OBJECT SEGMENTATION

Bhuvaneshwaran Ponnusamy Ilanthirayan
Department of Computer Science
Universität des Saarlandes
Saarbrücken, Saarland, DE 66123
bhpo00001@uni-saarland.de

Chistopher Jung
Department of Computer Science
Universität des Saarlandes
Saarbrücken, Saarland, DE 66123
chju00001@uni-saarland.de

March 31, 2021

ABSTRACT

Object Segmentation is an important task in Computer Vision. It is applied in various tasks such as Medical Imaging, Pedestrian detection and numerous others. Generally, object segmentation tasks involves pixel level classification of an image and more often the classification in real world scenario would be a multi class problem. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyse[1]. We experimented with Fully Convolutional Network with ResNet-101 baseline, followed by Recurrent Residual Convolution Neural Network based on U-Net (R2U-Net)[2] and improved the performance by using Atrous Spatial Pyramid Pooling Recurrent Residual Convolutional Neural Network with Attention Gates (ASPP R2U-Net with Attention Gates) for the segmentation task and have compared their performance against the R2U-Net on the cityscapes dataset [3].

1 Introduction

Our project is based on Object Segmentation more often referred to as Semantic Segmentation or Image Segmentation.The project is split into three tasks. The first task introduces us to the task of segmentation applied on PASCAL VOC 2012 dataset[4]. In the second one, we are provided with a paper and a dataset called the cityscapes dataset[3]. In this, we are required to implement the same architecture as mentioned in the paper [2] and evaluate it. Followed by the third, where we are allowed to choose any architecture of our choice that hopefully improves the performance that is achieved in the second task. Finally, the models are evaluated and the results are compared. This report compares the second and third task in detail. All the implementation are available in our Github repository - `https://git.io/JYzau`.

2 Task description and data construction

We are provided with two different datasets for Task 1 and Task 2.1 and 2.2: Pascal VOC 2012 dataset[4] for Task 1, Cityscapes [3] dataset for Task 2.1 and 2.2. The core requirement of the tasks are to perform segmentation on these dataset by training a Neural Network with training data and evaluate on the validation data. The comparison of the models from Task 2.1 and Task 2.2 are based on different metrics as mentioned in 4.3.

2.1 Task 2: Recurrent Residual Convolution Neural Network

Introduction: We are given with the Cityscapes dataset [3], a reference paper [Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation [2] and the requirement is to implement the architecture from the paper for our segmentation task on the cityscapes dataset. We can decide the hyperparameters but the code implemented should be reflecting the architecture.

Description of dataset and dataset loader: The cityscapes dataset contains data from 50 different cities with pixel quality annotations of 19 different classes falling under different groups such as flat, human, vehicle, construction,

