Semantic Segmentation for Autonomous Driving
Deep Squad

Bidhan Ghimire
Prashant Subedi
Amit K Rai
Khalid Alkady

05/08/2022

Contents

Abstract

Semantic segmentation is a necessary tool for autonomous driving applications. In this report, we explored the implementation of DeepLabV3+ models to accomplish the task of semantic segmentation on the benchmark datasets — CamVid and Cityscapes. In the third milestone, we explored the CamVid dataset, applied preprocessing to it and used it to train DeeplabV3+-based models. We were able to obtain similar accuracy and mean IoU to the referenced sources. In the fourth milestone, we studied the effects of image resolution, data augmentation and different backbone networks on the model performance. In the final milestone, we ran several DeepLabV3+-based models of different image resolutions and backbone networks (i.e., ResNet 18 and ResNet 50) on the Cityscapes dataset. The results of this study showed that the DeepLabV3+ models were able to achieve high global accuracies (i.e., around 90%), despite the image resolution and backbone network used. However, due to the class imbalance issues within the datasets, the mean IoU values were relatively low. Furthermore, the results showed that reducing the image resolution from 100% to 25% drastically reduced the training duration by around 90%. The best DeepLabV3+ models were able to achieve nearly accuracies of 90 – 92% for both datasets, while the highest mean IoU achieved for the CamVid dataset was nearly 73% and for the Cityscapes dataset was nearly 50%.

Chapter 1

Milestone 1: Project Ideas

1.1 Introduction

We brainstormed a few project ideas and discussed them further with the TA. Project Idea Sources page on Canvas proved to be a valuable resource while looking out for ideas. Apart from the page, our project ideas also came from Kaggle, and peer discussions. A few of our initial project ideas were

1. Semantic segmentation for autonomous driving
2. Auto-complete and Auto-Correction (Language Modeling) with sequence-to-sequence model
3. Image colorization
4. Recommendation of top tracks based on seed tracks/playlist for playlist continuation [1],
5. Prediction of whether the user will skip or play the subsequent tracks based on the user's interaction with the first half of listening session [2]

Next, we discussed these ideas with the TA and held several group meetings for further discussions. Eventually, we decided to proceed further with ideas 1 and 3. Table 1.1 shows what each team member worked on for the proposal.

Table 1.1: Contributions by team member for Milestone 1.

Team Member	**Contribution**
Amit K Rai	Semantic Segmentation
Bidhan Ghimire	Image Colorization
Khalid Alkady	Semantic Segmentation
Prashant Subedi	Image Colorization

1.2 Project Idea 1: Semantic Segmentation for Autonomous Driving

1.2.1 Problem Statement

Semantic segmentation is the task of assigning each pixel of an image to a corresponding class label from a predefined set of categories [3].

1.2.2 Applications

Semantic segmentation has applications in autonomous driving, field crop monitoring and identification. Autonomous driving requires a real time and accurate representation of the environment around the subject vehicle. Self-driving vehicles must understand their surrounding environment, i.e., other cars, pedestrians, road lanes, traffic signs or traffic lights [3, 4]. In autonomous driving, detecting objects and putting a label and boundary box is not sufficient. An instance mask of each object generated by semantic segmentation is in many ways richer than a box, as it allows an informed reasoning about occlusion and depth layering.

To achieve semantic segmentation in real time, we may have to trade execution speed for achievable segmentation accuracy [5]. In the following sections some of the benchmark datasets and deep learning-based architectures for semantic segmentation in the field of autonomous driving will be discussed.

1.2.3 Datasets

Several benchmark autonomous driving datasets have been created for training deep learning-based models for semantic segmentation . The datasets were collected using cameras and LiDAR (Light Detection and Ranging) scanners. They are available in the format of video sequences, 2D images, and point clouds (i.e., collected by LiDAR scanners). However, our focus in this project will be on some of the benchmark datasets collected by cameras (i.e., video sequences and 2D images) only. KITTI [6] is one of the popular benchmark datasets for semantic segmentation. It comprises more than 320,000 labeled images in a driving distance of 73.7 km in Karlsruhe, Germany. The labels incorporate 19 classes. Moreover, the Cityscapes dataset [7] contains videos collected from 50 different cities. More than 25,000 images were selected from the frames of the recorded videos to create an image dataset of versatile backgrounds and objects. 5000 images were finely annotated, while the remaining 20,000 images were coarsely annotated. The data were collected over several months and during different weather conditions (i.e., summer, fall, and spring days). In addition, the CamVid image dataset [8] is among the widely used datasets in the area of semantic segmentation for autonomous driving. The CamVid dataset consists of five video sequences of road scenes that are divided into images of 960 X 720 resolution. The dataset uses 32 classes. Furthermore, RaidaR [9] is one of the most challenging datasets for the semantic segmentation task as it contains

scenes taken during rainy conditions. Training the semantic segmentation model on images of rainy weather is challenging and critical for the robustness of the model during testing. Also, Berkeley Deep Dive dataset [10] is from different weather conditions (i.e., cloudy, sunny, rainy, snowy, and foggy) and parts of the day (i.e., daytime, and nighttime). The dataset is made of 100,000 40-second-long video sequences of moderate HD resolution sampled at a frequency of 30 frames per second (fps). The extensive dataset contains road objects, lane markings, 1 million cars, 300,000 street signs, 130,000 pedestrians, and drivable area information.

1.2.4 Literature Review

The past studies used several different approaches for building the benchmark deep learning-based architectures in semantic segmentation. In this section, we will discuss some of these novel architectures . The main approaches of semantic segmentation architectures are based on fully convolutional networks(FCNs) for end-to-end pixel-wise learning [5, 3]. Figure 1.1 shows the modular deep learning architecture for performing semantic segmentation. The architecture consists of a feature extraction and decoding portions. The feature extraction module is a convolutional network that learns feature maps from the input images. Then the decoder module consists of deconvolutional network for upsampling the feature maps for providing dense predictions. Transfer learning techniques can be used for the implementation of popular pre-trained networks in the encoder or decoder modules [5, 3, 11, 12, 13, 14, 15], as shown in Figure 1.1. For an instance the ResNet 18 network can be used as the backbone of a semantic segmentation network [16]. Deeplab v3+ is an encoder-decoder architecture that was trained using ResNet 18. The network was trained on the CamVid dataset [8] and was able to get a global test accuracy of 87.65% [16]. Furthermore, refinements can be made to the FCN modular architectures by incorporating context aware and temporal models into the FCN (i.e., encoder-decoder) architectures [3]. The context and temporal data can be considered by adding recurrent neural network cells between the convolutional layers, such as LSTMs and RNNs [3, 14, 15]. The temporal data can be inputted to the networks through video sequences. Hence, the semantic segmentation is applied to video sequences rather than still images [17, 18].

1.3 Project Idea 2: Image Colorization

1.3.1 Problem Statement

Image Colorization involves painting a grayscale image with colors to make it more aesthetically appealing and meaningful [19]. In a grayscale image, each pixel only contains the information about the amount of light [20]. The problem of colorization is to estimate RGB colors of each pixel.

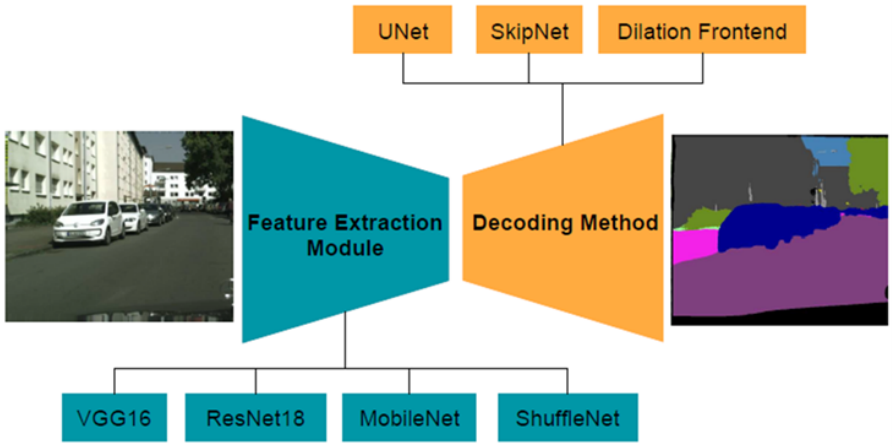

Figure 1.1: Overview of the different components in the framework with the decoupling of feature extraction module and decoding method (source: [3]).

1.3.2 Applications

Colorization can be used for restoring/enhancing old grayscale images to make them more meaningful. Since most of the images of the past were black and white, adding colors can restore the missing details of many historical images. They can also be used to restore images that have been converted to grayscale to save space or communication bandwidth [21].

1.3.3 Dataset

Some of the common datasets used for colorization are listed below[19]. Images are first converted to grayscale and then colorized to analyze models. Some models also use the labels present in these datasets for better colourization.

1. COCO-stuff dataset [22]
2. Places205 [23]
3. PASCAL VOC dataset [24]
4. CIFAR datasets [25]
5. ImageNet ILSVRC2012 [26]

1.3.4 Literature Review

Manual process of colorization is slow, labor-intensive, and tedious. Therefore, deep learning models are proposed to make colorization automated and time

Figure 1.2: Image Colorization results from [30].

efficient.

Deep Colorization [27] clusters a large image database into various clusters and trains a deep neural network(DNN) for each cluster. Given a target grayscale image, it is assigned to the DNN trained with the nearest cluster. Feature descriptors are extracted at each pixel as input to the DNN and output is the chrominance values of the pixel. [28] uses CNN to colorize grayscale image. It takes grayscale image as input and predicts one of 313 ab pairs in L*a*b color space for each pixel. Scribber [29] requires a user input in the forms of color strokes and sketches. It uses a deep generative adversarial architecture to color images. Iizuka et.al [30] proposes a model that combines global prior information such as whether it is outdoors or indoors, with local features like a regional texture or objects to colorize the image. The global features are learned with the help of labels for images in the datasets. Fig 1.2 shows result from their work. ChromaGAN[31] uses sentimental class distribution learning and generative adversarial networks to create a network that can color an image based on semantic understanding.

Deep Learning based models have shown good results for image colorization problems. However, the state-of-the-art approaches have not been used in real life critical applications due to network complexity, inadequate metric for evaluation and inability to handle real world degradation. [21]

1.4 Conclusions

At this milestone, we have presented two project ideas, and explored existing works and datasets for both semantic segmentation and image colorization. Semantic segmentation relies heavily on encoder decoder deep learning architectures, while image colorization has used GANs as well as CNN based techniques. We will proceed with one of these ideas as our class project.

Some of the questions we have before moving further are

1. Is it sufficient to use existing architectures with minor changes ?
2. Is it okay to use benchmark datasets that have already been used or do we need to create our own ?
3. Are there any performance goals we would have to meet ? In terms of metrics like accuracy or F1 score.

4. Is it possible to have a model that does both still images and videos for semantic segmentation ?

Chapter 2

Milestone 2: Project Selection

2.1 Introduction

Semantic segmentation is often used to recognize and locate certain categories of objects by assigning a label to each pixel [3]. As many tasks in computer vision, the state-of-the-art for semantic segmentation has been recently revolutionized by deep learning, and in particular convolution neural networks(CNN). Semantic segmentation has various application in the area of plant phenotyping, medical imaging, autonomous driving, etc.
Autonomous driving means that a vehicle can be driven without active physical control or monitoring by a human operator with the help of technology [32]. This technology is growing rapidly to meet the needs of road safety and transportation efficiency. Autonomous vehicles (AVs) have the potential to shape urban life and significantly modify travel behaviors. For the urban world, it has remained an important but elusive goal. Many notable attempts have been made, and several important milestones have been reached, starting from the Defense Advanced Research Projects Agency (DARPA) [33] to Tesla's FSD (full self-driving Car) [34]. Autonomous driving can be used in many applications where it may be inconvenient, dangerous, or impossible to have a human driver on site.
In the area of autonomous driving, semantic segmentation is an important step in the perception for autonomous vehicles, providing object level environmental understanding. Its performance directly affects other functions in the autonomous driving car, including decision-making and trajectory planning [3, 4].

2.2 Problem Specification

Statement: Semantic segmentation is the task of assigning each pixel of an im-

age to a corresponding class label from a predefined set of categories [3, 5]. **Our goal** is to reproduce a benchmark deep-learning based architecture to perform the semantic segmentation task on benchmark autonomous driving datasets.
Motivation: As per NHTSA, annually 38K people die in USA due to road accidents and 4.4M have serious injuries [35]. Human behaviour is the root of top 3 reasons of road accidents. These are distracted driving, drunk driving and reckless driving. Insurance companies loose 160 billion USD every year due to road accident.
Colorado State University's research found that if 90% of the cars in the USA were to become fully autonomous, an estimated 25,000 lives could be saved every year, with economic savings estimated at over 200 billion dollar a year [32]. It can help in avoiding the costs of crashes, including medical bills, lost work time, and vehicle repair. Fewer crashes may reduce the costs of insurance. It can help senior citizen and people with disability to be independent and live their life as per their will. In a fully automated vehicle, all occupants could safely pursue more productive or entertaining activities, like responding to email or watching a movie. A high-level overview of autonomous vehicle industry and its impact on today's society is illustrated in Figure 2.1.
Therefore, there is a critical need to employ deep-learning based semantic segmentation architectures to facilitate autonomous driving. Self-driving vehicles must recognize their surrounding environment (i.e., other cars, pedestrians, road lanes, traffic signs or traffic lights) to navigate without making accidents, and this can be achieved through the use of robust semantic segmentation deep-learning based architectures [3, 4, 5].

