Inspired from this, we are using CNN's to do semantic segmentation. In this paper, we implemented FCN network from scrach using Pytorch library to do the necessary task. We then compared its performance with transfer learning on VGG and Inception networks from Pytorch Library

2 Literature review

During initial days of deep learning, most of the classification networks had fully connected layers, thus requiring same size images which lead to most approaches being based on patch classification where we classify pixels based on patch of image around it.

Note that, we also have max pooling layers in CNNs which despite being useful for increasing receptive field and translation invariance discards the 'where' information. But, semantic segmentation requires the exact alignment of segments and thus, needs the 'where' information to be preserved across the network. Researchers threw away fully connected layers and tackled pooling layer problem with mainly two different architectures. First one is based on encoder-decoder system. In this design, we have an encoder which gradually downsample the space with pooling layers and a decoder to gradually recover the 'where' information. There are also shortcut connections from encoder to decoder to help decoder recover the object details better. Other architecture used dilated/atrous convolutions and do away with pooling layers.

In 2014 Long et al. from Berkeley came up with Fully Convolutional Networks (FCN) [1] and popularized CNN architectures for dense predictions without any fully connected layers. This allowed variable sized images and also worked faster then the past patch classification networks. Almost all the subsequent state of the art methods on semantic segmentation are inspired from this paradigm

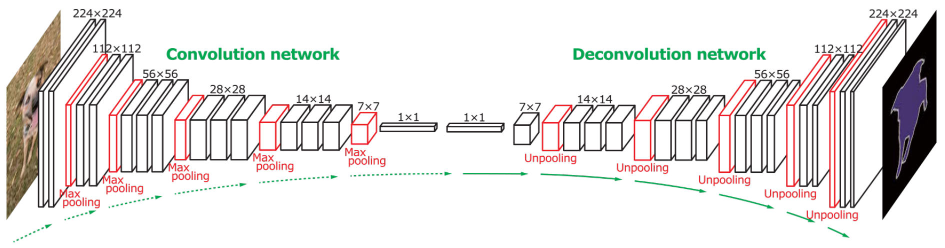

Figure 1: FCN Network. Image source

Then came SegNet[3] in 2015 which solved the issue of course maps from FCN by introducing connections between encoder and decoder parts of the network. Although, this network is not used any more, this lead to more popular U-Net architecture, which we are implementing for our review

The U-Net model architecture as the name suggests has U shaped encoder and decoder design. Encoder is made up of blocks that downscale an image into narrower feature layers while the decoder mirrors those blocks in the opposite direction, upscaling outputs to the original image size and ultimately predicting a label for each pixel. Skip connections cut across the U to improve performance.

Researchers parallelly came up with Dilated convolutional layers and pyramidal structures(atrous convolution in DeepLab[4]) which increase the receptive field exponentially without decreasing spatial dimensions. Today, all the state of the networks like Mask R-CNN (by FAIR) and DeepLab (by Google) combine many of the structures covered above to produce high quality segmentation masks on many popular datasets.

3 Training

3.1 Data

At the heart of this task, we have a large dataset of City scapes labelled by human annotators. Cityscapes is a bench marked large-scale dataset to train and test approaches for pixel-level semantic labeling. It contains a large, diverse set of stereo video sequences recorded in streets from 50 different cities. 5000 of these images have high quality pixel-level annotations;20 000 additional images have coarse annotations to enable methods that leverage large volumes of weakly-labeled data [5]. In order to have more robustness in our network, we augmented the dataset with affine transforms like translation, homography, rotation etc. We also did random centre crop to alleviate

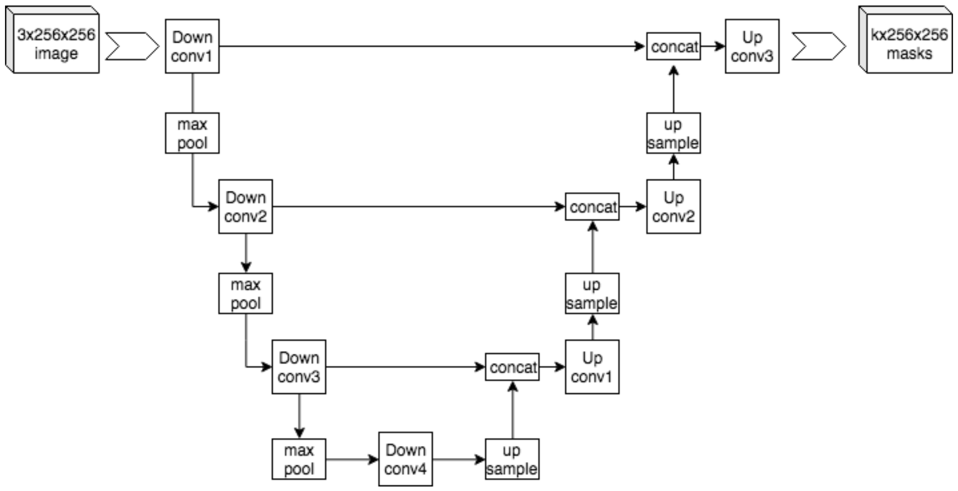

Figure 2: U-Net. Image source

border effects. We are expecting, these transforms will make the network robust and invariant to size, translation, rotation and homography etc, which inturn improved the performance of the network.