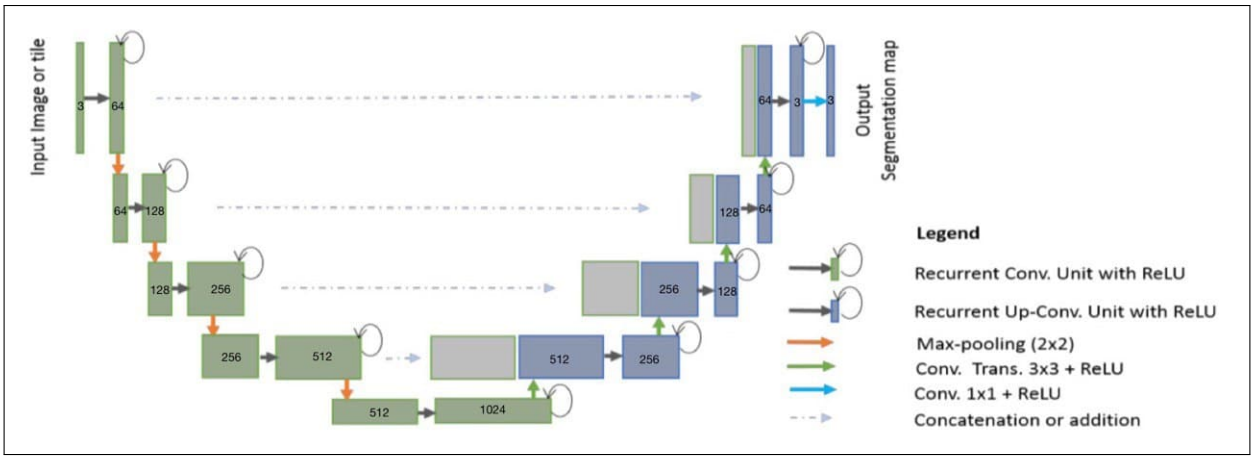

Figure 1: R2U-Net Architecture

object, nature, sky, void [3]. We use only the 'leftImg8bit_trainvaltest' and 'gtFine_trainvaltest' packages. We reduce the image size (512 x 256) from the higher quality (2048 x 1028) to reduce the computation overhead but without compromising the details in the image. The image when reduced should contain enough details else the segmentation model will not have enough information to compute the segmentation boundary and classify the classes accordingly. We chose to include classes that are necessary in evaluation and cannot be ignored as indicated in their GitHub Repository.

2.2 Task 3: Challenge Task

Introduction: We have seen that the R2U-Net performs decently well on the cityscapes dataset. But in some cases, it does not represent the classes of the data well. In this task, we try to improvise the performance of the R2U-Net by introducing some modifications to the architecture. The modifications are expected to be performed in terms of change in the pipeline of the architecture, or data representation, or a different architecture.

Description of dataset and dataset loader: As we are required to improve on the architecture from Section:2.1, We use the same cityscapes dataset and dataset loader as we saw in Section 2.1.

3 Solution Architecture and Observation

3.1 Task 2: Recurrent Residual Convolution Neural Network

Solution Architecture: The reference paper proposes a modified version of the famous U-Net architecture called Recurrent residual convolutional neural network (R2U-Net). The main idea of the paper is to use a U-Net style encoding and decoding with recurrent residual convolutional layers. In the U-Net style, the encoding unit will reduce the images from a high scale representation (larger number of maps) to low scale representation (lower dimensionality) and then the decoding unit will upsample the images back to their original representation [2]. The key idea is to introduce the idea of residual units to the encoding and decoding units recurrently such that features are accumulated and ensures better feature representation. Figure:1 represents the architecture and how the image is reduced and scaled back.

Observation: The idea of using the residual units helps us to train deeper models by accumulating the features in the deeper models. One problem with the deeper models are the vanishing gradient problem that were prominent during back-propagating the weights, this is reduced by the implementation of the Rectified Linear Unit (ReLU) at the end. The output from the model after training is performing well and the results can be seen in the Figure: 2 in the order - input, ground truth and prediction of our model. The architecture can also seen performing not so well in some cases and can be seen in the Figure: 3 ordered in the same manner as Figure:2. We can see that the model fails to recognise some of the classes in the dataset such as the pole group and cannot establish the segmentation boundary well for some classes such as the guard rail.These could be mainly because the data might have underrepresented while it is encoded and since two classes have neighbouring pixels the models fails to distinguish between them. We decided to go deep to five layers as indicated in the architecture diagram of the reference paper. As indicated in the paper, we could also observe that the parameters scaled up to 39 million.

