Project Report: NNTI Winter Semester 2021

Arpit Jadon
7010483
arja0001@stud.uni-saarland.de

Awantee Deshpande
2581348
s8awdesh@stud.uni-saarland.de

1 Introduction

In this project, we attempt to implement methods of image segmentation on different datasets using different model architectures. Unlike image classification, in segmentation tasks, every pixel in an image is associated with a class. There are two types of segmentation - semantic and instance. Here, we train models to perform semantic segmentations on objects in images i.e. all objects of the same class get the same label. The pattern of the report is as follows:

1. Tasks: The entire project is divided into three tasks. Each task implements a different model on some dataset. The project can be found in the group's Github Repository[1].
2. Datasets: We use the PASCAL VOC Dataset for one task and the Cityscapes Dataset for other tasks. The datasets are described in more detail in Section 2.
3. Model Architecture: We follow SOTA models and implement similar architectures for each Task. Task 1 implements standard UNet [1] architecture with a VGG16 [2] encoder, Task 2 implements R2UNet [3], and Task 3 implements UNet3+[4] and DeepLabV3 [5]. These are described in more detail in Section 3.
4. Metrics: We use different metrics like instersection over union (IoU), dice coefficient, f1-score, etc. for evaluating our models. These are described in Section 4.

The results that we obtain and our subsequent analysis on them are outlined at the end in the evaluation and analysis sections.

1.1 Project Structure

The directory structure in the repository is outlined as follows:

```
Vision-Project-Image-Segmentation
 ├── README.md
 ├── CV_Task_1.ipynb
 ├── Task_2_and_3
 │    ├── CV_Task_2.ipynb
 │    ├── CV_Task_3_UNet3Plus.ipynb
 │    ├── CV_Task_3_DeepLabV3.ipynb
 │    └── Helper modules - augmentations, dataloader, model, train, evaluate,
 │        metrics, visualise, util
 ├── Visualisations
 └── Report
```

The implementation scripts for Tasks 1, 2 and 3 are in their respective Jupyter Notebooks. Helper functions to work on the data, represent the model architecture, training, and evaluation are in the subdirectory for tasks 2 and 3 as Python files. The `Visualisations` folder contains segmentation prediction examples for all three tasks.

[1]https://github.com/arpit-jadon/Vision-Project-Image-Segmentation

2 Datasets

We utilise two datasets in this project.

PASCAL VOC Dataset : This dataset contains 20 object categories and has size of 2 GB. The classes include - vehicles, household, animals, and other: aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor, bird, cat, cow, dog, horse, sheep, and person. There are 1464 images for training and the number of validation images is 1449. Images and their labels are separately available. Including the background, we consider a total number of 21 classes.

Cityscapes Dataset : This dataset has images of street scenes. It has a size of 12 GB. For fine level label annotations, the dataset comprises 2975 training images and 500 validation images. The original dataset has 34 classes in all. However, in this project, we retain only the frequent classes and remove the rare ones, leaving 19 classes in all. All these rare classes are considered as an unlabelled class, giving us a total of 20 classes.

For both these datasets, we use manual data loaders. Since semantic segmentation on small training sets does not give good results, we also augment these datasets using different transforms like rotations and flips.

We use the PASCAL VOC Dataset for Task 1. The fine-tuned Cityscapes Dataset is used for Tasks 2 and 3.

3 Model Architectures

For Task 1, we have been given a half-implemented template on which we have to base our model architecture. For this, we choose the standard UNet architecture, which is a precursor to the R2UNet architecture we implement in Task 2. The standard UNet architecture looks as follows (Figure 1):