Resources: The Cityscapes [7] and CamVid [8] benchmark datasets will be used in this project. Cityscapes is available over tensorflow dataset library and will be directly downloaded from there. We will download the CamVid dataset manually and upload it to our directories on crane. The directory of the uploaded data will be utilized in the python training codes for data loading. In addition, the standard Tensorflow-GPU environment, which we are using for our homework and hackathon exercises, will be used in this project. The list of python libraries and computational resources needed to achieve the project objectives is as follows:

1. Jupyter lab
2. Python libraries: Numpy, Panda, Scikit, Matplotlib, Tensorflow, Keras, and Tensorflow-Dataset
3. Crane for training and testing the model

2.2.1 Benchmark Autonomous Driving Datasets

Several benchmark autonomous driving datasets have been created for training deep-learning based models for semantic segmentation. Generally, the datasets are available in the format of video sequences, 2D images, or point clouds (i.e.,

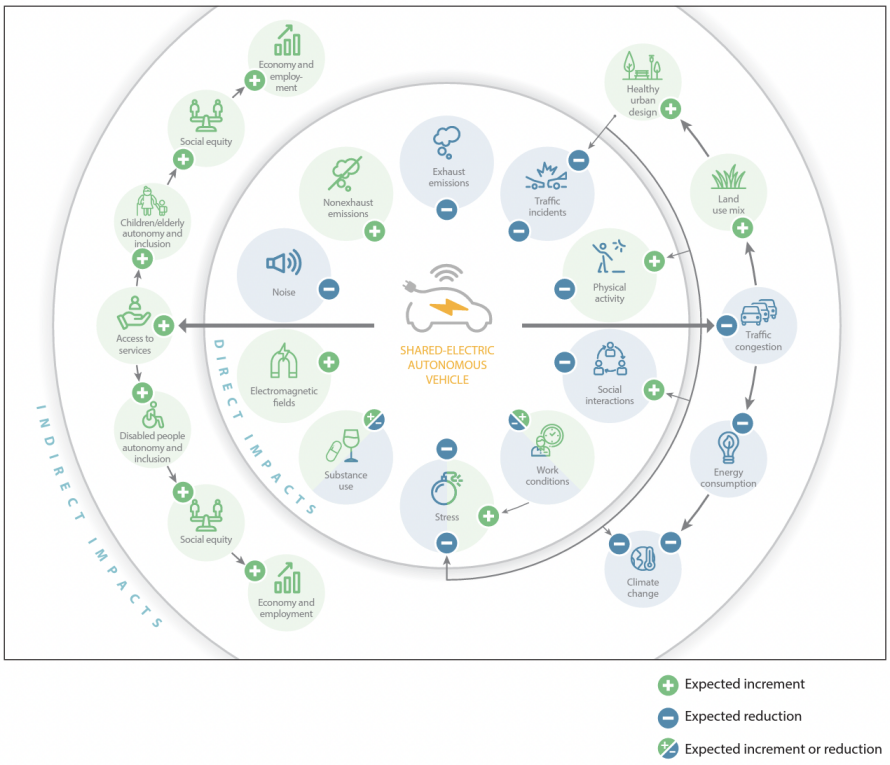

Figure 2.1: Autonomous Vehicle and its Impact (source: [32]).

collected by LiDAR scanners). In this project we will focus on some of the benchmark datasets collected by cameras (i.e., video sequences and 2D images) only. The Cityscapes and CamVid datasets will be used in this project for the training and testing of the created architectures for semanitc segmentation.The Cityscapes dataset [7] contains videos collected from 50 different cities. More than 25,000 images of 1024×2048 resolution were selected from the frames of the recorded videos to create an image dataset of versatile backgrounds and objects. 5000 images were finely annotated, while the remaining 20,000 images were coarsely annotated. The dataset uses 30 classes. The data were collected over several months and during different weather conditions (i.e., summer, fall, and spring days). Further, the CamVid image dataset [8] is among the widely used datasets in the area of semantic segmentation for autonomous driving. The CamVid dataset consists of five video sequences of road scenes that are divided into 701 images of 960×720 resolution. The dataset uses 32 classes.

2.2.2 Evaluation Metrics and Tools

Several evaluation metrics will be used for thoroughly analysing the performance of the reproduced benchmark deep learning architectures (i.e., DeepLabV3+ and UNET). The evaluation metrics will allow the quantitative comparison between the reproduced and the original architectures' performances. The metrics and tools will include the following:

1. Accuracy
2. Recall
3. Precision
4. Confusion Matrix: Confusion matrix will be used to make detailed comparisons between the number of misclassified instances between one class and another. This will allow us to understand if the network is confused over particular classes.
5. Boundary F1 Score (BFScore): BFscore is used for contour matching evaluation between the network output and the ground truth images.
6. Intersection over Union (IoU)/ Jaccard Index: IoU is used for evaluating the overlap between the ground truth and predicted output masks.

2.2.3 Work Plan

These are the general steps that will be used for both of the approaches we will propose.

1. Preprocessing
   (a) Download the dataset from their respective sites and create generators to load them lazily for training.

(b) Create utility functions to apply the masks, which are labels of each pixels of the image, to images for visualization.

(c) Shuffle the data and split it into train, validation and test at 70%, 10%, 20% ratio.

(d) Analyze the training data for class distribution and see if there are class imbalance problems.

(e) Downsample the image size, if necessary, to reduce their size for quicker training.

(f) Implement data augmentation and see if it gives better results.

(g) Merge certain classes if they are compatible and if it is necessary to improve model performance.

2. Building models
   Iterate and build models on Jupyter Labs on crane with small subset of the data for sanity check. In the first iteration, the goal will be to recreate the reference architecture from the papers.

3. Training model
   Create jobs to submit model for training on crane. This will be done a few times with a few sets of hyper parameters. Every model trained will be saved so they can be loaded later.

4. Evaluation
   Evaluate the models using metrics listed in subsection 2.2.2.

Since building good models is an iterative process in general, we will go through these steps multiple times throughout the project. For example, we might discover that training on the whole dataset at current resolution takes too much time even with HCC and want to downsample the images.

2.3 Proposed Method 1: DeepLabV3+

The DeepLabV3+ [36] is an encoder-decoder architecture that is proposed to be used for implementing the semantic segmentation task on the autonomous driving datasests. Figure 2.2 shows the DeepLabV3+ architecture proposed by Chen et al.[36]. The encoder unit of the DeepLabV3+ network uses a pre-trained deep convolutional neural networks (DCNN) (i.e., Resnet 18, Resnet 101, etc.) as a dense feature extractors [37]. The pre-trained DCNNs are known as backbone networks.The pre-trained DCNN generates low-level feature map, which contains high semantic information. Moreover, the encoder unit employs an atrous spatial pyramid pooling (ASPP), which applies several parallel atrous convolution with different rates to exploit information at different grid scales. Atrous convolution is also known as dilated convolution, where the weights of the atrous convolution's kernel are spaced by rate "r", as shown in Figure 2.3. The dilated kernel allows the capturing of bigger field of view with same number

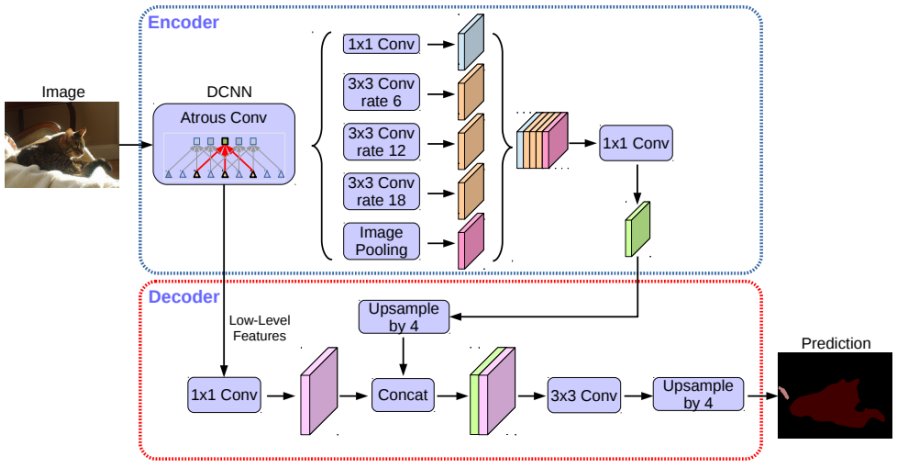

Figure 2.2: DeepLabV3+ architecture (source: [36]).

of weights as a normal kernel without dilation, which improves the robustness of the pixel classification at no additional training expenses. The use of the ASPP showed promising results on the semantic segmentation benchmark datasets [36]. Figure 2.2 shows the details of the ASPP unit. Then, 1×1 convolutional layers are applied to the outputs of the ASPP unit and the low-level feature maps to reduce the number of channels for an efficient training. Before feeding the feature maps produced by the encoder into the the decoder unit, the feature maps are bi-linearly upsampled by a factor of 4. The decoder unit recovers the feature maps and the boundary information by concatenating the feature maps from the encoder unit with the corresponding low-level feature maps from the pre-trained DCNN, as shown in Figure 2.2. Then, two 3×3 convolutions are applied to refine the concatenated feature maps for sharper segmentation results. At the end, a bi-linear upsampling of factor 4 is applied to the output of the 3×3 convolution blocks. The decoder unit helps with recovery of the boundary information in semantic segmentation for better results.
In literature, there were several implementations of DeepLabV3+ networks using different backbone networks such as, Resnet18, Resnet 50, and Resnet 101 [3, 16]. Furthermore, a DeepLabV3+ network was created based on Resnet 18 and was trained on the CamVid dataset for the semantic segmentation task [8]. The trained network was able to get a global test accuracy of 87.65% [16]. Therefore, it is promising to implement a DeepLabV3+ network based on Resnet 18 to tackle the semantic segmentation task for autonomous driving in our project.

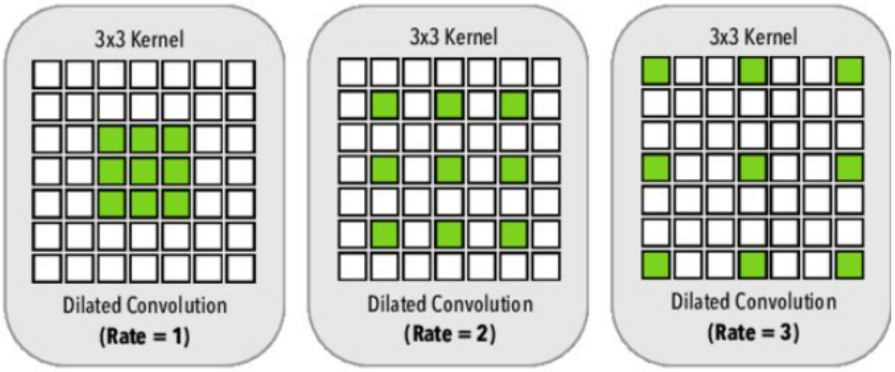

Figure 2.3: Atrous convolution of different dilation rates (source: [38]).

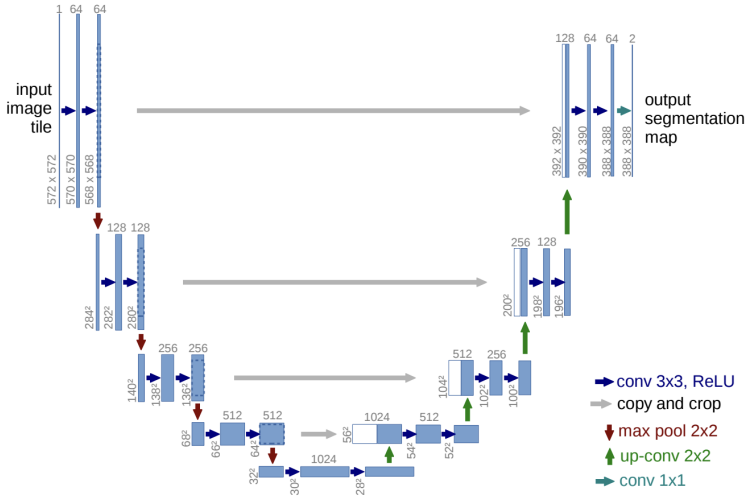

Figure 2.4: UNET architecture.(source: [39]).

2.4 Proposed Method 2: U-NET

U-Net was proposed by Ronneberger et al. [39] for Biomedical Image segmentation. Convolution networks, when used for classification tasks include max-pooling layers that provide translation invariance and help it capture global context. But this comes at an expense of localization accuracy which is necessary for semantic segmentation where each pixel has to be classified.

Figure 2.4 shows the architecture of U-NET. It is an autoencoder architecture with contracting part and expansive part. The contracting part has blocks that apply two 3×3 convolutions with ReLU activation followed by 2×2 max pooling operation for down sampling. Each block doubles the number of features and halves the number of channel. In the contracting part, the network learns the global context information but losses the localization information that is necessary for labeling every pixel [40].

The expansive path has blocks that double the size of input and halves the number of channels. The input to each block is the concatenation of the output of the previous block, which is upsampled using a transpose convolution, and the feature map output of the corresponding block in the contracting path. This is shown by the grey lines in Figure 2.4. The concatenation is followed by two 3×3 convolutions with ReLU activation that help the model learn a more precise output [40]. At the end, a 1×1 convolution is applied that maps the input to the desired classes at the output. Passing the output of contracting part to expansive part helps recover the localization information for the image.

[3] uses U-Net architecture for semantic segmentation on Cityscapes dataset [7]. But instead of using concatenation, they use addition to reduce to the overall computational complexity.

2.4.1 Risks and Checks

The following are risks associated:

1. Enough documentation might not be available to replicate the results of the papers.

2. The training duration might be too large even with HCC to build models described in the papers.

The following are the checks we need to put in place.

1. Mid-term check: Data preprocessing and setup and verification of models described in the literature.

2. Final check: Comparing the model performance with those published in the literature.

2.5 Conclusions

To conclude, among various applications of semantic segmentation, our project focuses on autonomous driving. In this report, two benchmark deep-learning based architectures were proposed and discussed (i.e., DeepLabV3+ and U-Net). Both of these architectures are encoder-decoder architectures, but with different implementation. Further, DeepLabV3+ architecture is more recent than U-Net. At first, we will reproduce the original architectures. Then, for training, testing and validation, we will be using two benchmark datasets, which are the Cityscapes and CamVid datasets. Also, we will compare the results from our architecture with those of the original reference architectures as discussed earlier in this report. This project will be helpful for us to learn about and reproduce benchmark architectures as per the literature.

We would like to be clear on the following questions before moving further

1. Are we overestimating or underestimating the problem and the timeline?
2. What happens if we are unable to meet the goal as stated in this report?
3. Is there any fundamental problem in any of the steps/approach discussed in this report?

Chapter 3

Milestone 3: Progress Report 3

3.1 Introduction

Semantic segmentation, which aims to assign dense labels for all pixels in an image, is a fundamental task in computer vision [3]. It has a number of potential applications in the fields of autonomous driving, video surveillance, robot sensing and etc. For such applications, keeping efficient inference speed and high accuracy with high resolution images is critical [3, 4, 5].
Deep learning models have been proven to be highly effective in performing semantic segmentation on autonomous driving [3]. For milestone 3, it was decided to perform semantic segmentation on the CamVid dataset. While doing the comparative literature [41] review for U-net and DeeplabV3+, it was found that DeeplabV3+ has better performance results and so it was decided to consider DeeplabV3+ for implementation. In the current milestone, DeeplabV3+ with CamVid dataset was implemented and it resulted in a MeanIoU (Mean Intersection over Union) of 69.3% on the test dataset.

3.2 Experimental Setup

3.2.1 Dataset

The Cambridge-driving labeled video database (CamVid) [42] is the first collection of videos with object class semantic labels. The database provides ground truth labels that associate each pixel with one of 32 semantic classes. It was captured in Cambridge, UK using a camera mounted inside the car. The filming was done in various environments like dusk having significantly darker and

grainier image as well as in daylight with urban and residential condition. The high definition resolution of these videos helps in detecting small objects, such as traffic lights, car lights, road markings, and distant traffic signs. The recorded video frames were decomposed into 701 densely annotated images with resolution size 960×720.