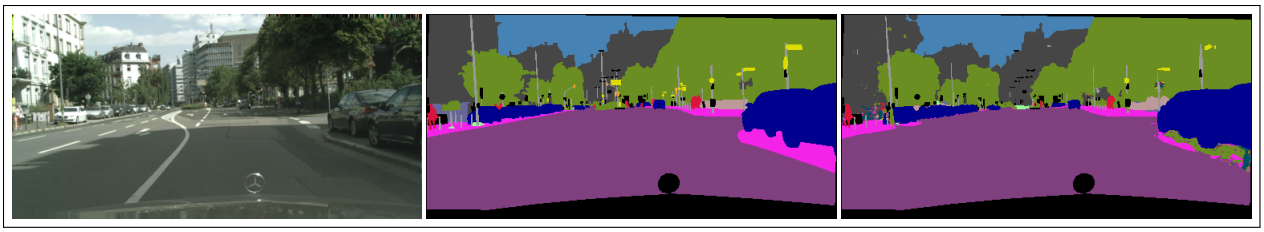

Figure 2: Evaluation of R2U-Net on 'val' split after training

Figure 3: Failed results

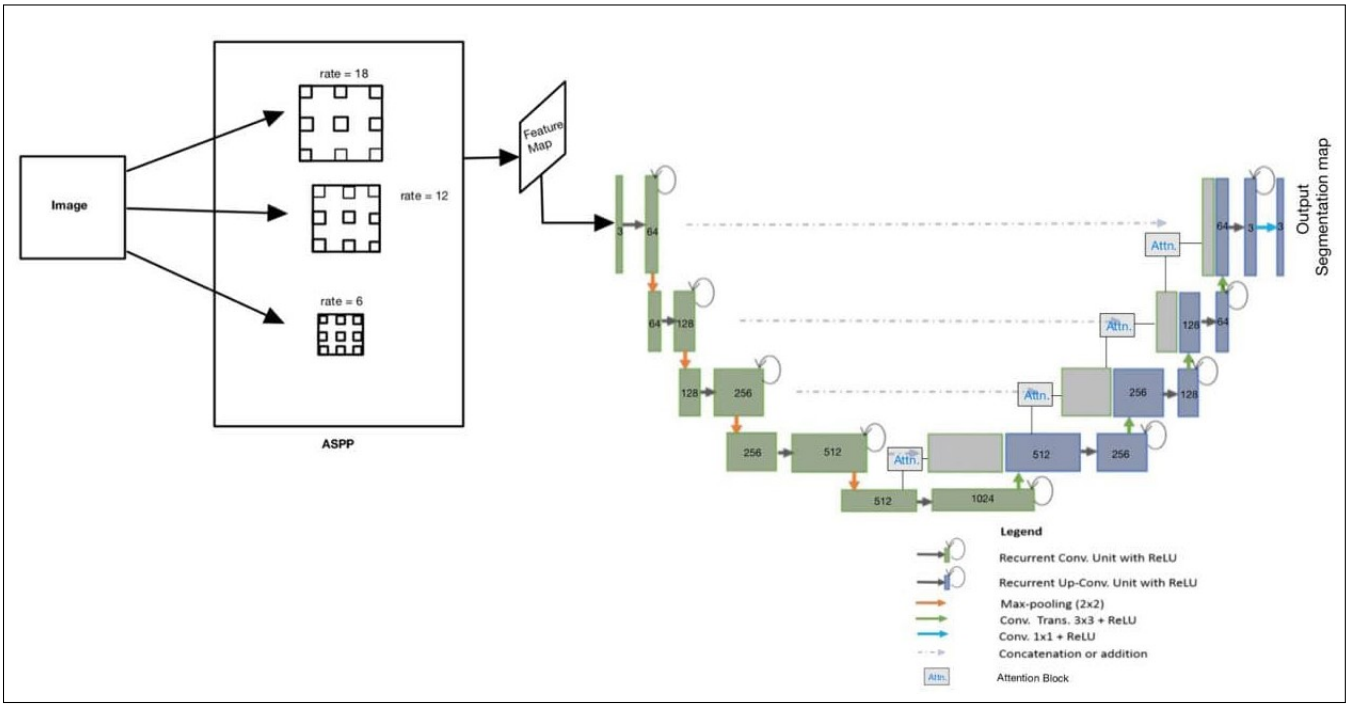

Figure 4: ASPP R2U-Net with attention gates Architecture

3.2 Task 3: Challenge Task

Solution Architecture: As we thought that the R2U-Net has an underrepresented data while encoding the data throught the encoding unit of the R2U-Net, we decided to introduce a model that can represent the data in a different manner while preserving the features. For this, we decided to make use of the Atrous Spatial Pyramid Pooling (ASSP) from the DeepLab model [5]. Since, the module samples the input at different rates it could represent the features well and feeding it though R2U-Net would classify the features well. We also decided to combine the R2U-Net with attention gates initially by referring Lecture 10 slides and later [6] as an idea indicated by LeeJunHyun in their GitHub repository [7] that possibly seem to improve the results slight than the R2U-Net itself. The introduction of attention gates might help to improve the understanding of interactions between the features and in turn help to distinguish between them. The architecture can be seen in the Figure: 4.