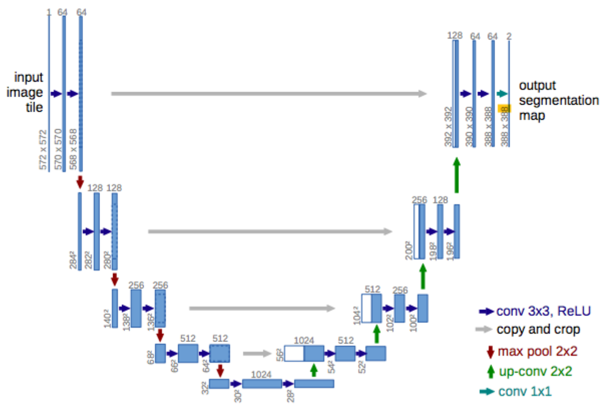


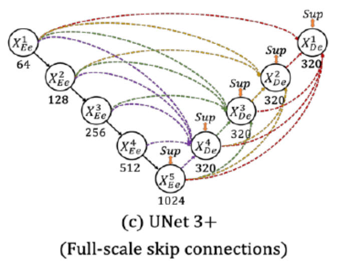
Figure 1: UNet Architecture (Source: Ronneberger et al.)

The U-Net family of architectures have an encoder-decoder style that is built as an extension to the fully convolutional networks (FCN) for semantic segmentation [6]. The FCN is modified in order to yield precise segmentations with fewer training images. U-Net has a contracting (or down-sampling path), also called as encoder and an expansive path (or upsampling path), also called as decoder. The expansive path is almost symmetric to the contracting path. The contracting path has a typical convolutional neural network (CNN) like architecture. Along the contracting path, the image is progressively downsampled while the number of channels are increased capturing the high resolution contextual features within the image. The expansive path consists of a series of steps where the feature map is upsampled followed by up-convolution, which reduces the number of feature channels. At the up-convolution step, the corresponding high resolution feature map from the contracting path is concatenated with the feature map in the expansive path. The up-convolution step is followed by successive convolutions. At the very last step, a 1x1 convolution maps all feature vectors to the total number of classes. The concatenation of feature maps between the encoder and decoder is an important step since it helps in combining the contextual information with the localization information. In our use case, for task1, we used VGG16 [2] as our encoder while utilizing the ImageNet [7] pretrained weights for VGG16 to benefit from the prelearned parameters and perform better on the PASCAL VOC dataset .

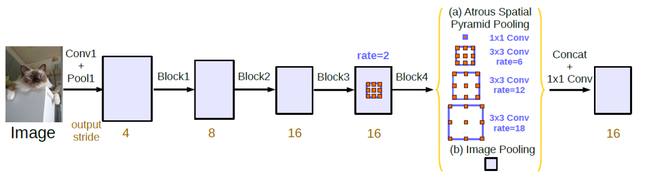
We also tried training the standard U-Net architecture without any pretrained encoder (i.e., training U-Net from scratch) but it did not give us good results, and hence we used pretrained VGG16 weights for the model. The implementation of Task 1 is in the notebook `CV_Task_1.ipynb`.

For Task 2, we use the R2UNet architecture provided in the project specifications. The architecture is almost entirely similar to the UNet architecture from Task 1. The main difference is that instead of a simple convolutional layer, the convolution block has a residual unit with recurrent convolutions. A parameter t passed in the initialisation of the convolution block determines the number of recurrent relations in the block. We have kept $t = 3$ in the implemented architecture. The channel depths per layer are kept in an increasing fashion similar to the UNet architecture in Task 1. However, in the final output, since we are retaining only 19 classes (20 classes including the unlabelled class) from the original 34 in the Cityscapes dataset, we change the output channels accordingly. Both architectures have max pooling kernels of 2x2. Task 2 has been implemented in the notebook `CV_Task_2.ipynb`

Task 3 strives to improve on the results from Task 2. For this, we have considered two state of the art models. One is an extension of the UNet and R2UNet architectures, called UNet3+. The second one is the DeepLab V3 architecture by Google (Refer Figure 2).



(a) UNet3+ (Source: Huang et al.)



(b) DeeplabV3: Parallel atrous convolution (ASPP) modules, augmented with global image context features using global average pooling. (Source: Chen at al.)

Figure 2: Architectures for Task 3

In UNet3+, full scale skip connections are used. These help to combine the low level features from the lower layers with the high level semantics from the upper layers, as observed in fig 2a. The encoder-decoder connections are applied by max pooling operations, while the decoder-decoder connections are formed by upsampling operations similar to UNet and R2UNet. UNet3+ also provides deep supervision with the help of a plain 3x3 convolution layer applied at the end of each decoder block. The UNet3+ model provided only a slight improvement to the previous tasks, so we tried to implement DeepLabV3 to see if the performance improved.