3.2 Network Design

Initially we implemented FCN and then modified it to U-Net with encoder-decoder architecture with skip connections between encoder and decoder parts of the network. Encoder compresses the image to lower dimensions, retaining most important features and we use decoder to segment the image using these features. We then explored transfer learning technique by using pretrained VGG [6] and then developed our network inspired from inception network, named it as Megatron.

3.2.1 FCN

We have 5 convolution layers and 5 deconvolution layers followed by one convolution layer and softmax for classification. Between every pair of convolution layers, we have a batch normalisation layer

conv → batch norm → conv2 → batch norm → conv3 → batch norm → conv4 → batch norm → conv5 → batch norm → de-conv → batch norm → de-conv2 → batch norm → de-conv3 → batch norm → de-conv3 → batch norm → de-conv4 → batch norm → conv → softmax layer

Table 1: FCN architecture

Layer	**Type**	**in-c**	**out-c**	**kernel-size**	**padding**	**stride**	**Non linearity**
1	conv	3	32	3	1	2	Relu
2	conv	32	64	3	1	2	Relu
3	conv	64	128	3	1	2	Relu
4	conv	128	256	3	1	2	Relu
5	conv	256	512	3	1	2	Relu
6	deconv	512	512	3	1	2	Relu
7	deconv	512	256	3	1	2	Relu
8	deconv	256	128	3	1	2	Relu
9	deconv	128	64	3	1	2	Relu
10	deconv	64	32	3	1	2	Relu
11	conv	32	34	1	0	1	Relu

3.2.2 UNet

This network is similar to FCN. In addition to having encoder and decoder, U-net provides skip connection from output of each convolution layer in encoder to input of de-convolution layer symmetrically.

Architecture:
Encoder:
conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →pool2 →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →pool2 →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →pool2 →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →pool2 →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm
Decoder:
Deconv3 →ReLU →batch norm →Concat →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →Deconv3 →ReLU →batch norm →Concat →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →Deconv3 →ReLU →batch norm →Concat →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →Deconv3 →ReLU →batch norm →Concat →conv3 →ReLU →batch norm →conv3 →ReLU →batch norm →Classifier

Table 2: UNet architecture

Layer	type	in-c	out-c	kernel-size	padding	stride	Non linearity
1	conv	3	64	3	1	1	ReLu
2	conv	64	64	3	1	1	ReLu
3	pool	64	64	2	0	2	
4	conv	64	128	3	1	1	ReLu
5	conv	128	128	3	1	1	ReLu
6	pool	128	128	2	0	2	
7	conv	128	256	3	1	1	ReLu
8	conv	256	256	3	1	1	ReLu
9	pool	256	256	2	0	2	
10	conv	256	512	3	1	1	ReLu
11	conv	512	512	3	1	1	ReLu
12	pool	512	512	2	0	2	
13	conv	512	1024	3	1	1	ReLu
14	conv	1024	1024	3	1	1	ReLu
15	pool	1024	1024	2	0	2	
16	deconv	1024	512	3	1	2	ReLu
17	conv	1024	512	3	1	1	ReLu
18	conv	512	512	3	1	1	ReLu
19	deconv	512	256	3	1	2	ReLu
20	conv	512	256	3	1	1	ReLu
21	conv	256	512	3	1	1	ReLu
22	deconv	256	128	3	1	2	ReLu
23	conv	256	128	3	1	1	ReLu
24	conv	128	128	3	1	1	ReLu
25	deconv	128	64	3	1	2	ReLu
26	conv	128	64	3	1	1	ReLu
27	conv	64	64	3	1	1	ReLu
28	conv	64	34	1	0	1	ReLu

3.2.3 Our Architecture - Megatron

Inspired from Inception, we designed our own architecture using components as shown in the figure below. We basically used different size of kernels of different convolutional layers and then concatenated the outputs of the these conv layers.

We have 5 encoder modules followed by 5 decoder modules in Megatron. This is followed by a classifer layer.

Encoder Module:
Intermediate Layer = (Conv1 →AvgPool2) + (Conv1 →Conv3 →MaxPoo2l) + (Conv1 →Conv5 →MaxPool2) + (MaxPool2 →Conv1)

Out Encoder = Intermediate Layer →Batch Norm →ReLU

Decoder Module :
Intermediate Layer = (Conv1 →Deconv3) + (Conv1 →Conv3 →Deconv3) + (Conv1 →Conv5 →Deconv3) + (Deconv3 →Conv1)