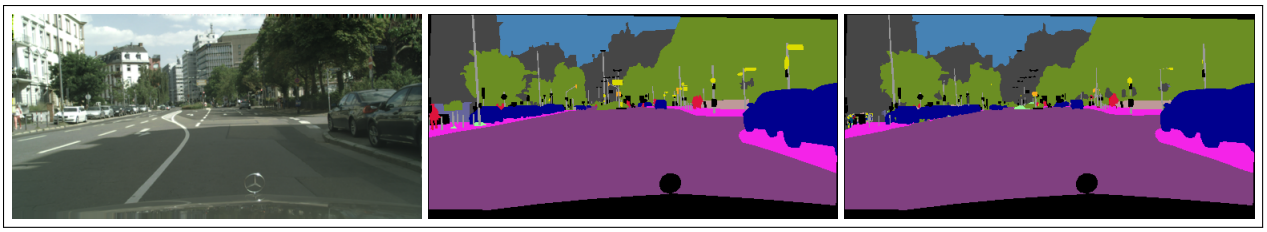

Figure 5: Result of ASPP R2U-Net Model

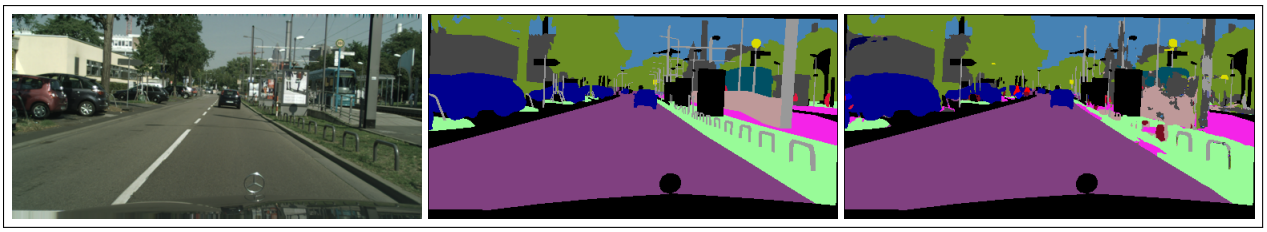

Figure 6: Failed Results

Observation and Comparison: The idea of using the Atrous Spatial Pyramid Pooling and attention gates with R2U-Net doesn't seem to drastically improvise the results while visualising it, but from analysing the model using the same metrics as we did for R2U-Net we could notice that the proposed model performs a little better than using just the R2U-Net as we can see in 4.3.Figure:3.2 shows how well the model performs with order of images as input, ground truth and predicted output.Figure:6 shows that the model fail to represent certain classes. We could also note that in both the R2U-Net and ASPP R2U-Net with Attention Gates, we have a slight coarse representation of classes with some classes such as poles have an interleaved segmentation. This might be due to the fact that both the models are encoding the images to a very larger channel and then decoding it. But when evaluated on trained data itself, we can see that the ASPP could represent the features well in different scales and fit them more properly, but the output of the R2U-Net could not be represented the pixels uniformly in a high quality. We can see these differences in the Figure: 3.2 given in the order of ground truth, R2U-Net output and ASPP R2U-Net with Attention Gates Model output where the boundaries are very well classified. In Figure:8 given in the same order as 3.2, we can notice that the fine structures such as the light pole is classified correctly without much interleaved pixels. For some coarse structures to be represented fine, one can use the PointRend method [8] which could possibly improvise the result while reducing the computation load unlike ASPP. But for our purpose of scaling invariant representation and also representing fine structures, we decided to stick on and experiment with ASPP even though it is a bit more computationally expensive. In Addition to ASPP, Attention gates also seem to have helped in representing input well through the decoding process. Together, both have given a slightly improvised result. Further details are found in the section: 4.3