3.2.2 Data analysis

While analysing CamVid dataset [42], it was observed that only 2.68% of the total pixels were labeled as void (i.e., not assigned to a class). This low rate is an indicator of consistent labeling and also shows that the choice of the 32 class labels was adequate for this problem. Since the images were captured in an urban environment, classes such as road, building and sky constitute the largest weight (i.e., between 15.81% and 27.35% of the total number of the pixels of the dataset) and they are present in virtually all the frames. On contrary, classes such as car and pedestrian, which are crucial entities in our context are very frequent (high occurrence value) but constitute small class weights of 3.97% and 0.64%, respectively. Other entities such as child and road-shoulder, which are also important constitute very small weights. The class weight was calculated as the number of pixels in the class divided by the total number of pixels in the dataset. From a statistical point of view, Figure 3.1 shows the first 11 classes from the priority and weighted percentage perspective. Those classes were selected for training and testing the models in this milestone, while the rest of the 32 classes were assigned to an additional void class, as those rare classes. These classes and the associated approach were also being considered by other literature [42] and so it was also being followed in this work. In the future milestones, we will examine the model training behaviour while increasing the number of classes or grouping some of rare classes in these 11 classes.

3.2.3 Preprocessing

In this step, the data was preprocessed and saved in the disk to avoid rerunning the preprocessing on the dataset while training the models. The size of the data was large. Therefore, when we tried to to download the data, it took long time and the kernel killed the process after sometime. So, the data was directly downloaded on the crane and saved on the disk to avoid this. The dataset had two parts: one with the original images and the other part with the corresponding masks same size as the original image. Each pixel in the mask was given a class label and corresponding RGB color code. The two parts in the dataset were not present in the same order. So, we execute a function to map them. The original images and the masks were then converted into a *numpy* array and then were resized. Then, the *numpy* arrays for the images were normalized by dividing each pixel value by 255 as it is an 8 bit image and the data has its integer values in the range of 0-255. As mentioned in Section 3.2.2, only 11 classes were selected as per the priority and weighted percentage attributes, and all the remaining classes were assigned to an additional void

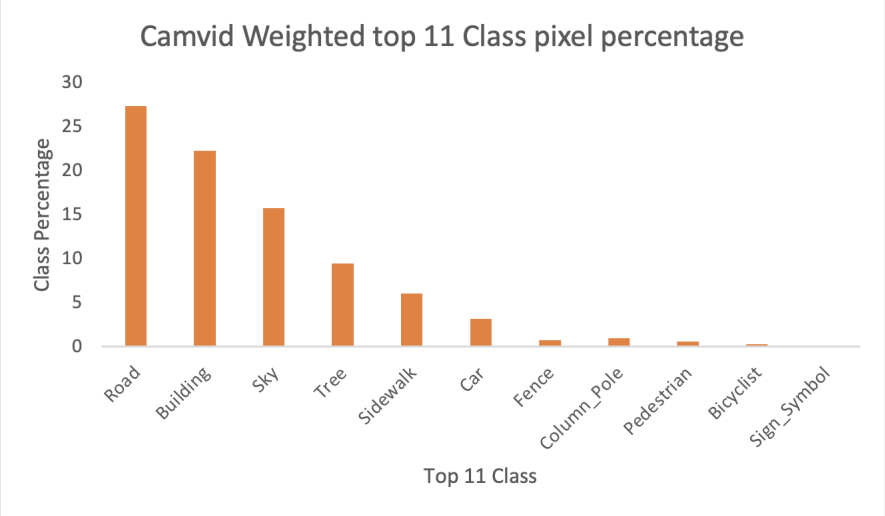

Figure 3.1: Camvid top 11 class percentage

class. The RGB color code in the masks representing these 12 classes were then mapped back to the corresponding classes, and then were further mapped to integers as shown in Figure 5.2.

Class	Color label	Integer label
Sky		0
Building		1
Column Pole		2
Road		3
Sidewalk		4
Tree		5
Sign		6
Fence		7
Car		8
Pedestrian		9
Bicyclist		10
Void		11

Figure 3.2: Mapping class to color and integer label

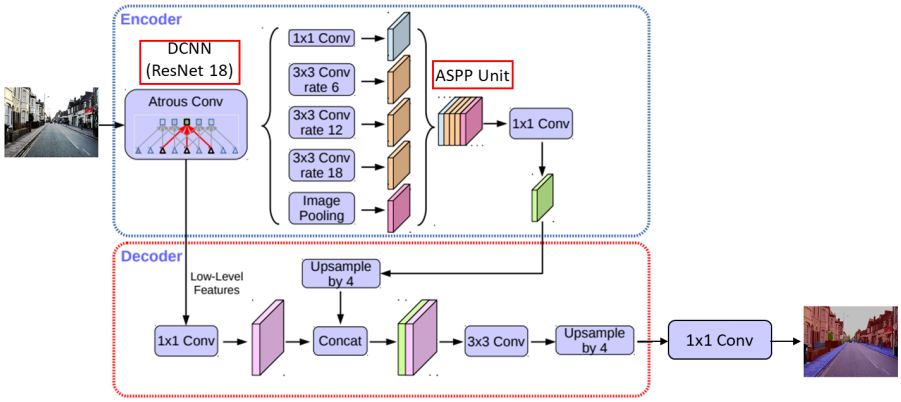

Figure 3.3: DeepLabV3+ architecture (adapted from the source: [36]).

3.2.4 Architecture

A DeepLabV3+ architecture incorporating the pre-trained ResNet 18 network as a backbone and feature extractor [3, 16] was built to accomplish the semantic segmentation task on CamVid dataset. The DeepLabV3+ network is an encoder-decoder architecture and one of the benchmark models used for implementing the semantic segmentation task on the autonomous driving datasets (i.e., CamVid, and KITTI datasets) [3, 43]. Figure 5.3 shows the DeepLabV3+ architecture used in this project milestone. The encoder unit of the DeepLabV3+ network uses a deep convolutional neural network (DCNN) such as using ResNet 18 or ResNet 101, etc. as a dense feature extractor [37]. The DCNN is known as the backbone network and it generates a low-level feature map, which contains high semantic information. Figure 5.4 shows the ResNet 18 architecture used as the DCNN for the DeepLabV3+ network. Moreover, the encoder unit employs an atrous spatial pyramid pooling (ASPP), which applies four parallel atrous convolutions with different rates to exploit information at different grid scales along with an image pooling layer. Atrous convolution is also known as dilated convolution, where the weights of the atrous convolution's kernel are spaced by rate r. The dilated kernel allows the capturing of bigger field of view with the same number of weights as a normal kernel without dilation. Figure 5.3 shows the details of the ASPP unit. Then, 1×1 convolutional layers were applied to the outputs of the ASPP unit and the low-level feature maps to reduce the number of channels for an efficient training. Before feeding the feature maps produced by the encoder into the the decoder unit, the feature maps were bi-linearly upsampled by a factor of 4. The decoder unit recovers the feature maps and the boundary information by concatenating the feature maps from the encoder unit with the corresponding low-level feature maps from the DCNN, as shown in Figure 5.3. Then, two 3×3 convolution blocks were applied to refine the concatenated feature maps for sharper segmen-

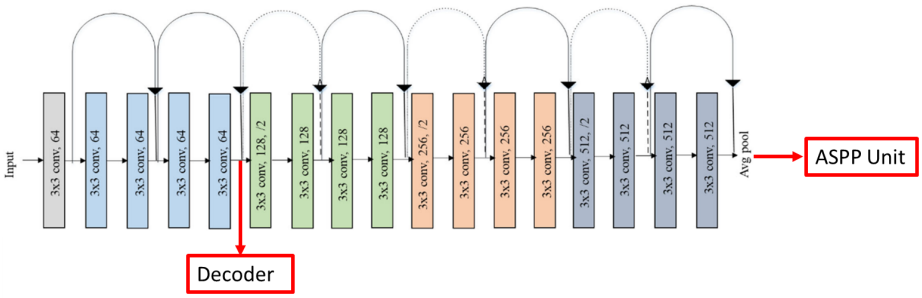

Figure 3.4: The ResNet 18 architecture which was utilized as the DCNN for the DeepLabV3+ architecture (adapted from the source: [44]).

tation results. At the end, a bi-linear upsampling of factor 4 were applied to the output of the 3×3 convolution blocks. The decoder unit helps with the recovery of the boundary information in semantic segmentation for better results.

In literature, there were several implementations of DeepLabV3+ networks using different backbone networks such as, ResNet18, ResNet 50, and ResNet 101 [3, 16]. However, for this project a DeepLabV3+ network based on a ResNet 18 pre-trained network was utilized following Mathwork's [16] DeeplabV3+ implementation on the CamVid dataset [8]. To utilize the ResNet 18 network in DeepLabV3+ (i.e., as depicted in Figure 5.3), the original fully connected and softmax layers of ResNet 18 [44] were removed. Then, the output of the last $3 \times 3 \times 64$ convolution block was passed to the decoder block, while the output of the last $3 \times 3 \times 512$ convolution block was passed to the ASPP unit, as shown in Figure 5.4. For a pixel-wise classification, the output of the last 3×3 block in the decoder unit was passed through a convolution 2D block. The number of filters was equal to the number of classes, the height and the width of the convolution layers were matching those of the input image, as shown in Figure 5.3. The final convolution 2D block employed the softmax activation function. The Adam optimizer and the cross entropy loss function were used for training the network. The created-from-scratch DeepLabV3+ network had a total of 16.6 million trainable parameters. The pre-trained ResNet 18 network on the Imagenet dataset [44] was loaded from the Keras library and prepared as mentioned above, while the ASPP and decoder units were handcrafted from scratch using the TensorFlow and Keras libraries and APIs.

3.2.5 Training

The training and testing were conducted in the Jupyter lab on Crane. The configuration used had 8 cores CPU, 32 GB RAM and 12 GB GPU. Training the model without a GPU was too slow, taking about half an hour per epoch. Initially, we tried to use the GPU provided by Crane by default. But we were unable to load the model in the memory while using it. We reduced the batch

size from 32 to 4, but it did not solve the issue consistently. The model would sometimes start training but fail at other times. Ultimately, we found out that we could request GPU with higher memory and that solved the memory issue.

The preprocessed data was split into training and test datasets with 80:20 split. These were both saved in the disk. This was done to avoid re-running the preprocessing on the dataset while developing the models. Two versions of dataset were created, one with a resolution of 256×256 and the other with resolution of 512×512. This was done to ensure that the models could be trained without memory issues and as well as to reduce training time. Models were trained and saved for both versions. During training, 80% of the training dataset was used for training and 20% was used for validation. Early stopping was implemented to stop training if there was no improvement in validation loss for 10 epochs. Regarding the rest of the hyperparameters, they were all chosen to be matching the Mathwork's [16] implementation, which was followed for this project task. As per Mathwork [16], the learning rate chosen was 0.001, and the activation functions were all chosen to be Relu.

3.2.6 Testing

The test dataset and models were loaded from the disk. The class labels for all images in the test set were evaluated using the model and accuracy, confusion matrix, mean intersection over union (MeanIoU) were computed. The classification accuracies and confusion matrices were calculated by considering each pixel in all the images as a separate data point.

3.3 Experimental Results

Figure 3.5 shows one of the outputs of the 512×512 model. The accuracy over this image was 93.32%. The color code for the pixel predicted and the true labels are as per Figure 5.2.

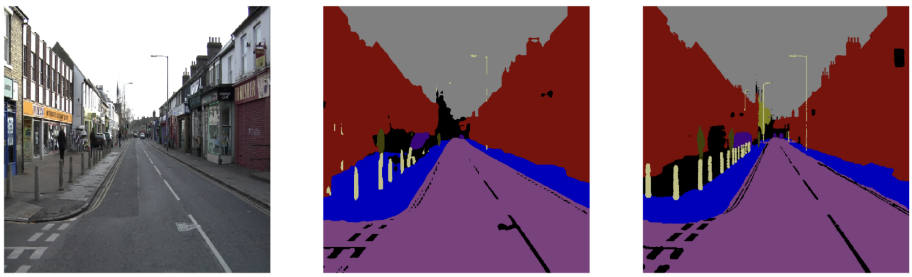

Figure 3.5: Sample output of the 512×512 model on an image in the test set. The three images are the original image (left), image masked with predicted labels (center) and image masked with the true labels (right)

Table 3.1: Performance metrics of the models

Model	Accuracy	MeanIoU	Precission	Recall	F1-score
256×256 resolution	0.9018	0.644	0.90	0.90	0.90
512×512 resolution	0.9159	0.693	0.91	0.92	0.91

Figure 3.6 and figure 3.7 show the learning curves for the 256×256 model and 512×512 model respectively. Table 3.1 shows performance metrics for both models. Precision, recall and F1-score listed are the weighted averages of respective values across all the classes. Average time per epoch was 9 seconds for 256×256 epoch and 27 seconds for 512×512 with the above listed hardware configuration. Figure 3.8 and figure 3.9 show confusion matrices for the two models. The values are normalized across true labels.

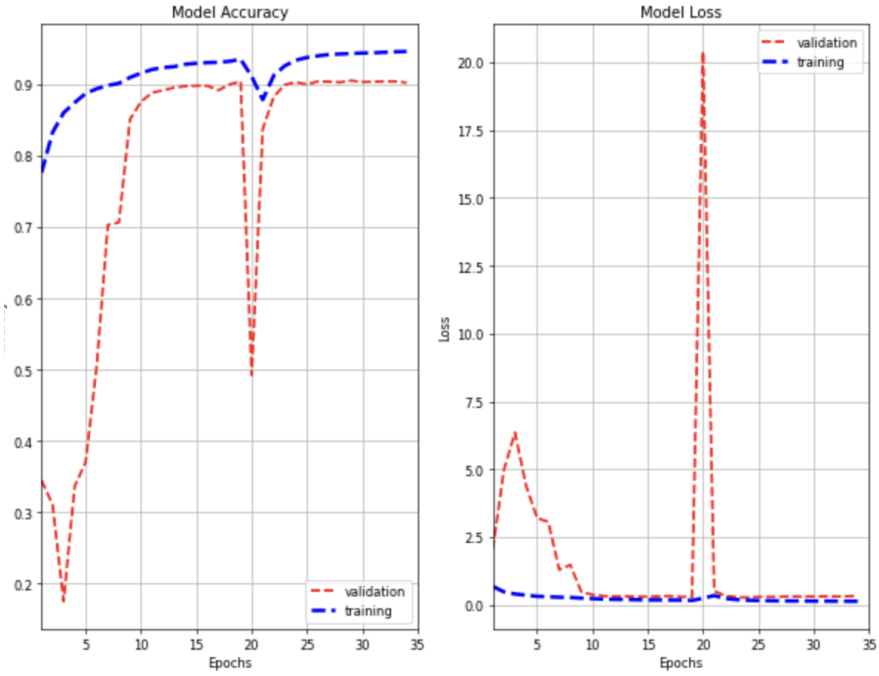

Figure 3.6: Learning curve for the 256×256 model

3.4 Discussion

3.4.1 Training Results

As discussed earlier, we had two image sets: one with the resolution of 256×256 and the other with the resolution of 512×512. While looking at the learning

curves of both models, as shown in Figures 3.7 and 3.6, we see that the validation and training accuracies were only slightly lower for model the 256×256 model as compared to the 512×512. Further, the training of the 512×512 model required slightly less number of epochs as compared to the 256×256 model. But the downsampling of the image resolution from 512×512 to 256×256 resulted in a significant reduction in the time required for the completion of one epoch by about one third. Although at this stage, we cannot conclusively claim that the lower resolution model can produce equally good training results as the higher resolution model at less amount of time. However, this will be of our interest to study the effect of lower resolutions on the training accuracy and the duration, and the model performance. Also, a dip in both the training and validation learning curves for both the 256×256 and 512×512 models was noticed, and the dip is deeper in validation compared to the training. We will investigate further into this during our next milestone as we could not reach to any conclusions regarding this in this milestone.