Out Decoder = Intermediate Layer →Batch Norm →ReLU

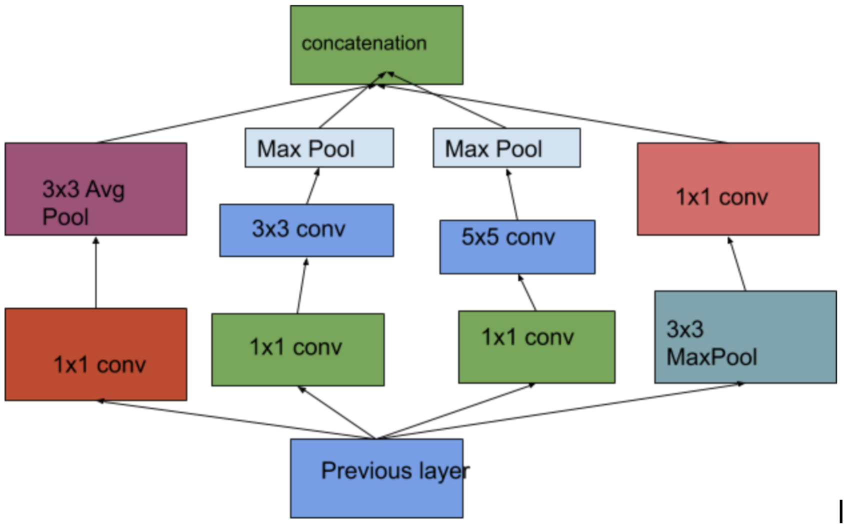

Figure 3: Encoder-Module of our architecture- Megatron

We initialised network with Xavier initialisation to have output variance same as input variance and maintain the variance of activations and back-propagate gradients all the way up or down the layers of a network. We also performed batch normalisation before every convolution layer which is shown to improve the speed of the network by normalizing the input layer through adjusting and scaling the activations. Batch normalisation seem to reduce the amount by what the hidden unit values shift around (covariance shift) and helped in improving the speed

For non linearity, we used Relu activation function and used softmax in the end for dense classification (included in cross-entropy criterion of PyTorch, no need to implement separately)

For loss optimisation problem we used gradient descent with Adam optimiser from Pytorch library on L^2 regularised loss. By training this way, we observed that some classes are neglected in model training because of imbalance in distribution of classes. To solve this issue, we implemented weighted loss (weighting infrequent classes more) and dice coefficient loss. We used pixel accuracy and more robust Intersection-Over-Union (IOU) as our main evaluation metrics.

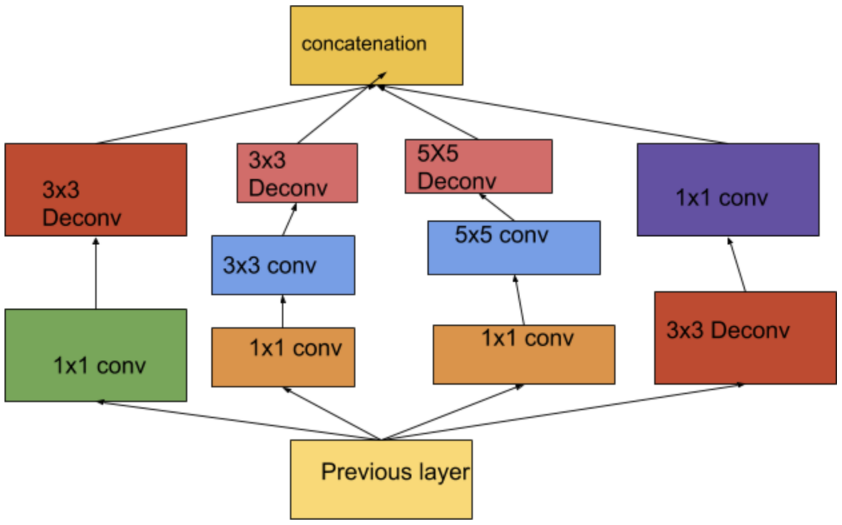

Figure 4: Decoder-Module of our architecture- Megatron

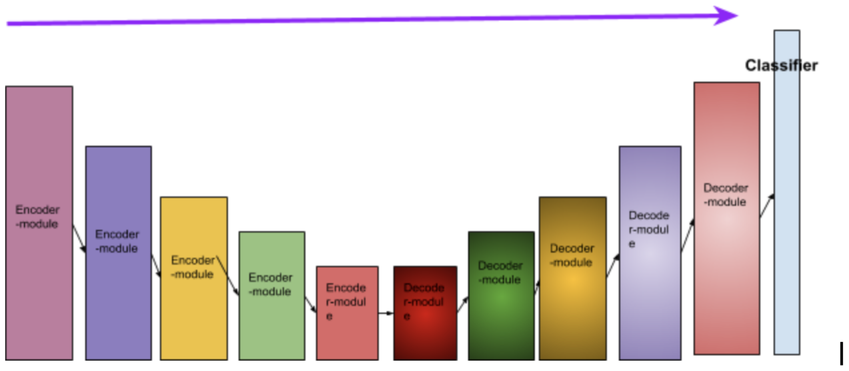

Figure 5: Our architecture- Megatron

4 Results

Our results are as follows

Table 3: Results summary

Model	Acc	Avg Iou	Avg Iou 11	Avg Iou 20	Avg Iou 24	Avg Iou 26	Avg Iou 33
FCN	0.8602	0.7259	0.6633	0.2510	0.4014	0.6858	0.3555
FCN with wei. loss	0.6872	0.5488	0.5248	0.2851	0.3307	0.4991	0.3505
Unet	0.8749	0.7552	0.7261	0.4220	0.4637	0.6813	0.4208
TL VGG	0.7328	0.5547	0.4876	0.1125	0.1476	0.4195	0.1245
Megatron	0.8876	0.7705	0.7374	0.3684	0.5029	0.7571	0.4814