In contrast to all these architectures, the DeepLab models employ a technique called atrous convolutions for image segmentation tasks. The atrous rate corresponds to the stride over the input feature map. The idea is that consecutive striding is harmful for image segmentation because the spatial information is lost at the deeper layers. The atrous rate is controlled by the *dilation* parameter to the `torch.nn.Conv2d` module used for implementing the convolution block. Using the atrous convolutions, we can maintain a constant stride while increasing the field of view without any increase in the number of parameters. Thus, we can have output feature maps of larger size, eventually benefiting our semantic segmentation task. The code for atrous convolutions is located in the file `model.py` in the class `ASPP`, which stands for “Atrous Spatial Pyramid Pooling". The complete architecture contains cascade blocks of residual networks followed by the atrous convolution and pooling. We use ResNet18 [8] as our residual network, which is used to generate a feature map with an input image, this feature map is then passed to the ASPP unit. The ASPP unit mainly consists of one 1x1 convolution, three 3x3 convolutions with atrous rates of 6, 12, and 18 respectively, a global average pooling layer to capture the global context, and finally another 1x1 convolution to generate the final logits. Using 3x3 convolutions allow us to capture the information at multiple scales. All the features obtained after applying the first 1x1 convolution, three 3x3 convolutions with different atrous rates, and the average pooling are concatenated and passed through the second 1x1 convolution to generate the final ASPP output. Figure 2b depicts this complete pipeline. Finally, the logits obtained from ASPP are upsampled to the required spatial dimension using bilinear upsampling. UNet3+ has been implemented in `CV_Task_3_UNet3Plus.ipynb` and DeepLabV3 has been implemented in `CV_Task_3_DeepLabV3.ipynb`.

4 Metrics and Loss Functions

To quantify the performance of the models, we evaluate them on the following metrics:

1. Intersection Over Union/ Jaccard Score: It gives the area of overlap between the predicted segmentation and ground truth divided by the area of union between the predicted segmentation and ground truth. The mean IoU is calculated by taking the IoU of each class and averaging them.

$$IOU = \frac{target \cap prediction}{target \cup prediction}$$

2. Dice Loss: Dice coefficient is twice the area of overlap between the ground truth and prediction divided by the total number of pixels of both images.

$$Dice = \frac{2 * (target \cap prediction)}{(target \cap prediction) + (target \cup prediction}$$

3. Per Class Pixel Accuracy: We take an average of the pixel accuracy for each class, i.e., average percentage of pixels for each class that are classified correctly.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

4. F1 score: F1 score is the harmonic mean of precision and recall. For semantic segmentation tasks, f1 score is equivalent to the Dice coefficient. However, following the norm of Alom et al. [3], we also implement it separately with the `sklearn metrics` library.
5. AUC score: AUC scores represents the area under the probability curve that plots true positive rate against false positive rate at different thresholds. This is also implemented using the `sklearn metrics` library.

These metrics are calculated for a per epoch basis to analyse how to model performance improves/degrades with each epoch. For all the tasks, we use Cross-Entropy Loss as the loss function during training, as is the standard for classification tasks.

5 Training and Evaluation

We do an ablation study over different hyperparameters for all three tasks. Based on the resources available and the performances observed, we fix the value of batch size[2], number of epochs, and learning rate. To accommodate the process on the available resources to the maximum capacity, we also adjust the image sizes to get sufficient batch sizes.

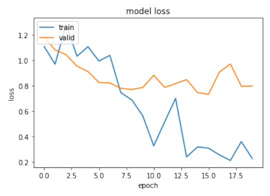
For Task 1, the image size is 512x512. We retain a batch size of 12 and train for 20 epochs with a learning rate of 10^{-4}. We use the Cross Entropy Loss available in the PyTorch library, and the optimiser used is Adam. Training the model from scratch does not give good results (IoU and Dice loss of the order 0.03 - 0.04), so we use VGG-16 pretrained weights for the encoder part of the UNet.