3.4.2 Test Results

At this stage, the test results of our models are better by 3-4% as compared to the test results of the original implementation of Mathwork [16] as shown in

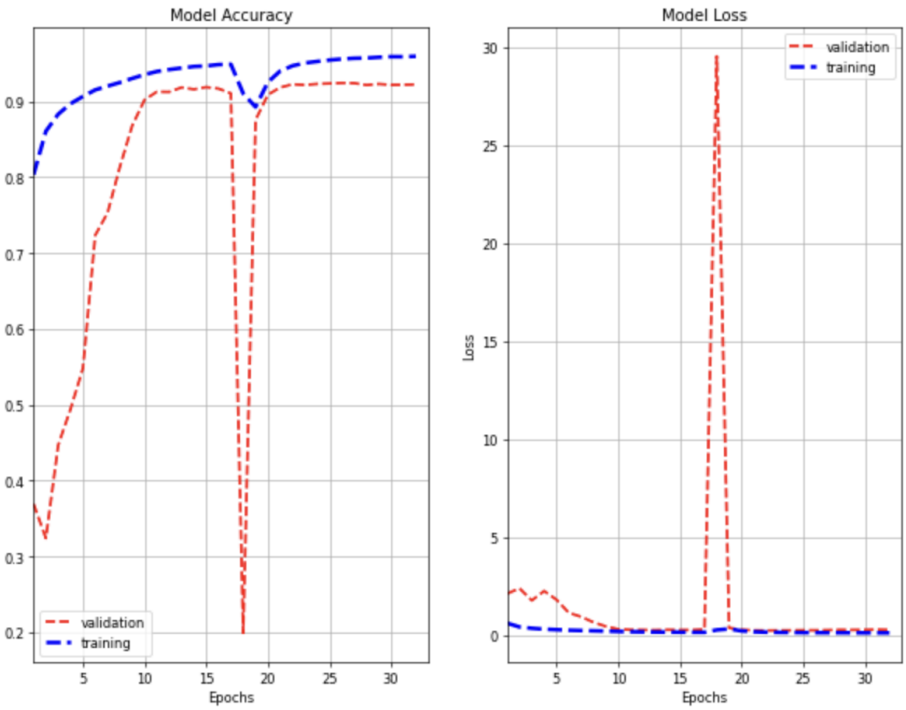

Figure 3.7: Learning curve for the 512×512 model

3.2. However, this is not a conclusive argument, and we ought to investigate further into this using different hyperparameters (i.e., image resolution, data augmentation). Similar to the training results, the test results for the higher resolution images (i.e., 512×512) were slightly better as compared to the results for the lower resolution images (i.e., 256×256). Although the global accuracies of both models were high, the confusion matrices, as shown in Figures 3.8 and 3.9, we can see that the accuracy for some of the classes were very low. For the 256×256 model, the accuracies for the classes column pole, sign, and pedestrian were 0.14, 0.28 and 0.47, respectively. Similarly, for the 512×512 model, the accuracies of the classes column pole, sign, and pedestrian were 0.39, 0.27, and 0.55, respectively. Despite these low class accuracies, the models were performing good with global accuracies of 0.90 and 0.91. This was because the

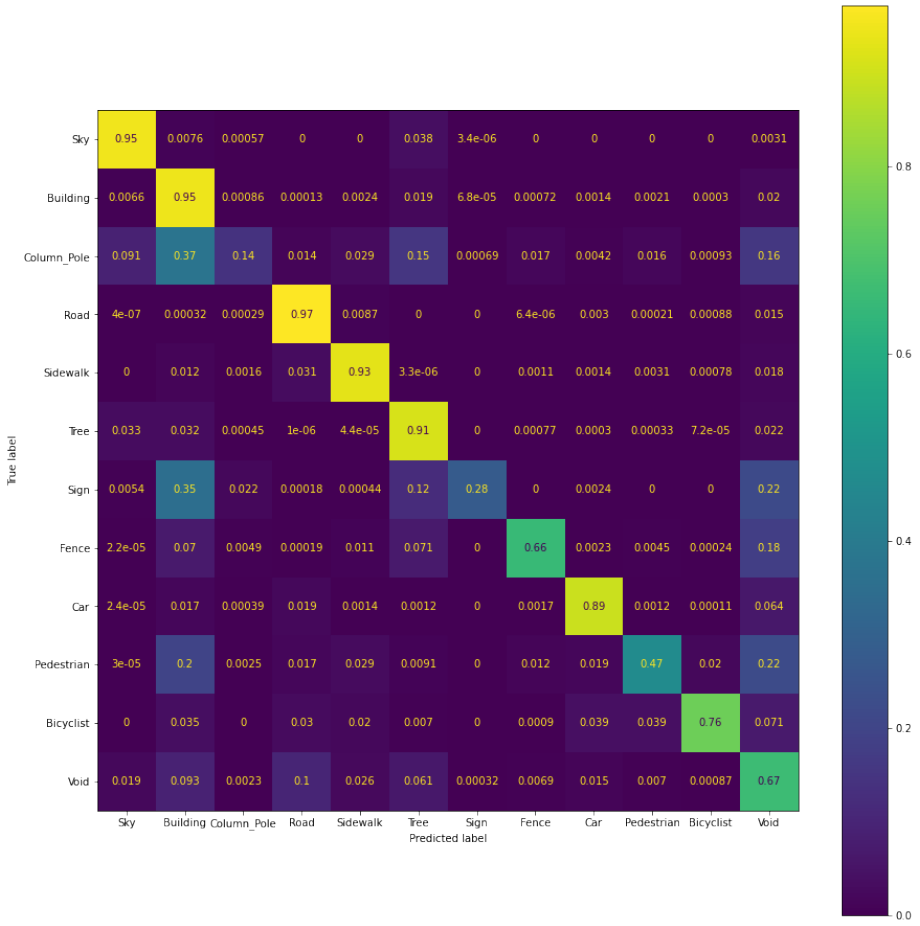

Figure 3.8: Confusion matrix for the 256×256 model

networks were predicting the classes with larger weights in the dataset (i.e., sky, and building) with higher accuracies, as shown Figures 3.8 and 3.9, as compared to classes with smaller weights in the dataset (i.e., column pole, sign, and pedestrians). Therefore, one of our future tasks will be tackling the class imbalance issues in the CamVid dataset.

3.4.3 Future Plan

As shown in the training results, several hyperparameters need to be further studied for a thorough analysis of the created DeepLabV3+ model performance and improvement of the overall network performance. The following items are proposed for a further investigation in the coming milestones:

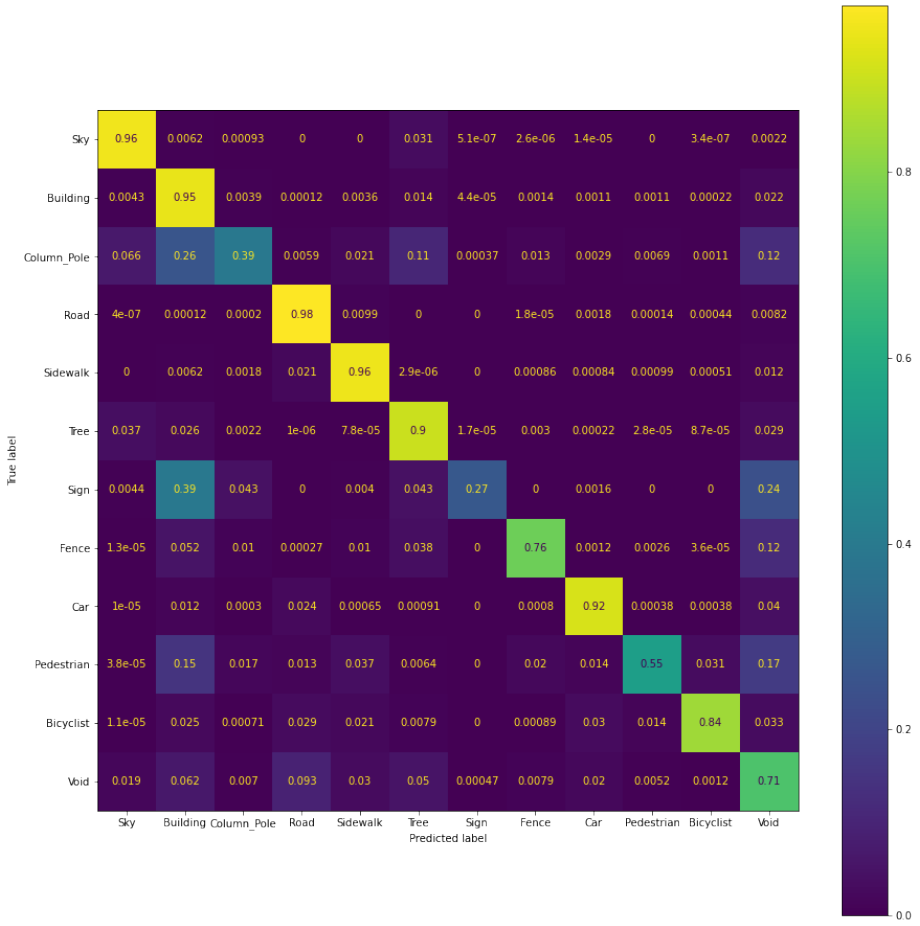

Figure 3.9: Confusion matrix for the the 512×512 model

Table 3.2: Performance metrics of the Mathwork model [16]

Image Resolution	**Accuracy**	**Mean IoU**	**Precission**	**Recall**	**F1-score**
720×960	0.87695	0.85392	0.6302	0.880851	0.65051

1. Consider the effect of lower image resolutions (i.e., 128×128, 64×64, and 32×32) on the training duration and accuracy, and the overall network performance.
2. Test the effect of the data augmentation techniques on the network performance and training accuracy.
3. Tackle the class imbalance issues in the CamVid datasets.
4. Consider using different pre-trained DCNNs (i.e., ResNet 50) as dense feature extractors in the DeepLabV3+ architecture.
5. Study further the huge dip occurred in the validation learning curves of both models trained in this milestone.

3.5 Conclusion

In this milestone, a DeepLabV3+ based on the pre-trained ResNet 18 DCNN was built, trained and tested on the CamVid dataset. Two different image resolutions were considered. The results showed that our implementation is slightly better than the original implementation that was followed in this milestone but this needs to be studied further. Also, several issues with the data and model training were encountered in this milestone and need to be investigated deeply in the future milestones. Also, the results showed that reducing the image resolution by from 512×512 to 256×256 reduced the training duration by more than 30%. Hence, the effect of image resolution on the training accuracy and duration will be heavily studied in the future milestones.

Chapter 4

Milestone 4: Progress Report 2

4.1 Introduction

In Milestone 3, a DeepLabV3+ model using ResNet 18 as a backbone network was trained on the CamVid dataset [42] with two different image resolutions (i.e., 512×512 and 256×256). The results showed a mean IoU (mean intersection over union) of 69.3% on the test dataset. Also, the results revealed that the CamVid dataset suffers from an inherent class imbalance issue that needs to be addressed for a better model performance. Hence, in Milestone 4, we decided to apply several data augmentation techniques to tackle the class imbalance issues. In addition, DeepLabV3+ models using ResNet 18 will be trained and tested on different image resolutions for a comparison of the training accuracy and duration. To study the effect of the backbone network on the model performance, a DeepLabV3+ model based on ResNet 50 was built and trained on the CamVid dataset for comparison with the ResNet 18-based model. Furthermore, a framework for downloading, loading and preprocessing the Citscapes dataset [7] was created in this milestone. DeepLabV3+ models will be trained and tested on different image resolutions of the Cityscapes dataset in the next milestone.
The results in this milestone showed that the ResNet 50 based model had nearly one percentage improvement in the global accuracy and mean IoU, as compared to the ResNet 18 based models. Also, data augmentation was able to improve the mis-classification problems between classes by nearly 3–4%.

4.2 Experimental Setup

4.2.1 Dataset

The Cityscapes dataset is recorded from fifty different cities to be used for autonomous driving in an urban setting [7]. It consists of various highly complex

inner-city street scenes. In comparison to CamVid dataset [42], which only has 701 images, Cityscapes has 5,000 finely annotated images. Several hundreds of thousands of frames were acquired using a camera mounted on a moving vehicle for several months. These frames have a variety of complex environmental and urban driving behaviour. They include data from Spring, Summer, and Fall seasons from fifty different cities, primarily in Germany but also in neighboring countries. Although it does cover most of the weather conditions throughout the year, its does not cover adverse weather conditions such as heavy rain or snow . The recorded images were of a resolution size of 1024×2048. In addition to the finely annotated images, the Cityscapes dataset also contains 20K coarsely annotated images. Out of fifty cities, the recordings of twenty seven cities were used to generate the finely annotated images, and the remaining recordings were used to create coarsely annotated images. The dataset consists of 30 pixel-label classes, which are further grouped into eight categories (i.e., flat, construction, nature, vehicle, sky, object, human, and void).

4.2.2 Data analysis

A statistical view of the Cityscapes dataset class labels and their associated percentage is showcased in Figure 4.1 [7]. In the graph, the x-axis represents all the labelled classes grouped into eight categories, and the y-axis represents the percentage of pixels for each class. Out of 30, 19 classes were selected for model training and testing. This selection was based on their class frequency and semantic meaning. These 19 classes are shown in Figure 4.2. The selected classes are represented in green, while all the ignored classes were regrouped as a void class. The ignored classes are shown in orange in Figure 4.2. The void class accounted for 7.04% of the total number of pixels. The data analysis task was implemented on the training dataset.

In addition of being larger dataset, the Cityscapes dataset also has a more variety in the driving environment. The image resolution of Cityscapes i 1024×2048, which is bigger than CamVid's image resolution (i.e., 720×960). In terms of number of classes and associated class percentage, CamVid and Cityscapes are similar. The only exception is for the sky class. Due to cameras placed with a comparably large vertical field-of-view, CamVid has an abundance of sky pixels, as compared to Cityscapes. Figure 4.3 shows a comparison between the classes of the CamVid and Cityscapes datasets.