FCN with unweighted loss

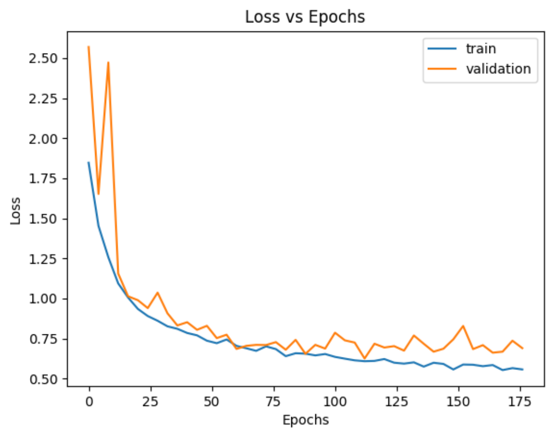

Figure 6: FCN loss over training time

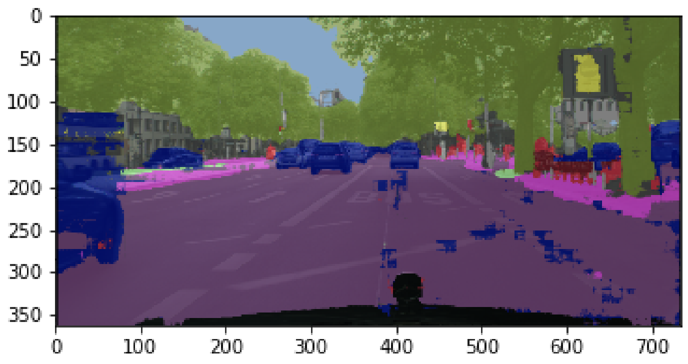

Figure 7: dense prediction using FCN

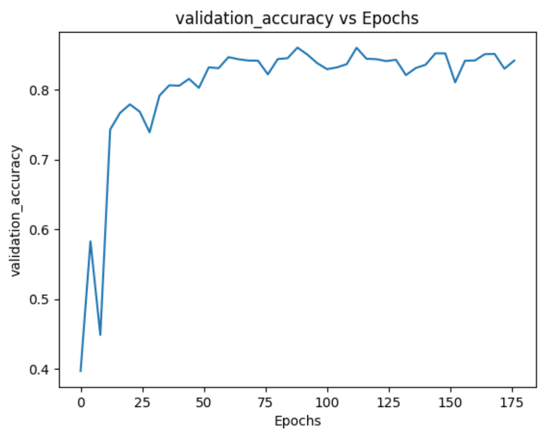

Figure 8: validation accuracy over training using FCN

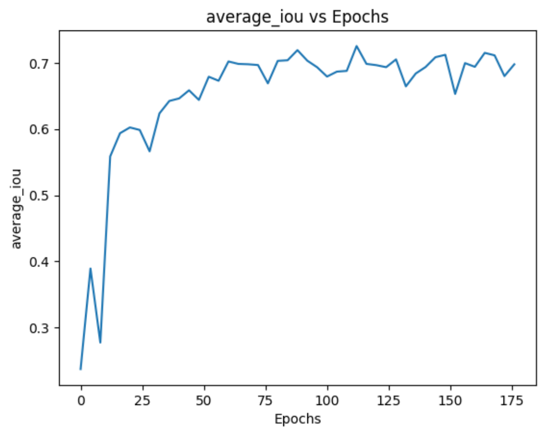

Figure 9: avg_iou over training using FCN

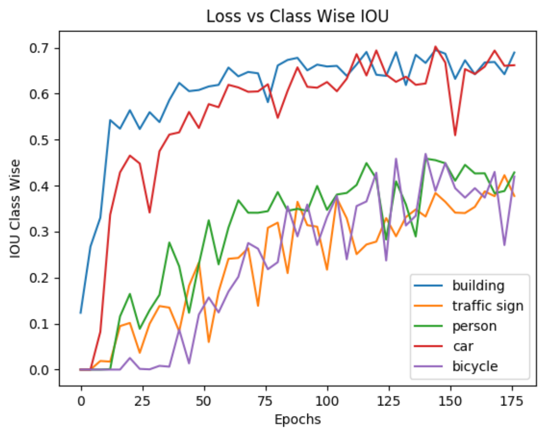

Figure 10: class wise iou over training using FCN

FCN with weighted loss

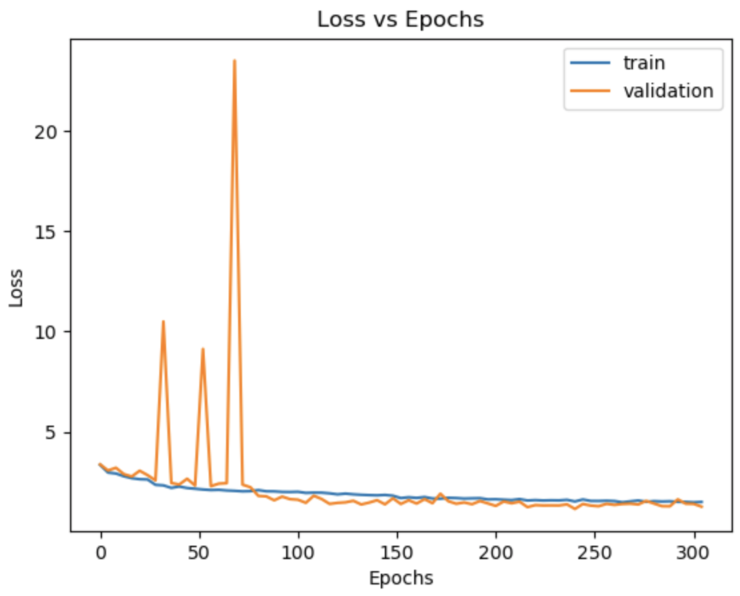

Figure 11: FCN with weighted loss loss over training time

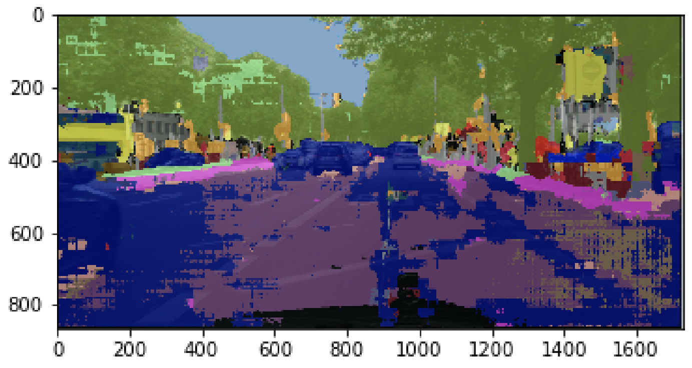

Figure 12: dense prediction using FCN with weighted loss