For Task 2, we resize the images to 256x256 and change the batch size to 6 and train for 20 epochs. The learning rate is 10^{-4}. The loss function and optimiser are same as those from Task 1. The PyTorch implementation of the optimiser can also add weight decay to the bias parameters or other parameters of shape 1. In order to avoid that, we have implemented our own weight decay function

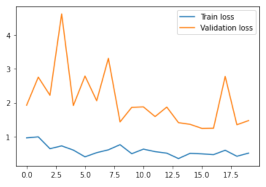
[2]The values mentioned for batch size were not tested for higher values than the ones mentioned because of the limited resource capacities

(l2 value of 0.0001) that takes care of this issue. In order to improve further, we have also initialised the model weights with kaiming weights [9].

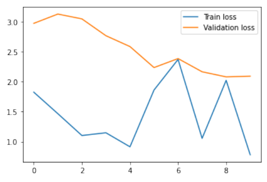
For Task 3, we check different parameters for UNet3+ and DeepLabV3. We train UNet3+ on an image size of 256x512, batch size of 3, and 10 epochs, and a slightly higher learning rate of 10^{-3}. For DeepLabV3, the images are of size 256x256. We keep a batch size of 84, number of epochs as 30, and a learning rate of 10^{-4}. Like tasks 1 and 2, the loss function is Cross-Entropy loss and the optimiser is Adam, with the weight decay function (l2 value of 0.0001).
To compensate for the smaller training set, we use augmentations like random rotation, random translation, and random horizontal and vertical flips for these tasks as well.
Figure 3 shows the training and validation curves for each of these architectures.



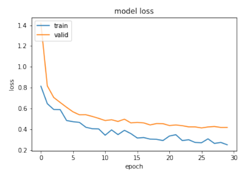
(a) Task 1



(b) Task 2



(c) Task 3 (Unet3+)



(d) Task 3 (DeepLabV3)

Figure 3: Training and Validation Curves

Evaluation: As mentioned in Section 4, for Task 1, we evaluate based on dice scores, f1 scores, and AUC. Tasks 2 and 3 are evaluated on all the 5 metrics (IoU, Dice coefficient, per class pixel accuracy, f1 score, and AUC score). We present the results below.

5.1 Results

Overall scores: The average scores of IoU, Dice, F1, Classwise Pixel Accuracy, and AUC over the training process for the three tasks are listed in Table 1.

Table 1: Average metric scores for the three tasks

Task	Model	IoU	Dice	F1	Pixel Accuracy	AUC
Task 1	UNet	–	0.3608	0.1595	–	0.8360
Task 2	R2UNet	0.146	0.193	0.198	0.197	0.631
Task 3	UNet3+	0.213	0.259	0.279	0.263	0.774
Task 3	DeepLabV3	0.483	0.607	0.419	0.582	0.882

Per Epoch Weights: The model weights are updated in every iteration and stored for evaluating how the performance is affected with each epoch of training. We plot the change in these metrics at every epoch using these saved weights.

The plots for task 1 are illustrated in Figure 4. They contain only Dice Score, F1 Score, and AUC for evaluation. The plots for tasks 2 and 3 contain all 5 metrics, and are present in Figures 5, 7, and 6.

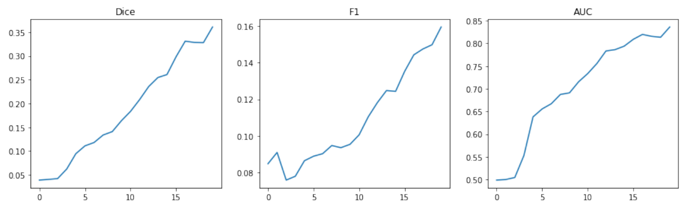


Figure 4: Task 1: Per epoch values for metrics: Dice coefficient, F1 score, AUC

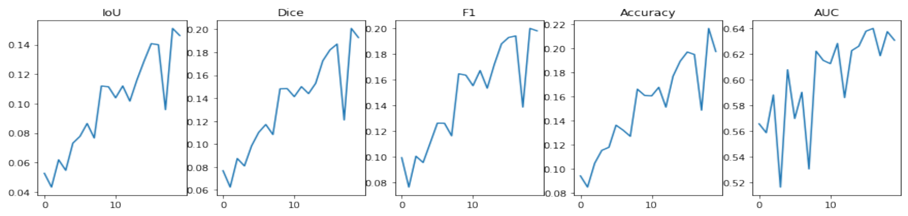


Figure 5: Task 2: Per epoch values for metrics: IoU, Dice coefficient, F1 score, Accuracy, AUC