4 Setup, Implementation and Results

4.1 Setup

To demonstrate the performance of the R2U-Net and ASPP R2U-Net with Attention Gates models, we have implemented in the PyTorch framework in a provided GPU Cluster that contains eight Tesla P100-SXM2 GPU having a capacity of

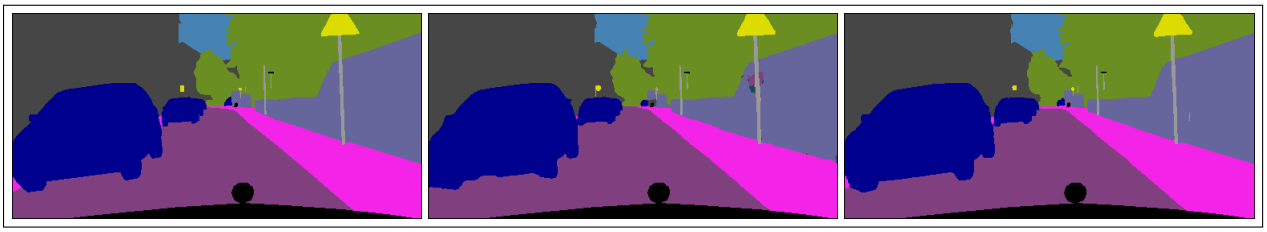

Figure 7: Comparison of segmentation boundaries of R2U-Net Model and ASSP R2U-Net with Attention Gates Model

Figure 8: Comparison of fine structures of R2U-Net Model and ASSP R2U-Net with Attention Gates Model

16 GB each and runs on Intel Xeon CPUs with 503 GB RAM. The local disk space is always kept free of minimum 2 GB incase we need to store the model weights in the middle of training.

4.2 Implementation:

The implementation of the model is done mainly using the PyTorch library. All the implementation code can be found in our Github repository - `https://git.io/JYzau`. The Task 2.1 can be run using 'Task 2.ipynb' file and Task 2.2 can be run using 'Task 3.ipynb' file.

4.2.1 Preprocessing:

The cityscapes dataset shoudl be downloaded to '/dataset' folder. We downloaded the packages - 'gtFine_trainvaltest' and 'leftImg8bit_trainvaltest' for out training and validation of our model. The 'utils.py' file contains a function called *'resizeAllImages()'* that resizes all the images onto a size of 512 x 256 using the *.resize()* method of PyTorch. We decided to reduce the size from the original 1024 x 2048 to reduce the computation load. Our chosen size, strikes a good balance between computation load and also the feature representation. Once the image size is reduced, they are stored in the target path given in the notebooks as '/dataset/cityscapes/medium/'. The 'cityscapes.py' contains the cityscapes class, which returns the dataset. The input images and target images can be obtained to process in the model using the pytorch's inbuilt *DataLoader*.

4.2.2 Task 2.1:Recurrent Residual Convolution Neural Network

The implementation of the architecture contains in the 'r2uModel.py' file. The class *R2U_Net* provides us with an instance of the R2U-Net architecture model which can be used for training. The file also contains all the blocks as separate classes that constitute the R2U-Net [2]:

***down_block*:** This class denotes the encoding unit that encodes the image from its original size down to 'n channels'.

***recurrent_block*:** This class denotes the Recurrent Convolutional Layer(RCL) that uses the *recurrent_layer* to form the recurrent block.

***up_block*:** This class denotes the decoding unit that decodes the image back to its original size.

***default_layer*:** This class denotes the CNN type layers i.e. Convolutional followed by Maxpool and ReLU. This class is established for ease of re-usability and reduce redundant code.

***recurrent_layer*:** This class denotes the *default_layer* but in a useful manner for the *recurrent_block*. Even this class is established for re-usability and reducing redundant code.