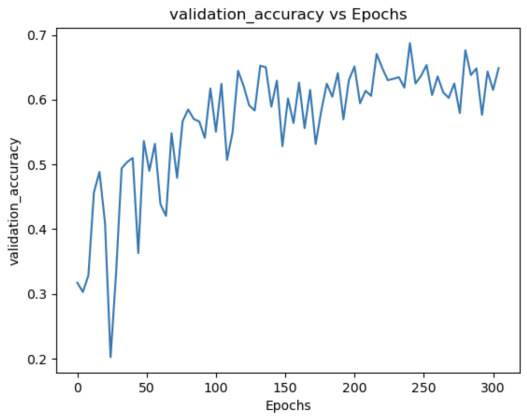

Figure 13: validation accuracy over training using FCN with weighted loss

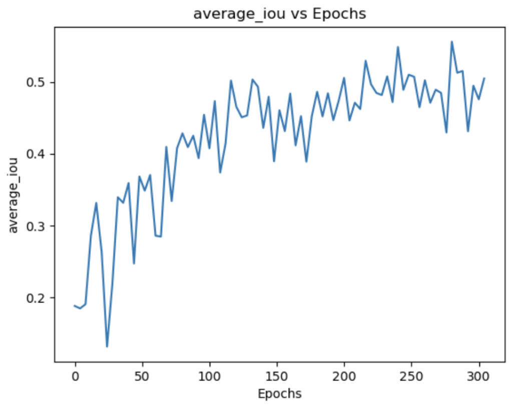

Figure 14: avg_iou over training using FCN with weighted loss

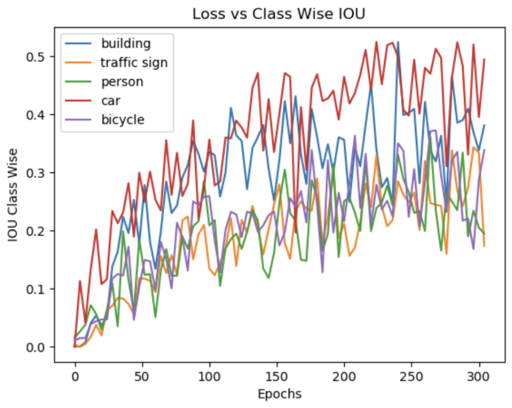

Figure 15: class wise iou over training using FCN with weighted loss

Unet with unweighted loss

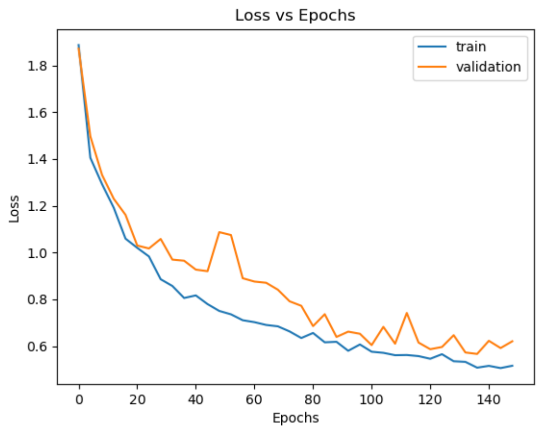

Figure 16: Unet loss over training time

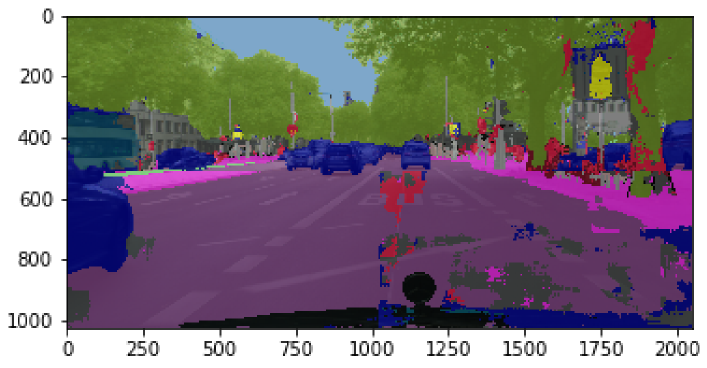

Figure 17: dense prediction using Unet

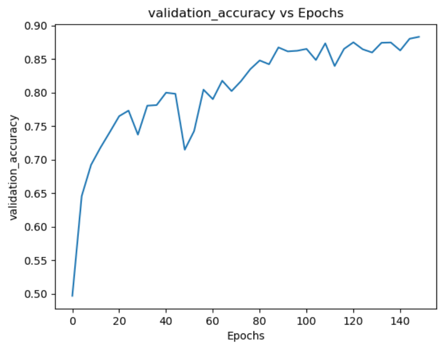

Figure 18: validation accuracy over training using Unet

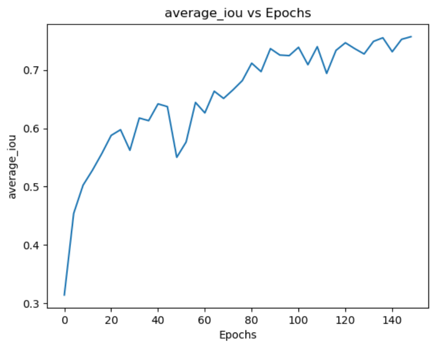

Figure 19: avg_iou over training using Unet