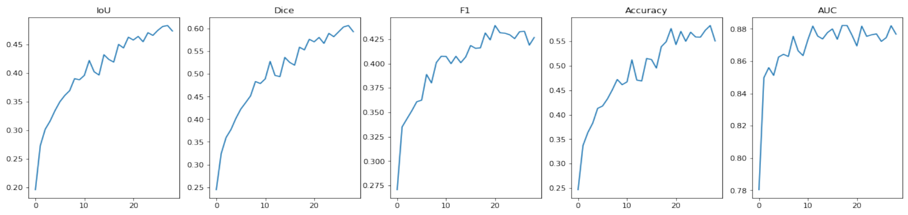


Figure 6: Task 3 DeepLabV3: Per epoch values for metrics: IoU, Dice coefficient, F1 score, Accuracy, AUC

Visualisation of Predictions: We also publish the visualisations of the original images, ground truths i.e. pixel-wise class labels, and the predicted segmentations for each task. The visualisations for task 1 are in Figure 8a, those for task 2 are in Figure 8b. The task 3 visualisations are in Figures 9a and 9b.

6 Analysis

The scope of this project is to understand how image segmentation works with the help of Task 1, implement the R2UNet architecture in Task 2, and improve the results obtained in Task 2 in the 3rd task.

The training and validation loss curves convey quite a lot about the model. In Task 1, we can see that the validation curve in Fig. 3a saturates while the training curve continues to decrease. So, the model begins to overfit somewhere around 15 epochs. Possible reasons could be the inadequacy of the training data, since we only used half of the dataset for training (1464 images) over the complete set of 21 classes. One solution here could be to reduce the learning rate (e.g. by using a learning rate scheduler), augment data, and perhaps add few dropout layers. However, we are still able to get decent results and metric scores because we utilise a pretrained VGG16 encoder for U-Net, which is already trained on the ImageNet dataset.

For task 2, unlike task 1, we train the R2UNet architecture from scratch. Thus, we don't have any pre-learned features or parameters in our network to start with. In the original paper, R2UNet was trained for binary segmentation task. However, for the Cityscapes dataset, we need to segment 19 classes while utilizing only 2975 training images, which is a relatively difficult task. Thus, from Fig. 3b, we observe that the our model has some amount of overfitting. However, during our initial attempts at training R2UNet, we faced even severe overfitting issues. We were able to overcome these to some extent by using different regularization methods. We augmented the training set with