4.2.3 Task 2.2: Challenge Task - ASPP Recurrent Residual Convolution Neural Network with Attention Gates

The implementation of the architecture contains in the 'r2uOptimModel.py' file. The class *R2U_Net_Optimized* provides us with an instance of the ASPP R2U-Net with Attention Gates architecture model which can be used for training. In addition to the classes seen above in 4.2.2. The additional layers includes:

***ASPP*:** This class denotes the 'Atrous Spatial Pyramid Pooling' [5] method and gives us with an instance that does the ASPP for a given input image.

***attention_block*:** This class denotes the attention gates [6][7] that is used on while decoding using the *up_block*.

4.2.4 Training

The training of the model is done using the *train()* method in the 'trainLoop.py' file. The loss function used is Mean Squared Loss function with the Adam Optimiser. We chose this arbitarily to start with the training and it seemed to perform good.

4.2.5 Evaluation and Visualisation

The evaluation of the models trained are done using the *evaluate()* function in the the 'metrics.py' file. The *evaluate()* function returns the *EvalResult* object containing the Dice Coefficient(DC), Jaccard Similarity (JS) and a Confusion matrix with which rest of the metrics as indicated in Section:4.3 are calculated. The evaluation results are displayed using the *displayEval()* function inside the same file. The visualisation of the images is achieved using the *preview()* method inside the 'utils.py' file.

4.2.6 Saved Models

The repository also contains the trained models under the *'models/'* folder. Where the 'r2u.model' contains the weights of the model trained for Task 2.1 and 'r2u_optim.model' contains the weights of the model trained for Task 2.2. These models are trained for 100 epochs.

4.3 Qualitative Analysis

4.3.1 Task 2.1: Recurrent Residual Convolution Neural Network

As given in the paper under the "Quantitative Analysis Approaches" [2], we are required to calculate five different metrics for the architecture indicated by the formulas in that section. Those metrics are Accuracy (AC), Sensitivity (SE), Specificity (SP), Dice coefficient (DC), and Jaccard similarity (JS).

Accuracy: Accuracy or Pixel Accuracy is the pixel wise correctness of the segmentation result. Accuracy checks for the pixel to pixel matching of the ground truth with the predicted output. The formula for calculating accuracy can be given by,

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Sensitivity: Sensitivity or True positive rate measures how much true positive that are identified against the total positives identified. This helps us to measure how far the model is able to identify the true classes. It is given by,

$$SE = \frac{TP}{TP + FN} \tag{2}$$

Specificity: Specificity or True negative rate measures how much true negatives are identified against the total negatives identified. This helps us to measure how far the model is able to detect the negative classes. It is given by,

$$SP = \frac{TN}{TN + FP} \tag{3}$$

Dice Coefficient: The Dice Coefficient measures the similarity between two images by comparing the overlapping regions. It helps us to identify how similar is the output of the model to the ground truth. It is given by,

$$DC = 2\frac{|GT \cap SR|}{|GT| + |SR|} \tag{4}$$

Jaccard Similarity: The Jaccard Similarity is similar to Dice Coefficient, but Jaccard Similarity can be considered as a distance metric. It also compares the similarity with the output but it can help us see how far is the output from the ground truth. It is given by,