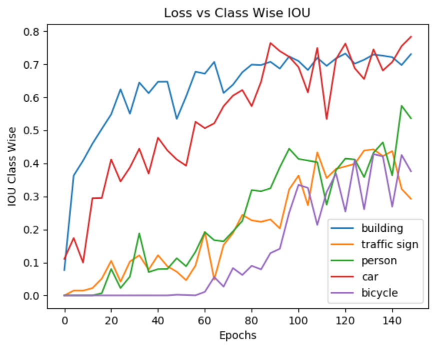

Figure 20: class wise iou over training using Unet

Transfer learning using pretrained VGG with unweighted loss

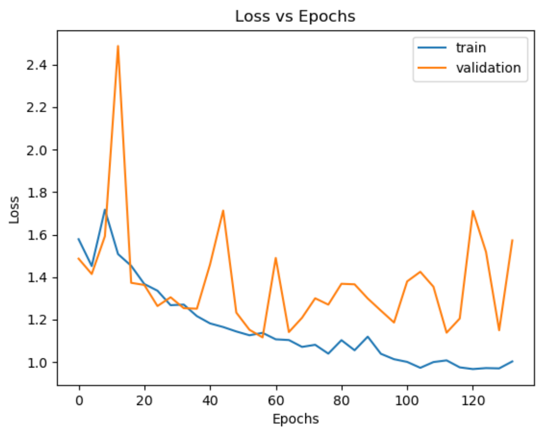

Figure 21: Transfer learning using Pretrained VGG loss over training time

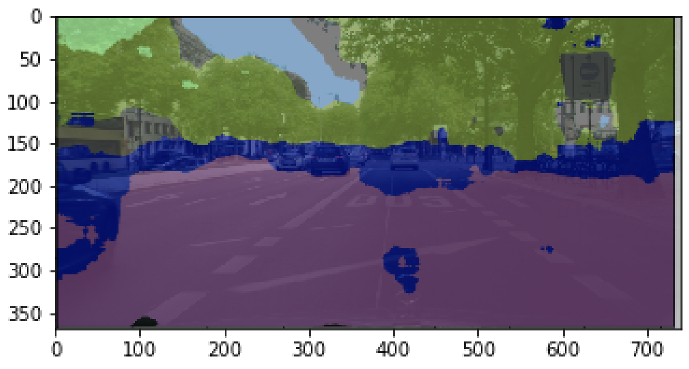

Figure 22: dense prediction using Transfer learning using Pretrained VGG

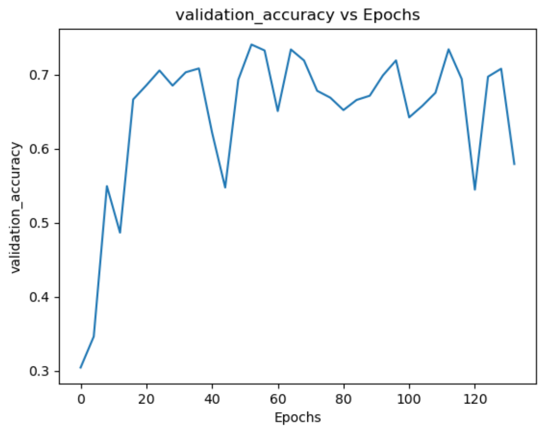

Figure 23: validation accuracy over training using Transfer learning with Pretrained VGG

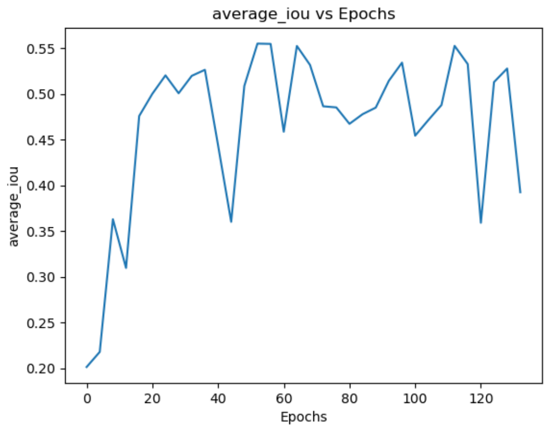

Figure 24: avg_iou over training using Transfer learning with Pretrained VGG

Results with Megatron(our own architecture)