4.2.3 Preprocessing

We decided to use the 5,000 finely annotated images for semantic segmentation. The Cityscapes dataset provided us with some of the challenges that we had not faced in the previous milestone. The Cityscapes dataset in the tensorflow-datasets is only available for non-commercial use and redistribution is not allowed [45]. So, we downloaded the raw dataset manually. We had to create an account and sign into the Cityscapes dataset website. Therefore, the dataset could not be downloaded directly into the HCC server, and we had to

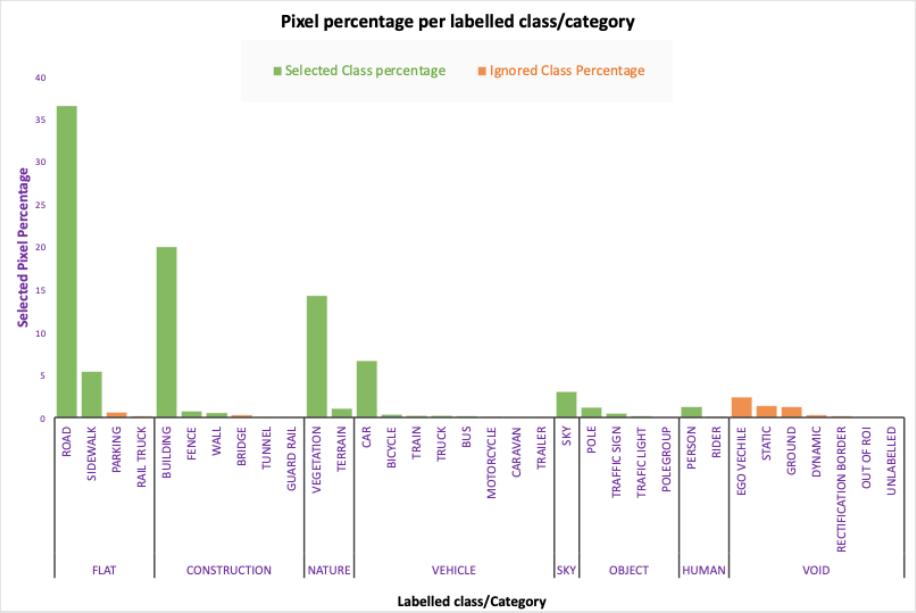

Figure 4.1: Percentage of annotated pixels (y-axis) per class and their associated categories (x-axis)

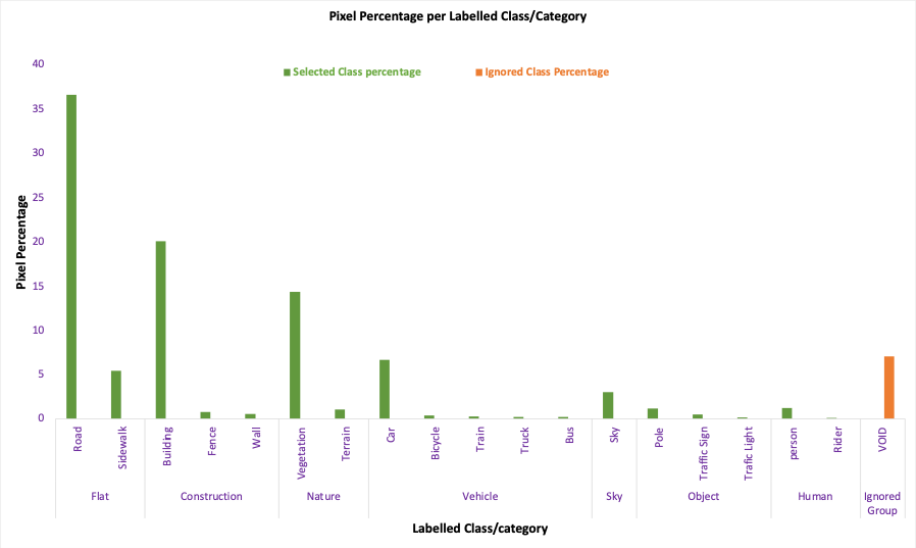

Figure 4.2: Percentage of pixels (y-axis) per class and their associated categories (x-axis)

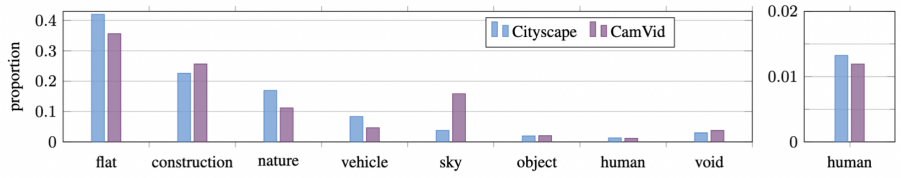

Figure 4.3: Comparison between the classes of the Cityscapes and CamVid datasets (adapted from the source: [7])

download the 11 GB of data locally. Further, HCC did not allow us to upload the zipped file directly due to the large file size. So, we used scp, which is a tool that uses the SSH protocol to transfer the data to the remote server. After providing it with the zipped file of the Cityscapes dataset, the tensorflow-dataset library was used to process the data feed the data generator with images and labels.

We created a generator that takes a batch of data, which was provided by the tesnor-flow datasets, resizes the images and labels to the desired resolution, and converts labels into classes. The generator returns images and the masks. A generator that can handle a portion of the data at a time was necessary because it was not possible to load the whole dataset into the memory at once. Although running image resizing and label assignment every time is inefficient, the simplicity and flexibility of this approach outweighed by far the slight extra overhead the generator added during the training time. The images and labels were resized using the OpenCV library [46]. The labels were converted into classes by selecting 19 of the labels and treating everything else as a single class named as void. This was done with the utility framework provided by the authors of the Cityscapes dataset [7].

We ran into an issue with the test partition of the Cityscapes dataset. The masks were not correctly set and we could only see the labels of the void class, based on manually checking few masks from the test dataset. We saw a resized dataset of Cityscapes in Kaggle [47] but it also did not have the properly annotated test dataset. So, we are assuming that there is an issue with the test dataset. We plan to use the validation dataset as a test dataset, and split the training dataset into training and validation datasets. We also had tried using the dataset available in Kaggle but the labels had different pixels and most of the implementations on Kaggle used K-means clustering to derive the actual labels. It also did not not allow us to experiment different resolutions.

4.2.4 Architecture

In this milestone, the DeepLabV3+ model with the backbone network of ResNet 18, which was developed in the previous milestone (see Section 3.2.4), was used again. Also, a new DeepLabV3+ model using the pre-trained ResNet 50 [48] as a backbone network was built for a further comparison with the performance of

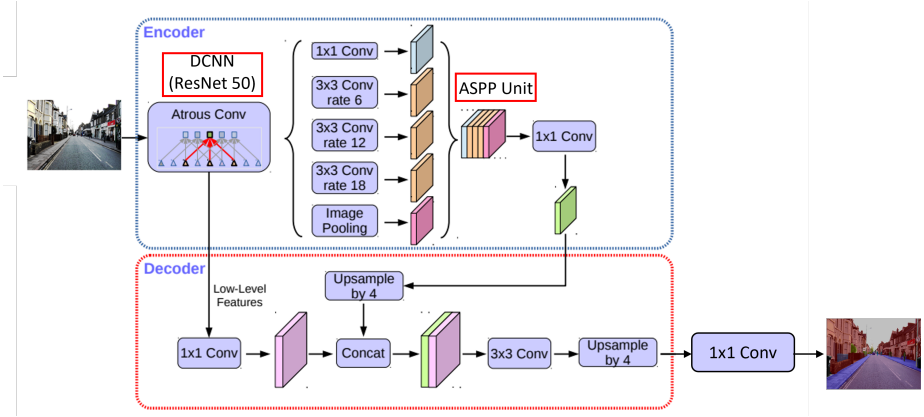

Figure 4.4: DeepLabV3+ architecture based on ResNet 50 (adapted from the source: [36]).

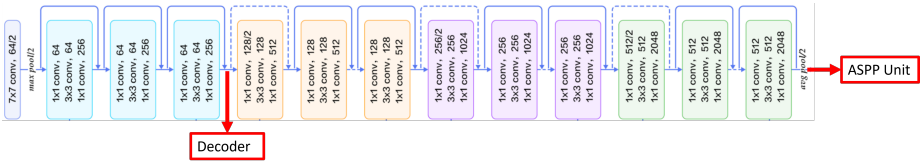

Figure 4.5: ResNet 50 architecture (adapted from the source: [49]).

the DeepLabV3+ models built based on ResNet 18. ResNet 50 was utilized as a stronger dense extractor than ResNet 18. Figure 5.5 shows the architecture of ResNet 50, where it has double the number of residual blocks of ResNet 18 (see Figure 5.4). Unlike ResNet 18, ResNet 50 uses bottleneck residual blocks (i.e., 3-layer blocks) [48]. The bottleneck residual blocks use 1×1 convolutions to reduce the number of parameters and matrix multiplications, as shown in Figure 5.5. This technique is essential to make residual blocks as thin as possible to increase depth and have less parameters for deeper networks. The ResNet 50 network utilized for the DeepLabV3+ model (see Figure 4.4) was trained on the Imagenet dataset and it was loaded from the Keras library. All the activation functions used were ReLu. The total number of parameters of the DeepLabV3+ model based on ResNet 50 was nearly 40.5 million, while the ResNet 18-based model had nearly 16.6 million parameters. Therefore, the ResNet 50-based models were nearly 2.5 times denser.

4.2.5 Data Augmentation

The OpenCV library [46] in Python was used for implementing three data augmentation techniques on the training partition of the CamVid dataset. The data augmentation framework was applied to both the images and masks (i.e.,

Table 4.1: Parameters experimented in this milestone

Parameters	Type
Image resolution	960×720, 480×360 and 240×180
Feature extractor	ResNet 18 and ResNet 50
Data Augmentation	Data with and without image augmentation

labels of each pixel). The data augmentation framework generated three images and masks for every input image and mask, respectively. The augmented dataset was generated after each of the horizontal, vertical, and combined horizontal and vertical flip techniques individually to the original dataset. The flip techniques were applied in this milestone to prevent the models from overfititng over the training data and tackle the class imbalance issues encountered in the previous milestone.

4.2.6 Training

The experimental setup was similar to that implemented in the previous milestone (see Section 3.2.5). Except in this milestone, we implemented three different image resolutions while preserving the original aspect ratio of the CamVid dataset. Also, two different feature extractors and three different data augmentation techniques were implemented. The different parameters considered are tabulated in Table 5.3. Further, the training and testing were conducted using Jupyter Lab on Crane, and the configuration used had 16 cores CPU, 62 GB RAM and two 32 GB GPUs. This configuration made the training and testing time, for all the different models based on the aforementioned parameters, a lot faster. The experiments conducted in this milestone were for the CamVid dataset only.

Similar to the earlier milestone (see Section 3.2.5), we split the dataset into training and testing with 80:20 split. We created three different versions of the dataset with image resolutions of 960×720, 480×360 and 240×180, and the data were saved on the disk. All the models created for the different combinations of the parameters were trained and saved to ensure that testing could be done without any memory issues. The training dataset was further split into training and validation with 80:20 split. An early stopping criterion with a patience of ten epochs was utilized during training. The Adam optimizer was used for training the models. The learning rate selected was 0.001 as per Mathwork's implementation [16].

4.2.7 Testing

Testing was done in compliance with the framework presented in the previous milestone. The test dataset and models were loaded from the disk. The class labels for all images in the test dataset were predicted using the models. Further, we also computed the global accuracy, confusion matrix and mean intersection

over union (mean IoU) for each of the runs. The classification accuracies and confusion matrices were calculated by considering each pixel in all the images as a separate data point.

4.3 Experimental Results

We have categorized the experiments into the following three sections. The results that are directly relevant to the discussion sections are presented in this section. Additional results (i.e., confusion matrices, classification reports and learning curves) of all the models can be found in the Appendix 5.8.

4.3.1 Comparison of Image Resolutions

Training

The model utilizing ResNet 18 was used to compare the model performance on three different image resolutions — 960×720, 480×360 and 240×180. Table 4.2 shows the time required per epoch, number of epochs for training and the best validation accuracies for different image resolutions.

Table 4.2: The training results for the ResNet 18 models of different image resolutions

Attributes	960×720	480×360	240×180
Time per epoch(seconds)	27	8	3
Epochs	37	31	31
Best validation accuracy	0.923	0.914	0.898

Testing

Table 4.3 shows the global accuracy and mean IoU for models of different image resolutions. Figure 4.6 shows a sample of the predicted masks of different image resolutions. The color code for the pixel predicted and the true labels are as per Figure 5.2.

Table 4.3: The testing results for the ResNet 18 models of different image resolutions

Metric	960×720	480×360	240×180
Global accuracy	0.920	0.908	0.896
mean IoU	0.729	0.685	0.639

Figure 4.6: Predicted mask for image resolutions of (a) 960×720, (b) 480×360, (c) 240×180, and (d) the original image and (e) the ground truth mask

4.3.2 Comparison between ResNet 18 and ResNet 50

Training

The ResNet 18 Model with the highest image resolution (i.e, 960×720) was chosen to compare between the performance of the ResNet 18 and ResNet 50-based models, since it had better global accuracy and mean IoU compa. Table 4.4 shows the time required per epoch, training epochs, the best validation accuracies for the DeepLabV3+ models based on ResNet 18 and ResNet 50 using an image resolution of 960×720. Figure 4.7 and Figure 4.8 show the learning curves for the ResNet 18 and ResNet 50 models, respectively. The rest of the learning curves are in Appendix 5.8.