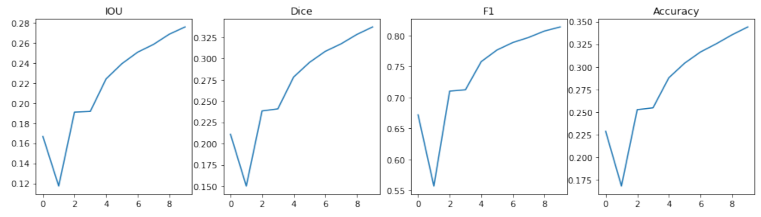
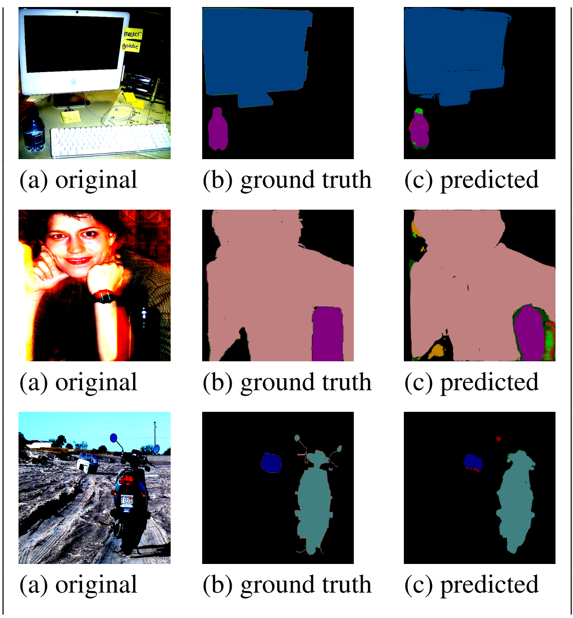
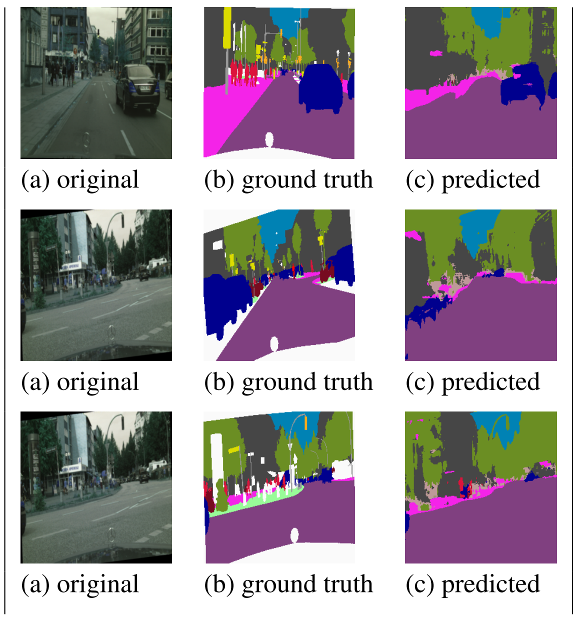


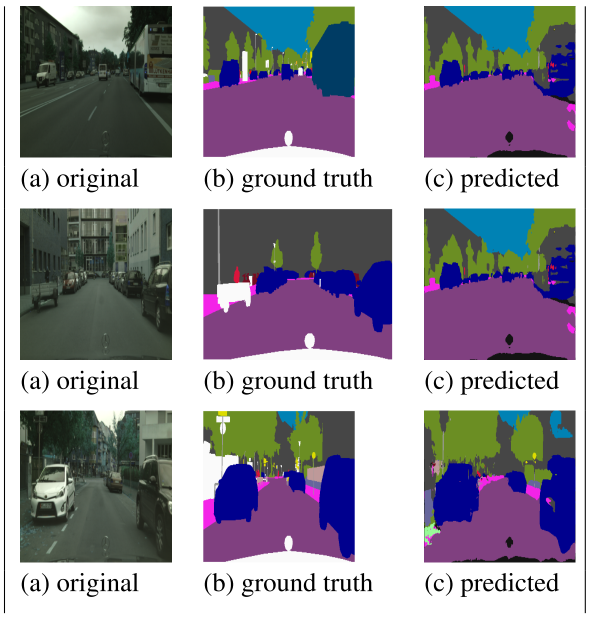
Figure 7: Task 3 UNet3+: Per epoch values for metrics: IoU, Dice coefficient, F1 score, Accuracy



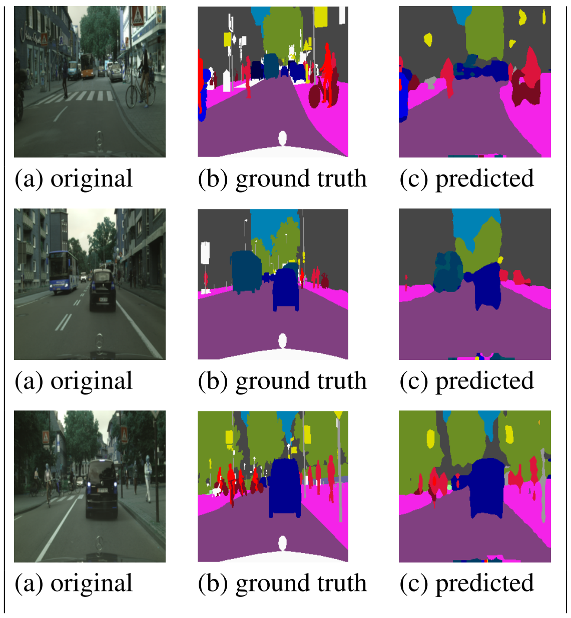
(a) Task 1 Visualisations



(b) Task 2 Visualisations



(a) Task 3 Visualisations (UNet3+)