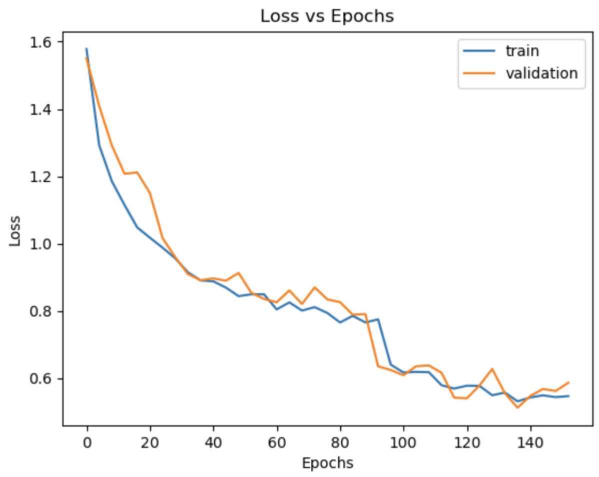

Figure 25: Loss vs Epoch using Megatron

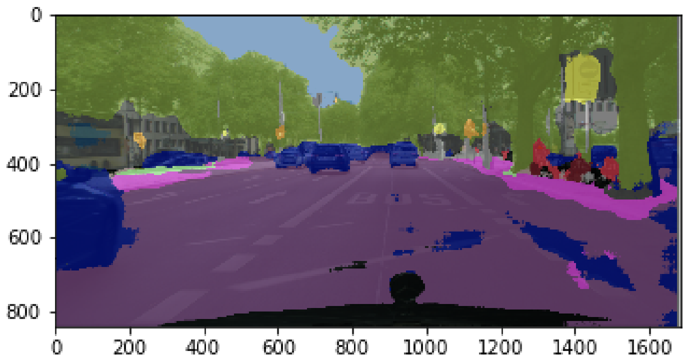

Figure 26: Prediction using Megatron

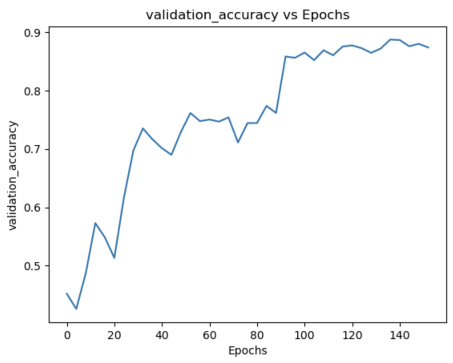

Figure 27: Validation Accuracy using Megatron

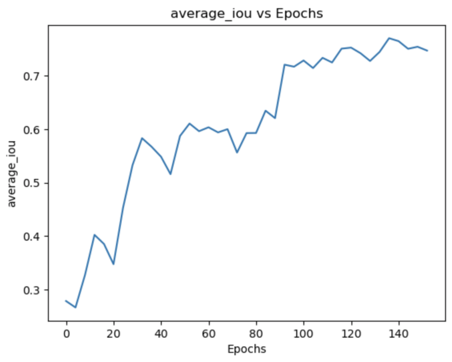

Figure 28: avg iou using Megatron

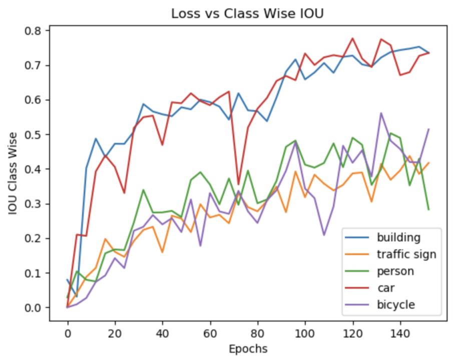

Figure 29: class ious using Megatron

5 Discussion

5.1 Data augmentation

Inorder to create a more robust network, We decided to augment the dataset by various affine transfoms(translation, rotation, homography etc). We tried cropping images to random size, but then chose 256x512 sizes to use a larger batchsize and for faster training process. We were undecided whether to use random crops or resize for getting the smaller images in this case. We performed an ablation study where we trained two models and one was fed random cropped images while the other was fed downsampled images. Model trained on cropped images performed much better than the model trained on resized images (model's output seems very coarse, couldn't learn the finer features)

Based on the current dataset, we are expecting random horizontal flips would be more beneficial to the model than vertical flips as images are taken with its top towards sky. We found that vertical flips were infact detrimental to the model validation performance and were excluded from data augmentation process. It makes sense because it would be unnecessary for the model to learn on vertically flipped images whose features are not very useful on validation data. We also decided to add a random rotation of upto 5 degrees to the images.

5.2 FCN Unweighted

The first thing we did was to play around with the base FCN architecture with different number of layers and various hyper parameters(learning rate, batch norm and relu positions, regularisation loss etc.). In the end, we chose the best model out of this and this base FCN model gave the reported performance. We are using the same hyper parameters throughout our future networks

5.3 FCN Weighted

For handling class imbalance, we initially used Dice-loss but observed that average IOU increases by less than 1% while the pixel accuracy drop is larger. To avoid this, we tried implementing weighted cross-entropy loss and weighted dice loss. The we formulated our weights to each class inversely proportional to its class size i.e, $\frac{N_{max}}{n_i}$. Training with this weighted loss seems to be difficult as can be seen by the loss plot which is very noisy. The model tries to learn minority classes as well, which can be inferred from the segmented image on the test set, but the training is too slow. We tried both stochastic weights, where we compute the class weights from the given batch, as well as from the complete data. We feel that given more time for this model to train, this should beat the baseline FCN. Or maybe a larger crop size with larger batch size should decrease training time since that would decrease input data variance, but that would require more GPU compute.

5.4 UNet