Table 4.4: The training results of ResNet 18 and ResNet 50 models for an image resolution of 960×720

Attributes	ResNet 18	ResNet 50
Time per Epoch (seconds)	27	43
Epochs	37	32
Best validation accuracy	0.9234	0.9231

Testing

Table 4.5 shows the global accuracy and mean IoU for models with ResNet 18 and ResNet 50 for the image resolution of 960×720. Figure 4.11 shows sample

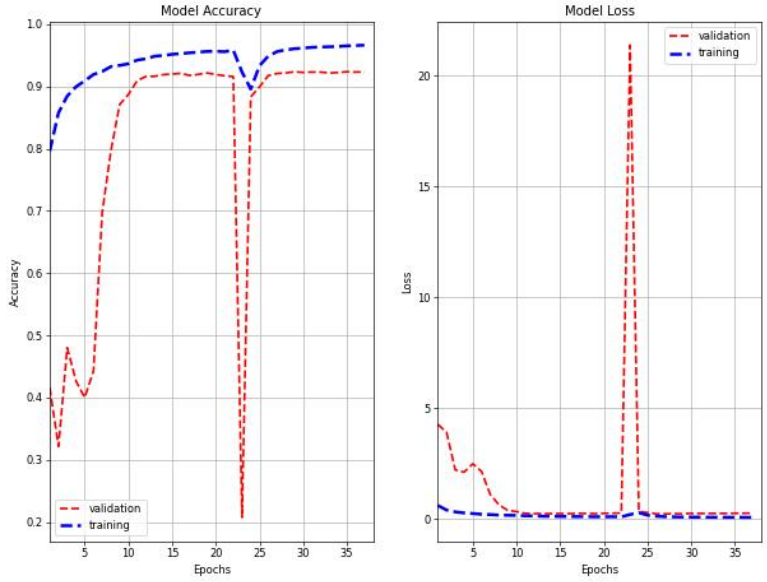

Figure 4.7: Learning curve for model with ResNet 18 and Image resolution of 960×720

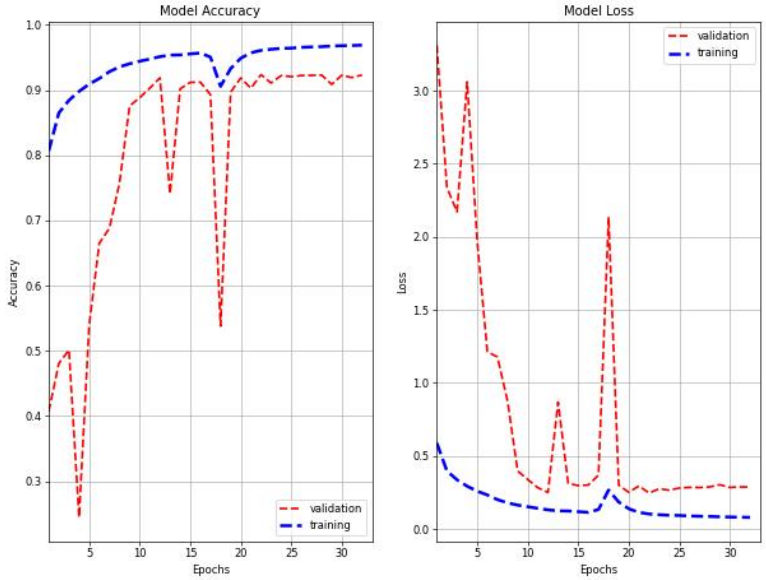

Figure 4.8: Learning curve for model with ResNet 50 and Image resolution of 960×720

of the predicted masks for both models. The color code for the pixel predicted and the true labels are as per Figure 5.2.

Table 4.5: The testing results of ResNet 18 and ResNet 50 models for an image resolution of 960×720

Metrics	ResNet 18	ResNet 50
Global accuracy	0.920	0.922
mean IoU	0.729	0.732

4.3.3 Data Augmentation

Training

Initially, we tried to implement the data augmentation techniques on an image resolution of 960×720, but Jupyter Lab crashed several times during training, probably due to memory issues. So, we tried the image resolution 480×360 and it worked. Table 4.6 shows the time required per epoch, training epochs, and the best validation accuracies for the ResNet 50-based models trained with and without data augmentation.

Table 4.6: The training results for the ResNet 50 models with and without data augmentation for an image resolution of 480×360

Attributes	Without data augmentation	With data augmentation
Time per Epoch(seconds)	14	55
Epochs	27	31
Best validation accuracy	0.912	0.9289

Testing

The ResNet 50-based models that were trained with and without data augmentation were tested on the test dataset. The data augmentation techniques were not applied to the test dataset. Table 4.7 shows the global accuracy and mean IoU for the models trained with and without data augmentation. Figure 4.14 shows sample of the predicted masks. Figures 4.12 and 4.13 show the confusion matrices for model trained with and without data augmentation.

Table 4.7: The testing results for the ResNet 50 models with and without data augmentation for an image resolution of 480×360

Metrics	Without data Augmentation	With data Augmentation
Global accuracy	0.911	0.921
mean IoU	0.694	0.727

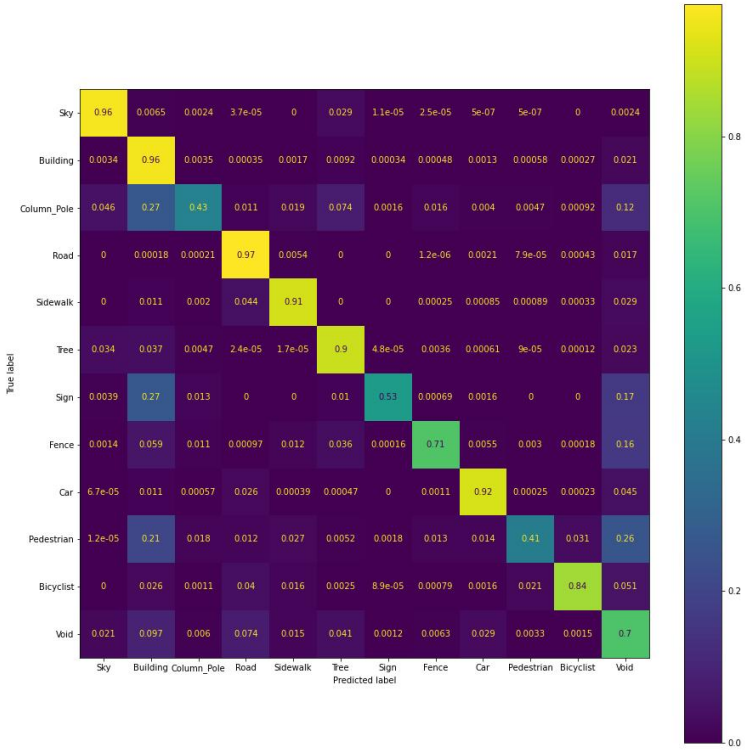

Figure 4.9: Confusion matrix for the model with ResNet 50 and Image resolution of 480×360

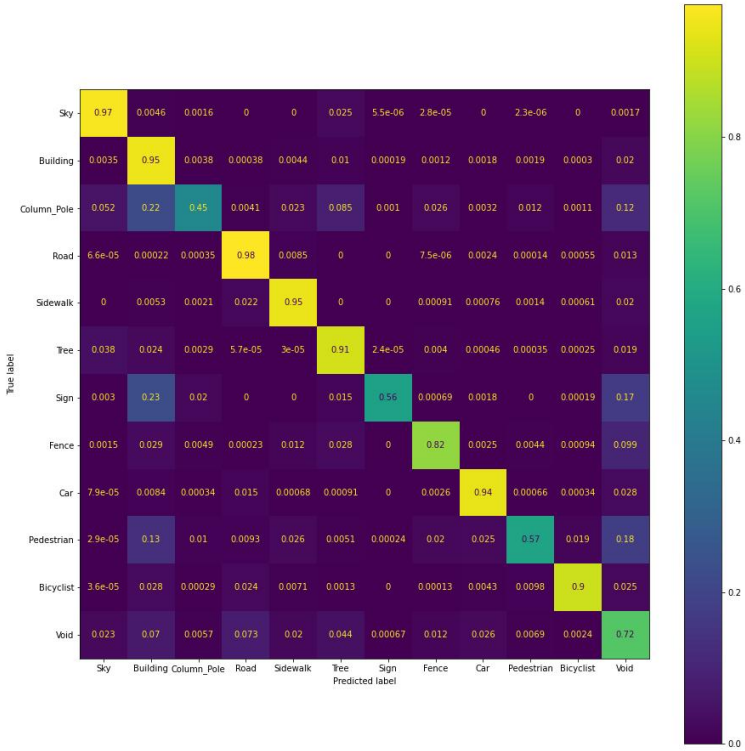

Figure 4.10: Confusion matrix for the model with ResNet 50, Image resolution of 480×360, and data augmentation

4.4 Discussion

4.4.1 Image Resolution

Table 4.2 shows the training results for ResNet 18-based models on image resolutions — 960×720 (i.e., original image size), 480×360 (i.e., 50% of the original image size), and 240×180 (i.e., 25% of the original image size). It was observed that reducing the image resolution from 100% to 50% reduced the validation accuracy by 0.9%, but the training duration decreased significantly by 70%. While reducing image resolution from 100% to 25% reduced the validation accuracy by 2.5%, but the training duration was decreased by nearly 90%. The testing results (i.e., global accuracy, and mean IoU) of the different image resolutions (see Table 4.3) showed the same pattern as the training results, where the accuracy decreased as the image resolution decreased.

4.4.2 ResNet 18 and ResNet 50

We tried to train a few models by replacing the ResNet 18 feature extractor with ResNet 50. Table 4.4 shows that the ResNet 50 models took almost double the time per epoch during training phase, as compared to the ResNet 18 models. But table 4.5 shows that there was not any significant improvement in terms of global accuracy and mean IoU scores between the two models. The training curve for the ResNet 50 model (see Figure 4.8) shows several dips in accuracies before converging, which is different from the ResNet 18 model, which had only one significant dip during training (see Figure 4.7).

4.4.3 Data Augmentation

Data augmentation is one of the techniques that was implemented in this milestone to tackle the class imbalance issue. The DeeplabV3+ model with ResNet 50 and an image size of 480×360 was trained on the original and augmented CamVid datasets for comparison. Table 4.6 shows that data augmentation was able to improve the global accuracy by nearly 1%. Also, Table 4.7 shows that the model that was trained on the augmented dataset was able to achieve

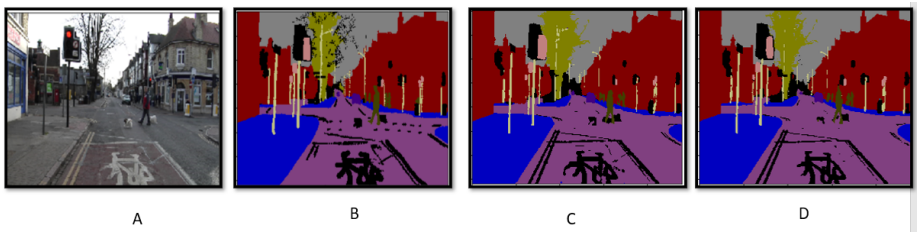

Figure 4.11: (a) original image, (b) groundtruth mask, and predicted images for models (c) ResNet 18 and (d) ResNet 50 for an image resolution of 960×720

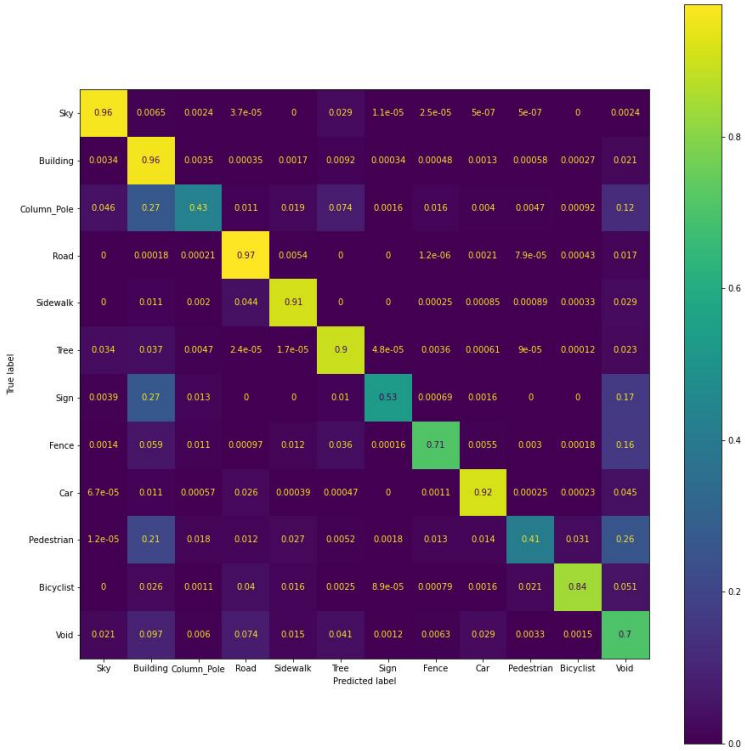

Figure 4.12: Confusion matrix for the ResNet 50 model trained without data augmentation for an image resolution of 480×360

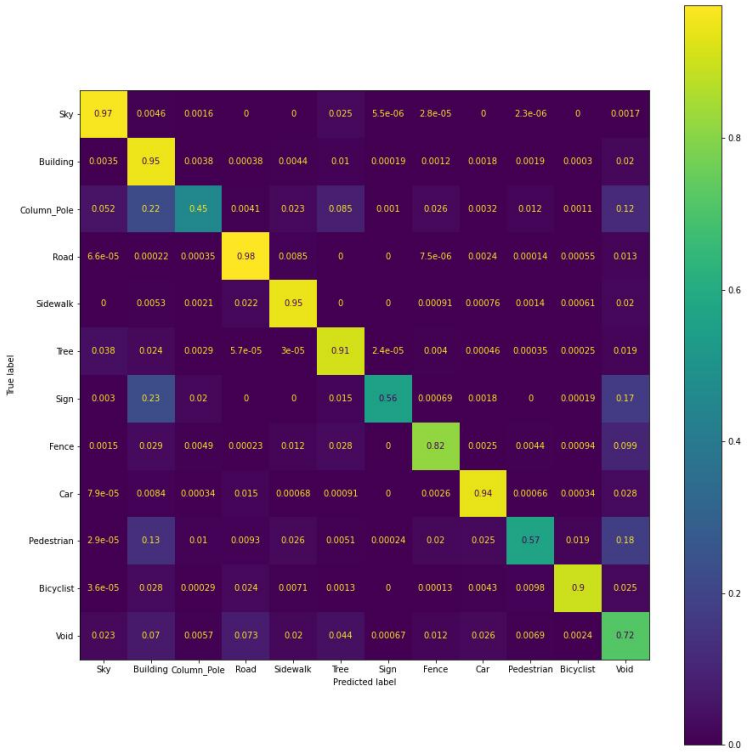

Figure 4.13: Confusion matrix for the ResNet 50 model trained with data augmentation for an image resolution of 480×360

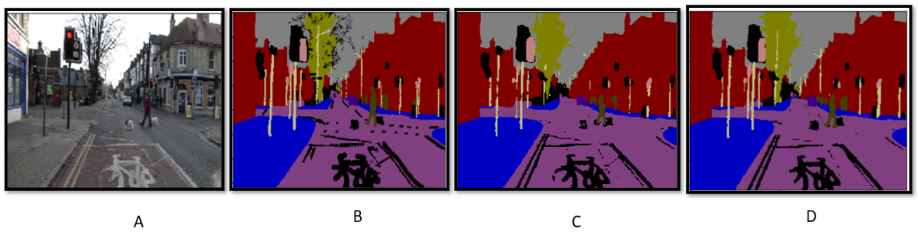

Figure 4.14: (a) original image, (b) groundtruth mask, Predicted masks for models (c) without data augmentation, and (d) with data augmentation

higher global accuracy and mean IoU on the testing dataset, as compared to the model that was trained on the original (i.e., non-augmented) dataset. The improvement in the mean IoU indicated that model got less confused about the classification of the pixels along the borders of the objects in the image. For a closer look at the individual class accuracies, the confusion matrix (see Figure B.7) of the model trained on the augmented training dataset showed diagonal values that are higher by 3-4% than those of the confusion matrix of the non-augmented model (see Figure B.5). Also, the off-diagonal values of the confusion matrix of the augmented model were 2-3% less than those of the confusion matrix of the non-augmented model. Therefore, these results indicate that the data augmentation techniques implemented in this milestone were capable of reducing the mis-classification happening between semantically relevant classes, and improved the individual class accuracies.