	Name	SE	SP	ACC	F1	Dice	Jaccard
0	road	0.9735239336541928	0.9898087778919389	0.9836815872978714	0.978210079952225	0.9424627075195312	0.9246572265625
1	sidewalk	0.8453858577261198	0.9891801533642876	0.9813996717282771	0.8310371425791184	0.7154949547532565	0.6199721219783316
2	building	0.9291061355512548	0.9714547165993213	0.9621679833486892	0.9150461512044764	0.8551626586914063	0.7779834594726562
3	wall	0.3070482667242882	0.9971887794660651	0.9921262442916838	0.3639152337263952	0.12851695237655986	0.09608122018667367
4	fence	0.3712541950046034	0.9968206475418102	0.9916842033829908	0.4230124046447321	0.10870869439623133	0.0774016283784318
5	pole	0.5555437667033299	0.9962003239695858	0.9896895325381683	0.614230486488741	0.54891357421875	0.403203125
6	traffic light	0.49013783164911723	0.9996243141127749	0.9986205425248301	0.5833412249965803	0.33011870222365575	0.2439304282086325
7	traffic sign	0.6009425811012489	0.9989286161301032	0.9962809898929431	0.6825326965510269	0.5758336539220328	0.4560229677142519
8	vegetation	0.9442767531402959	0.983476376524939	0.9766836397277064	0.9334912447267709	0.8763361684306112	0.8102360115739291
9	terrain	0.6437766084368367	0.99712203143845	0.9941815655598678	0.6480752982028812	0.2235806346451264	0.17284552119832183
10	sky	0.960432371194343	0.997199674250874	0.9959650746923375	0.9411261785934218	0.774073486328125	0.727723388671875
11	person	0.7993801747097334	0.9951688407727327	0.9926258003347652	0.737946180424451	0.4489291127004837	0.3494897286906505
12	rider	0.287353624290882	0.9995088274810553	0.9979760142032771	0.37932857944075	0.1540548006693522	0.10906025215431496
13	car	0.930353964325613	0.995670732475344	0.9914138718004027	0.9338782025486251	0.8161356362951807	0.7494536863273407
14	truck	0.33813924402445805	0.9986425341421907	0.9966556046704379	0.3782241637818531	0.07046090767358654	0.05636624592106517
15	bus	0.5446243869167015	0.9989774800633006	0.9972138428721365	0.6027852413991766	0.06338887655434489	0.05156908115419019
16	train	0.2263428101685977	0.9991512472870403	0.9982833770322943	0.22848151308527545	0.022347370783487957	0.015922020147512626
17	motorcycle	0.23229294344247942	0.9996858553207298	0.9990751594259606	0.28559511315626596	0.060682750591511125	0.04389458683909327
18	bicycle	0.695446164874904	0.9976737278695965	0.9955294437077676	0.6882205812146883	0.3403632977257477	0.26151721852932486
19	**Average	0.6144927165073157	0.9948149293001126	0.9911186394227582	0.6393935640377608	0.4239771021315254	0.36564894308995244

Figure 9: Evaluation Metrics on the R2U-Net model.

$$JS = \frac{|GT \cap SR|}{|GT \cup SR|} \tag{5}$$

The model is evaluated on the 'val' data split under 100 epochs . The R2U-Net is able to perform decently well on representing the data after the training. The metrics are calculated for individual classes and we could notice that the R2U-Net performs decently well on most foreground objects. The results among the classes are then averaged out and can be seen in Figure: 9 The model achieves the scores as SE - 0.614,SP - 0.994, ACC - 0.991,F1 - 0.639,DC - 0.423, JS - 0.365. We could note that apart from the mentioned metric we also have an F1 Score which potentially indicated that the model performs well 63% of the time.

verify these details

Change metrics picture

4.3.2 Task 2.2: Challenge Task - ASPP Recurrent Residual Convolution Neural Network with Attention Gates

We have improvised the R2U-Net Architecture by introducing Atrous spatial pyramid pooling (ASPP)[5] and also and Attention Gates [6] [7] and have achieved an improved performance with regards to increase in almost all the metrics. The introduction of the ASPP layer has helped the model in identifying the classes of images on a different scale. In order to compare the proposed model with the R2U-Net, we have computed the same metrics under the same 100 epochs and same batch size across all classes and have averaged it out as we did for R2U-Net. The model achieves scores as SE - 0.624, SP - 0.995, ACC - 0.9915, F1 - 0.656,DC - 0.430,JS - 0.372. We could note that the F1-score is 65% now. The outcome of the evaluation gives us with approximately more than a 1.5% increase in almost all metrics whereas in Specificity (SP) and Accuracy (ACC), they increase to approximately 0.02% and 0.05% respectively. The detailed results can be seen in Figure: 10

verify the details

5 Conclusion

We understood the segmentation task and implemented the R2U-Net architecture to it. We proposed an improvised architecture called the Atrous Spatial Pyramid Pooling Recurrent Residual Convolutional Neural Network based on U-Net with Attention Gates (ASPP R2U-Net with Attention Gates) that slightly improved on the results of R2U-Net at the cost of increase in number of parameters and of additional computation.