We implemented UNet architecture as mentioned in the paper. Since it is larger model having a lot of weights, we had to use a smaller batchsize as compared to baseFCN model in order to avoid CUDA running out of memory. We observe that UNet beats FCN model in both Pixel accuracy and Avg. IOU by 1.5% and 3%. We expected UNet to have better performance than basic FCN because of the skip connections which circumvent the lower dimensional latent space bottleneck. The skip connected features would help the model learn coarse features while other feature maps learn fine grained features. These skip connections also allow the gradients to backpropagate very efficiently and thus, the learning is somewhat faster. Our results reinforced the same. A rather interesting observation from the class wise IOU plots is that the classes with high frequency are learnt quickly and as the performance on them saturate, the performance on low frequency classes start to pick up and gets better.

5.5 Pretrained VGG

We chose the pretrained VGG [6] on ImageNet dataset and used it as encoder for our segmentation task and froze its network weights. We made our own decoder, not an exact replica of the encoder which otherwise would lead to very large model and cannot be stored in CUDA memory during training. We are expecting that this architecture couldn't train properly or learn the underlying model because the data on which VGG was trained has a very different distribution from the cityscape dataset and it's parent task(recognition) is considerably different from semantic segmentation. Recognition involves just identify the object in image where as in semantic segmentation, along with identifying the objects, we also need to identify its boundaries. This lead to having some of the classes were not recognized by the deep CNN network which can be seen from the very small IOU values in the table. We also feel that adding skip connections to the model would help improve the model and a symmetric encoder-decoder structure would be helpful.

5.6 Megatron

We built our own network Megatron, inspired from Inception network. We expected the different kernel sizes in our network would help the model to learn features at different scales. As expected, we observed that this model beat all our previously explored models even without weighting the loss. The model beats baseline by 2.7% Pixel accuracy and 5.5% Average IOU.

6 Author's contributions

Everyone contributed equally towards discussion, debugging, training and monitoring models.

6.1 Astuti Sharma

UNet, Transfer Learning, Exploring Architectures

6.2 Divyanshu Mund

Exploring new architectures, hyperparameters tuning, loss for imbalanced classes case, FCN

6.3 Kunal Jain

Dataloader, transformations, loss computation, graph plotting, checkpoints, FCN

6.4 Ria Aggarwal

Transfer learning, Unet, designing/manipulating architectures, data transformations

6.5 Sai Akhil Suggu

Report, inferences, UNet, literature review

References

[1] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014. arXiv: `1411.4038 [cs.CV]`.