4.5 Future Plan

In the next milestone, we will train and test the DeepLabV3+ models on the Cityscapes dataset with different image resolutions. We will compare between the performance and results of the models on the CamVid and Cityscapes datasets for a further understanding.

4.6 Conclusion

In this milestone, a DeepLabV3+ based on the pre-trained ResNet 50 DCNN was built, and trained and tested on the CamVid dataset. The performance of the model based on the ResNet 50 backbone network was compared with the model that had ResNet 18 as a backbone network. Also, the DeepLabV3+ model based on ResNet 18 was tested on three different image resolutions to examine the effect of image resolution on the training time and accuracy. Furthermore, three techniques of data augmentation were implemented to tackle the class imbalance issues from the previous milestone. The results showed that using different backbone networks did not improve the global accuracy and meanIoU significantly. However, data augmentation was able to improve the individual class accuracies by 3-4%. In addition, in this milestone a prepossessing framework for loading and preparing the training images and masks for the Cityscapes dataset was created. In the next milestone, DeepLabV3+ models will be trained on different image resolutions of the Cityscapes dataset. Also, data augmentation techniques will be implemented on the Cityscapes dataset and the results will be compared with those of CamVid for a thorough analysis of the DeepLabV3+ models.

Chapter 5

Milestone 5: Final Report

5.1 Introduction

Semantic segmentation is the process where a label is assigned for each pixel of an input image. It plays a vital role in many practical applications, such as medical image segmentation, plant phenotyping, video surveillance, robot sensing and navigation of autonomous vehicles [3]. Autonomous driving requires a realtime and an accurate representation of the environment around the subject vehicle. Self-driving vehicles need to recognise the objects in their surrounding environment such as other cars, pedestrians, road lanes, traffic signs or traffic lights. This requirement makes semantic segmentation an important task for autonomous driving.

With the rise of deep learning technologies, convolution neural networks are applied to image segmentation and greatly outperform the traditional segmentation methods [3]. Since the encoder-decoder based deep convolution neural networks (DCNN) were proposed to handle semantic segmentation problems, a series of novel networks (i.e., DeepLabV3+) have been proposed based on DCNN [5, 3, 11, 12, 13, 14, 15]. In milestones 3 and 4, DeepLabV3+ [36] models were built for different image resolutions using ResNet 50 and ResNet 18 as feature extractors. Models were trained and tested on the CamVid dataset [42]. In this milestone, we repeated all the previous experiments using the Cityscapes dataset. Also, we summarized and discussed all the results for both datasets for better analysis of the impact of resolution, data augmentation, and feature extractors on the model performance.

5.2 Motivation

As per NHTSA [35], human behaviour is the root cause of top three reasons of road accidents, which are distracted driving, driving under influence and reckless driving. Annually, the US faces a loss of 38,000 human lives and 160 billion dollars due to road accidents. Applications such as driver assistance or

fully autonomous vehicle can reduce the number of road mishaps drastically. Autonomous vehicles have the potential to shape the urban life by modifying the travel behaviour, improving the transportation efficiency and increasing the driving safety [32]. Colorado State University's research found that if 90% of the cars in the USA were to become fully autonomous, an estimated 25,000 lives could be saved every year, with economic savings estimated at over 200 billion dollar a year [32]. These technologies can help in avoiding the costs of crashes, associated medical bills, vehicle repair and human time. Also, fewer crashes may reduce the costs of insurance. It can be a great helping hand for senior citizens and people with disability to let them live their life independently. In a fully automated vehicle, all occupants can safely pursue more productive or entertaining activities, like responding to email or watching a movie. Due to these reasons, we decided to implement semantic segmentation as a class project.

5.3 Problem Statement

The goal is to train a model that given an image, assigns correct label to each of its pixels. Our proposed goal was to reproduce DeepLabV3+ with ResNet 18 as a backbone network architecture using the Cityscapes [7] and CamVid [42] autonomous driving benchmark datasets. After accomplishing this objective, we added the following experimental objectives.

1. Examine the effect of different image resolutions on the network performance.
2. Evaluate the significance of data augmentation.
3. Analyze the effect of different backbone networks (i.e., ResNet 18 and ResNet 50).

5.4 Experimental Setup

5.4.1 Dataset

CamVid

Cambridge driving labeled video database (CamVid) [42] consists of 700 densely annotated frames and the resolution of each frame is 960×720. We split the dataset into 504 images for training, 54 images for validation, and 140 images for testing. The CamVid dataset consists of 32 classes. However, as per section 3.2.2, the models were trained using 11 classes based on the class weighted percentage. The remaining 21 classes were assigned to a void class. Furthermore, as discussed in section 3.2.2, CamVid suffers from inherent class imbalance issues.

Table 5.1: Camvid and Cityscape Dataset feature Comparison

Dataset	**Image Resolution**	**Number of Trained Class**
CamVid	720×960	12
Cityscapes	1024×2048	19

Table 5.2: Camvid and Cityscape Dataset feature Comparison

Dataset	**Training Images** count	**Validation images count**	**Testing images count**
CamVid	504	112	140
Cityscapes	2677	298	500

Cityscapes

Cityscapes [7] is one of the well-known datasets focusing on urban street scenes. It contains five thousand finely annotated images. Since the test dataset (i.e., 1525 images) is not available publicly, the remaining images were only considered for this project. The number of images considered for training, validation and testing were 2,677, 298 and 500, respectively. Coarsely-labeled images were not considered in this project. The original resolution of each image is 2048×1024. The dataset has 34 classes [7]. However, as mentioned in section 4.2.2, only 18 classes were selected.

Tables 5.1 and 5.2 show the image resolution, number of classes, and number of training, validation and testing image for the CamVid and Citscapes datasets, respectively.

5.4.2 Preprocessing

The preprocessing of the datasets has been discussed in detail in the earlier milestones (refer to section 3.2.3 for the CamVid dataset and section 4.2.3 for the Cityscapes dataset). Since the CamVid dataset is publicly available and can be downloaded using command line tools in HCC, we pulled it directly into Crane. The CamVid dataset consists of input images and their corresponding masks. Each pixel in the mask had a corresponding RGB color code. The original images and masks were converted into a numpy array, resized into different image resolutions and normalized. For our project, we selected 12 classes and mapped them to RGB color codes in their masks and then mapped to integer labels as shown in Figure 5.2.

Cityscapes had licensing restrictions which required logging into the site to download the dataset, so we had to download it locally and then upload it to Crane using SSH copy. We leveraged the tools provided by the authors of the dataset [50] and the Tensorflow-dataset to do most of the data preprocessing. Tensorflow dataset library provides tools to load Cityscapes datasets, but requires manual downloading due to the same licensing issues. We created a custom generator on top of the Tensorflow-dataset batch loader, which adjusts the image resolution as well. Iterating through the entire dataset using this

generator required 10 seconds, which was added to every epoch during training. This was an extra overhead, but eliminated kernel crashing due to the memory issues. Tensorflow-datasets already converted the color codes in masks into integer labels, but for visualization we used the mapping shown in Figure **??**. The mapping was as per the tool provided by the authors of the dataset.

Class	Color Label	Integer label
Road		0
Sidewalk		1
Building		2
Fence		3
Wall		4
Vegetarian		5
Terrain		6
terrain		7
Car		8
Bicycle		9
Bus		10
Truck		11
Train		12
Sky		13
Pole		14
Traffic light		15
Traffic sign		16
Person		17
Rider		18
Void		19

Figure 5.1: Cityscapes Mapping class to color and integer label

Class	Color label	Integer label
Sky		0
Building		1
Column Pole		2
Road		3
Sidewalk		4
Tree		5
Sign		6
Fence		7
Car		8
Pedestrian		9
Bicyclist		10
Void		11

Figure 5.2: CamVid mapping class to color and integer label

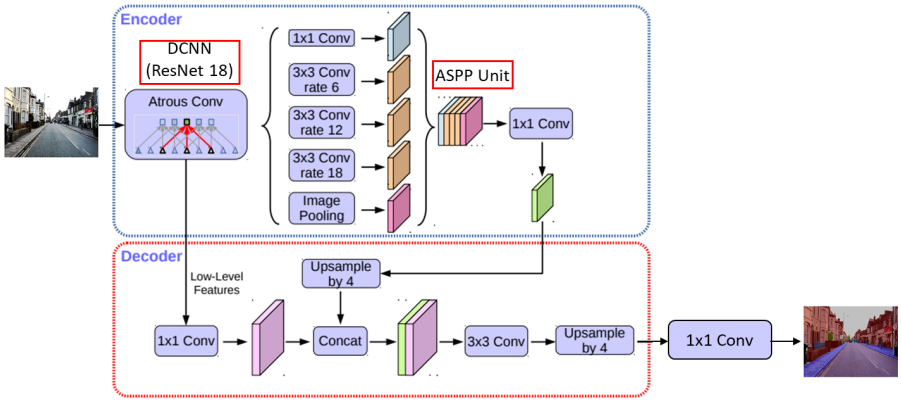

Figure 5.3: DeepLabV3+ architecture (adapted from the source: [36]).

5.4.3 Architecture

As discussed in milestones 2 and 3, the past studies used several different approaches for building the benchmark deep learning-based architectures in semantic segmentation. The main approaches of semantic segmentation architectures are based on fully convolutional networks (FCNs) for end-to-end pixel-wise learning [5, 3]. In this report, several DeepLabV3+ architectures incorporating different pre-trained network as backbone and feature extractors [3, 16] were built to accomplish the semantic segmentation tasks on the CamVid and the Cityscapes datasets. The DeepLabV3+ network is an encoder-decoder architecture and one of the benchmark models used for implementing the semantic segmentation task on the autonomous driving datasets (i.e., CamVid) [3, 43]. Figure 5.3 shows the architecture of the DeepLabV3+ models implemented in this project.

The encoder unit of the DeepLabV3+ network uses a deep convolutional neural network (DCNN) such as using ResNet 18 or ResNet 101, etc. as a dense feature extractor [37]. The DCNN is known as the backbone network and it generates a low-level feature map, which contains high semantic information. Figure 5.4 shows the ResNet 18 architecture used as the DCNN for the DeepLabV3+ network. Moreover, the encoder unit employs an atrous spatial pyramid pooling (ASPP), which applies four parallel atrous convolutions with different rates to exploit information at different grid scales along with an image pooling layer. Atrous convolution is also known as dilated convolution, where the weights of the atrous convolution's kernel are spaced by rate r. The dilated kernel allows the capturing of bigger field of view with the same number of weights as a normal kernel without dilation. Figure 5.3 shows the details of the ASPP unit. Then, 1×1 convolutional layers were applied to the outputs of the ASPP unit and the low-level feature maps to reduce the number of channels for an efficient training. Before feeding the feature maps produced by the

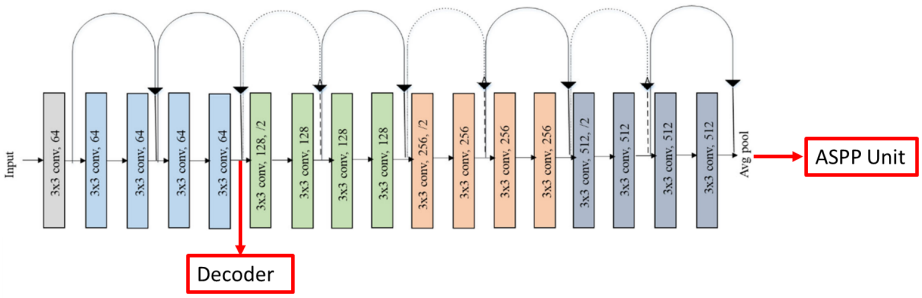

Figure 5.4: The ResNet 18 architecture which was utilized as the DCNN for the DeepLabV3+ architecture (adapted from the source: [44]).

encoder into the the decoder unit, the feature maps were bi-linearly upsampled by a factor of 4. The decoder unit recovers the feature maps and the boundary information by concatenating the feature maps from the encoder unit with the corresponding low-level feature maps from the DCNN, as shown in Figure 5.3. Then, two 3×3 convolution blocks were applied to refine the concatenated feature maps for sharper segmentation results. At the end, a bi-linear upsampling of factor 4 were applied to the output of the 3×3 convolution blocks. The decoder unit helps with the recovery of the boundary information in semantic segmentation for better results.

For a pixel-wise classification, the output of the last 3×3 block in the decoder unit was passed through a convolution 2D block. The number of filters was equal to the number of classes, the height and the width of the convolution layers were matching those of the input image, as shown in Figure 5.3. The final convolution 2D block employed the softmax activation function. The Adam optimizer and the cross entropy loss function were used for training the network. The created-from-scratch DeepLabV3+ network had a total of 16.6 million trainable parameters. All the activation functions used were ReLu. For comparison, in this project, the pre-trained DCNN ResNet 18 and ResNet 50 were used as the backbone networks for the DeepLabV3+ models. The pre-trained networks were trained on the Imagenet dataset [44] and were loaded from the Keras library. The total number of parameters of the DeepLabV3+ model based on ResNet 50 was nearly 40.5 million, while the ResNet 18-based model had nearly 16.6 million parameters. Therefore, the ResNet 50-based models were nearly 2.5 times denser. The DeepLabV3+ models were built using the Keras and Tensorflow libraries as per the Keras tutorial on semantic segmentation [51].

5.4.4 Data Augmentation

To tackle the class imbalance issues of the CamVid dataset, the OpenCV library [46] in Python was used for implementing three data augmentation techniques on the training portion of the CamVid dataset. As explained in Milestone

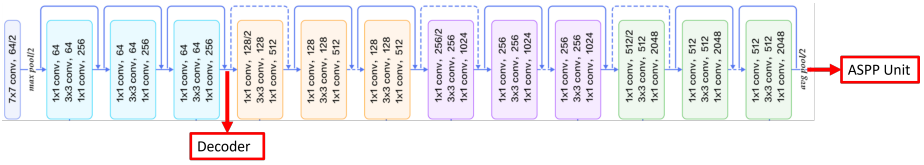

Figure 5.5: ResNet 50 architecture (adapted from the source: [49]).

Table 5.3: Different training parameters

Parameters	CamVid	Cityscapes
Image resolution	960×720, 480×360 and 240×180	1024×512, 512×256 and 256×128
Feature extractor	ResNet 18 and ResNet 50	ResNet 18 and ResNet 50
Data augmentation	Data with and without image augmentation	Data without image augmentation

4, the data augmentation framework was built and applied to both the input images and the masks. The augmented dataset was generated after each of the horizontal, vertical, and combined horizontal and vertical flip techniques were individually applied to the original dataset. The flip techniques were chosen to prevent the models from overfitting over the training data and tackle the class imbalance issues encountered in the previous milestones.