(b) Task 3 Visualisations (DeepLabV3)

random rotation, random horizontal flipping, and random vertical flipping. Moreover, we added dropout layers (with a dropout rate of 0.5) [10] at the end of our up-convolution block as well as at the end of the convolution pipeline in our recurrent block. We also used weight decay (l2 value of 0.0001) as well as the 'kaiming' weight initialization. Also, empirically, keeping the number of recurrent relations in the recurrent residual block ('t' parameter) to 3 helped in improving the model's performance. However, there is still quite some scope of improvement in terms of resolving this overfitting issue. We could tweak (reduce) the number of model parameters and using transfer learning. Transfer learning can be used by training R2UNet on another similar dataset like PASCAL VOC and then retraining this model on Cityscapes while preserving the encoder weights. We can also try training the model for a higher number of epochs (while training R2UNet from scratch). However, all these methods require significant time and computation.

In task 3, we observe a relatively good convergence 3c for DeepLabV3, and a poorer curve for UNet3+ 3d. Our training attempt at UNet3+ suffers from the same issues as we discussed previously i.e. we train UNet3+ from scratch while DeepLabV3 benefits from the pretrained ResNet18 weights used to produce feature maps. Moreover, UNet3+ is trained on smaller batches and for a lesser time, resulting in a relatively worse performance. The results could be improved for UNet3+ by modifying it to a simpler less complex architectures with relatively lesser number of parameters. The validation loss is still decreasing for UNet3+, indicating that the training the model for significantly more epochs with some hyperparameter tuning can result in improved performance.

When it comes to metrics, we analyse our models in 5 ways. IoU and Dice coefficient are amongst the most popularly used metrics for image segmentation tasks. While F1 score is equated to the Dice coefficient (numerically), as mentioned before, we have calculated it separately in line with the R2UNet paper by Alom et al.[3], since they can have different results for multiclass segmentation. We see a steady improvement in Table 1 for the IoU, Dice, F1 and Pixel Accuracy values over the models in Tasks 2 and 3, as expected. Also, we calculate per class pixel accuracy instead of overall accuracy since overall accuracy might not give an accurate estimate of the model's performance with respect to different classes. Moreover, metrics like AUC are designed to work better on binary classification tasks. We see a good AUC score (> 0.5) for the models, but their IoU and Dice scores are relatively lower. Another point to note here is that the metric scores are lower for those models that have been trained from scratch, despite their superior architecture. Among all the models, DeepLabV3 performs the best in terms of all metrics.

Computer Vision tasks are always heavy computation based problems. Implementing deep learning solutions for them involves a lot of careful evaluation, not just for the model architecture, but also with respect to various hyperparameters. It is a complex task to do a grid-like search to find the best sets of hyperparameters like batch size, learning rate, optimiser, loss function etc. that will result in the optimal model. In our case, we have chosen these parameters with limited resources, but they can definitely be improved on with a more comprehensive but careful analysis.

Thus, we can see that there are plenty of ways in which we can refine the results we get. We can improve the loss curves, lower the train time complexity, and overcome the lack of the training data by the expedient of simplifying the model architecture w.r.t the parameters and number of layers as well as using transfer learning as mentioned previously. In this project, we have replicated the architectures from the original papers as closely as possible.

We have added 2-3 types of augmentations to the dataset to compensate for the lack of enough training data. Adding more such augmentations is a possibility, especially for Task 1, where we haven't used any augmentations at all.

Another possible point of research would be the type of the loss function. We have used the standard Cross Entropy Loss everywhere. Implementations have also focused on using Jaccard loss or Dice loss for evaluation while training. It can also be possible to use a combination of Cross-Entropy loss and Dice loss by weighting them appropriately.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[3] 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. 2018.

[4] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. Unet 3+: A full-scale connected unet for medical image segmentation. 2020.

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. 2017.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[7] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.