	Name	SE	SP	ACC	F1	Dice	Jaccard
0	road	0.9810196712245387	0.9874077189484858	0.985004208951264	0.9800909621464816	0.9449384155273437	0.9282400512695312
1	sidewalk	0.8323306239954262	0.9909455300060535	0.9823631286115045	0.8362545163221524	0.7157537841796875	0.6224810791015625
2	building	0.9364220544780436	0.9719011797327022	0.9641208677588062	0.9196580156482804	0.8575966796875	0.784402099609375
3	wall	0.3267329788977036	0.9978770249860515	0.9929538386527342	0.4048684251177311	0.12776081410748172	0.09473942562579066
4	fence	0.3283177129959227	0.9975355632220834	0.9920407031575286	0.4038356915008121	0.10451776010018808	0.0743441562103146
5	pole	0.5350629937318673	0.996785143825393	0.989963103919362	0.6116988730245584	0.5398409423828125	0.3970004577636719
6	traffic light	0.4761955282073045	0.9997885785172422	0.9987570146865297	0.6015244157802161	0.34188339558053527	0.25519502923843707
7	traffic sign	0.5973555363548294	0.9994388648900071	0.9967639811592821	0.7106539849457776	0.5948165801449321	0.474684041750503
8	vegetation	0.9473413342004653	0.9835533215284977	0.9772782995363116	0.935273784239001	0.8751849975585938	0.810271728515625
9	terrain	0.6026096675750687	0.9976745735801952	0.9943869269876199	0.6411687415094759	0.22510467726608802	0.17419971268752527
10	sky	0.9562643817681197	0.9976720335536595	0.9962816169507378	0.945268542461772	0.7787838861316383	0.7350584347405749
11	person	0.8213047174846518	0.9955404439706133	0.993277348219964	0.7604017287896112	0.4511750065824635	0.355168332996023
12	rider	0.36189497365844187	0.9993737541655533	0.9980016713180421	0.43806939519210053	0.1696760234352653	0.12310179092791315
13	car	0.9370748499313132	0.9954616932612346	0.991656473493849	0.9360583904190504	0.8163188643244854	0.7484564723738705
14	truck	0.3389788308319437	0.9994595399385922	0.9974726783951556	0.4465829080562358	0.08232477952807729	0.06716097115371633
15	bus	0.5817077931693658	0.9994741216126907	0.9978525012360006	0.6777203860349857	0.11133481573870802	0.09522833786611481
16	train	0.38153955919532206	0.9976105604904976	0.9969187076701153	0.21759590973825507	0.014488449729824462	0.010292969304001677
17	motorcycle	0.21217825249518474	0.9998040626938018	0.9991772653368582	0.29101498003662457	0.06897863219766055	0.04891085157207414
18	bicycle	0.71608314695564	0.9980624924555268	0.9960618680302527	0.7206851054787683	0.3625911888218325	0.2801066123292037
19	**Average	0.6247586635342712	0.9950192737567832	0.9915964317932587	0.6567591977074679	0.43068787858026936	0.3725811871071489

Figure 10: Evaluation Metrics on the ASPP R2U-Net with attention gates model

References

[1] Wikipedia contributors. Image segmentation — Wikipedia, the free encyclopedia, 2021. [Online; accessed 18-March-2021].

[2] Md. Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M. Taha, and Vijayan K. Asari. Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. *CoRR*, abs/1802.06955, 2018.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[6] Ozan Oktay, Jo Schlemper, Loïc Le Folgoc, Matthew C. H. Lee, Mattias P. Heinrich, Kazunari Misawa, Kensaku Mori, Steven G. McDonagh, Nils Y. Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. *CoRR*, abs/1804.03999, 2018.

[7] LeeJunHyun. Image_Segmentation – GitHub, 2021. [Online; accessed 30-March-2021].

[8] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross B. Girshick. Pointrend: Image segmentation as rendering. *CoRR*, abs/1912.08193, 2019.