5.4.5 Training

We experimented with different parameters as shown in Table 5.3. For CamVid dataset, the parameters were image resolutions, feature extractors and presence and absence of data augmentation. For Cityscapes, the parameters were image resolution and feature extractor networks. We trained all the models for different combinations of parameters with a patience of ten epochs as an early stopping criterion. The Adam optimizer was used for training the models. The learning rate selected was 0.001 as per Mathwork's implementation [16]. The training and the testing were conducted using the Jupyter Lab on Crane. For CamVid, the configuration used had 16 cores CPU, 62 GB RAM and two 32 GB GPUs. This made training and testing faster. Similar configuration was used for Cityscape but with only one 32 GB GPU. This was because we faced difficulties in getting higher resources from the Crane.

5.4.6 Testing

Testing for CamVid dataset has been explained in detail in the previous milestones (see Section 3.2.6 and Section 4.2.7). For Cityscapes, the generator described in preprocessing section was created with the test dataset and passed to Keras. The test metrics used to evaluate the results were global accuracy and mean IoU. We also computed confusion matrix for every run. We considered each pixel in all the images as a separate data point to calculate the classification

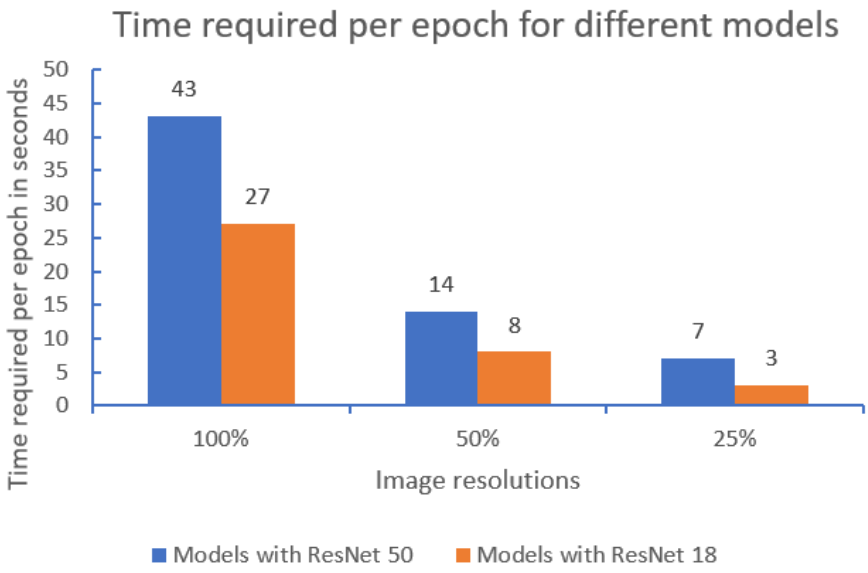

Figure 5.6: Comparing time required per epoch for different models using CamVid dataset

accuracies and confusion matrices.

5.5 Experimental Results

We have divided the experimental results into two different sections for each of the Camvid and Cityscapes datasets. Further, as in Milestone 4, only the relevant results will be presented here and all additional results (i.e., confusion matrices, classification reports and learning curves) of all the models can be found in the appendix section.

5.5.1 Camvid

Models were trained with different backbone networks (i.e., ResNet 18 and ResNet 50) and three different resolutions — 100%, 50% and 25% of the resolution of the original image.

Training

Figures 5.6 and 5.7 show total number of epochs and time required in seconds for each epoch for different configuration. Further, Figure 5.8 shows the highest validation accuracies obtained for each model configuration.

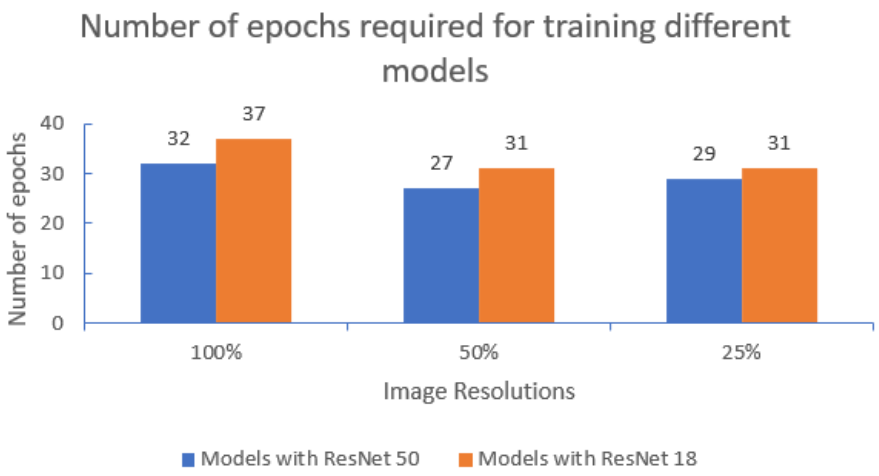

Figure 5.7: Comparing number of epochs required for training for different models using CamVid dataset

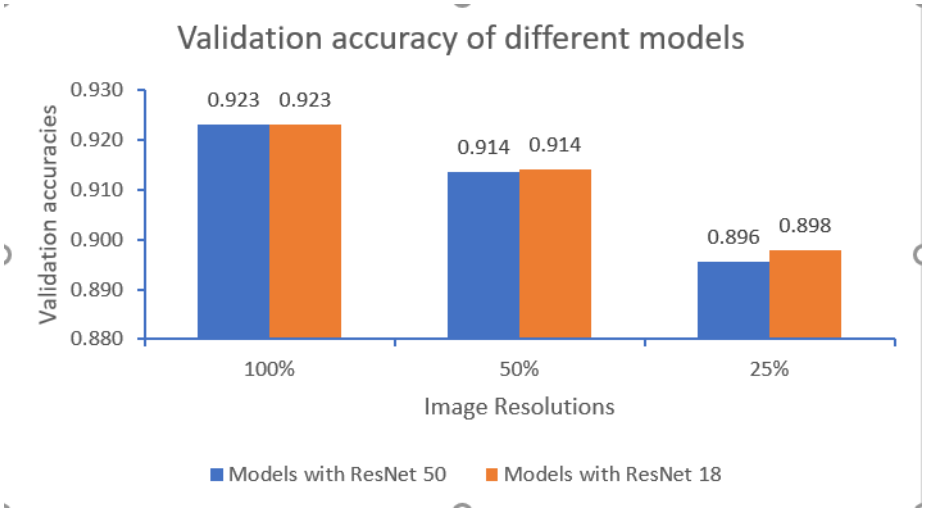

Figure 5.8: Comparing Validation accuracies for different models using CamVid dataset

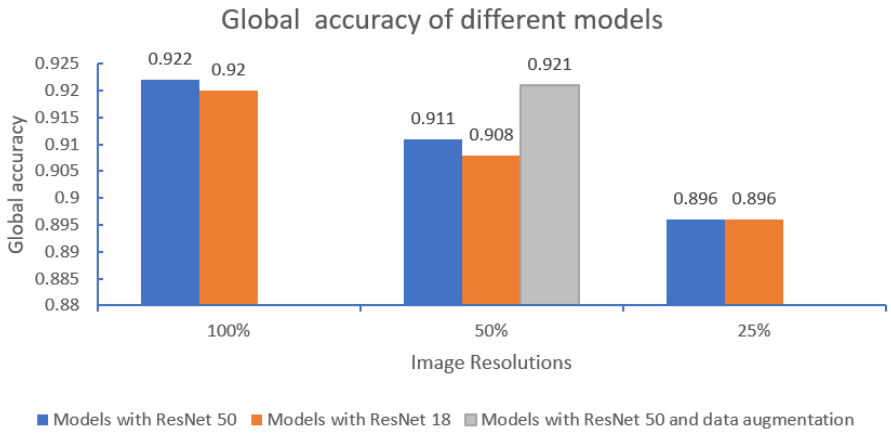

Figure 5.9: Comparing global accuracies of different models using CamVid dataset

Testing

During our first iteration, we reproduced the original implementation of DeepLabV3+ on the CamVid dataset. The test results of our models, both the global accuracy and mean IoU, were better by $3-4\%$ as compared to the test results of the original implementation of Mathwork [16]. Figure 5.9 shows the global accuracies obtained on the test set for all the model configurations trained and Figure 5.10 shows the mean IoU for all the models. Figure 5.11 shows the predicted masks of all the trained models for a sample input image along with the ground truth mask.

5.5.2 Cityscapes

Models were trained with two different backbone networks(i.e., ResNet 18 and ResNet 50) and 3 different resolutions — 50%, 25% and 12.5% of the original image. It was not possible to train at 100% of the original resolution because it was too large.

Training

Figures 5.13 and 5.12 show the total number of epochs and time required in seconds for each epoch for different model configurations. Due to lazy loading operation, it takes 10 seconds to go iterate through the entire data set without any other operations. So, this would be additional time added in each epoch. Figure 5.14 shows the highest validation accuracies obtained for each model configuration.

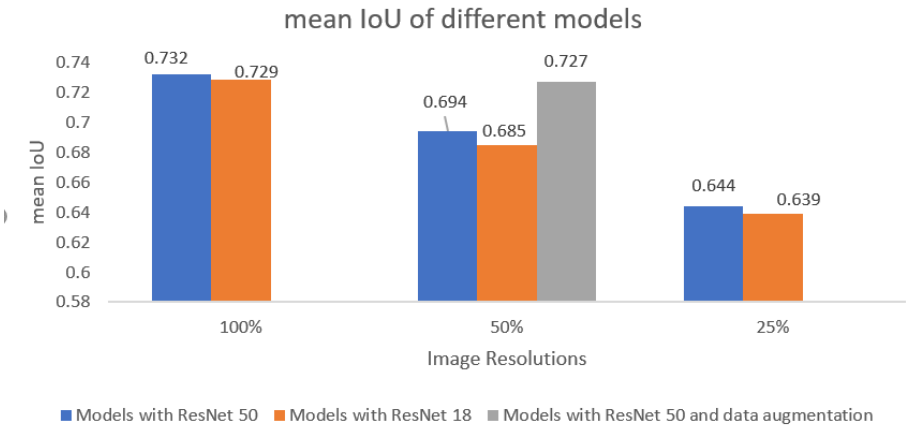

Figure 5.10: Comparing mean IoUs of different models using CamVid dataset

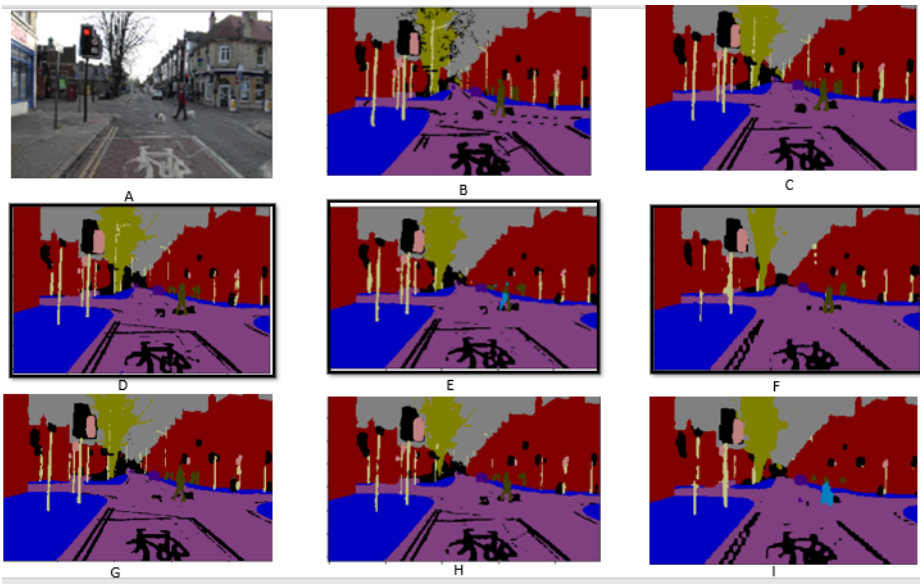

Figure 5.11: CamVid dataset: (A) Original image, (B) ground truth mask; predicted mask trained by model with ResNet 50 and resolutions (C) 480×360 and data augmentation (D) 960×720, (E) 480×360 (F) 240×180, predicted mask trained by model with ResNet 18 and resolutions (G) 960×720, (H) 480×360 (I) 240×180

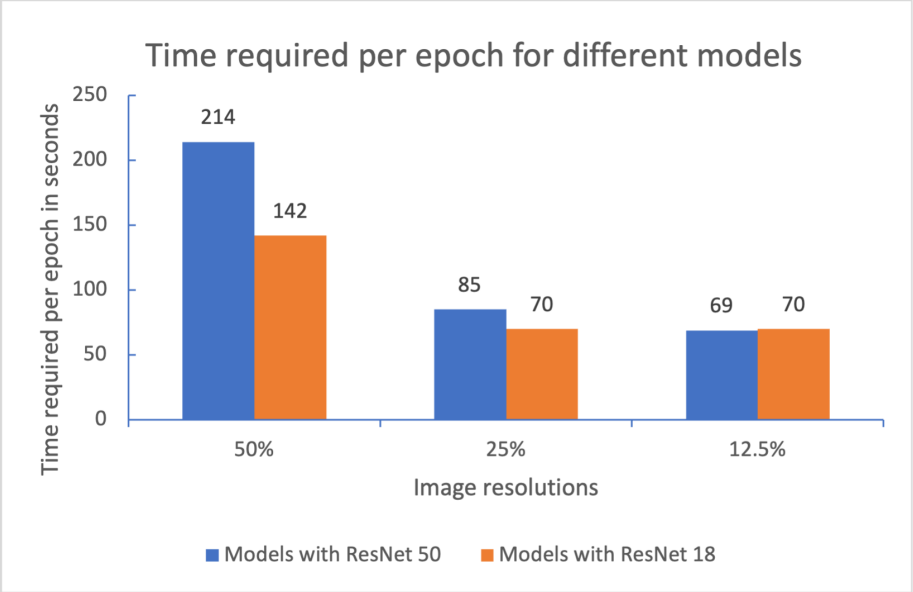

Figure 5.12: Comparing epochs required for training for different models using Cityscapes dataset

Testing

Figure 5.16 shows the global accuracies obtained on the test set for all model configurations trained and Figure 5.15 shows the mean IoU for all models. Figure 5.17 shows the predicted mask from all models trained for a sample image along with the ground truth mask.

5.6 Discussion

5.6.1 Qualitative Analyses

Figures 5.11 and 5.17 qualitatively show that the DeepLabV3+ models were able to predict masks that were similar to ground-truth masks. Collectively, the predicted pixel labels were similar to the ground-truth pixel labels, despite the use of different image resolutions, and backbone architectures. Moreover, the results showed the model would often incorrectly classify pixels around the boundaries of the objects, but pixels away from boundaries were oftentimes correctly classified. Therefore, the qualitative comparisons proved that the DeepLabV3+ models are robust to changes, however, deeper investigations of the model performance will be provided in the following quantitative results sections.

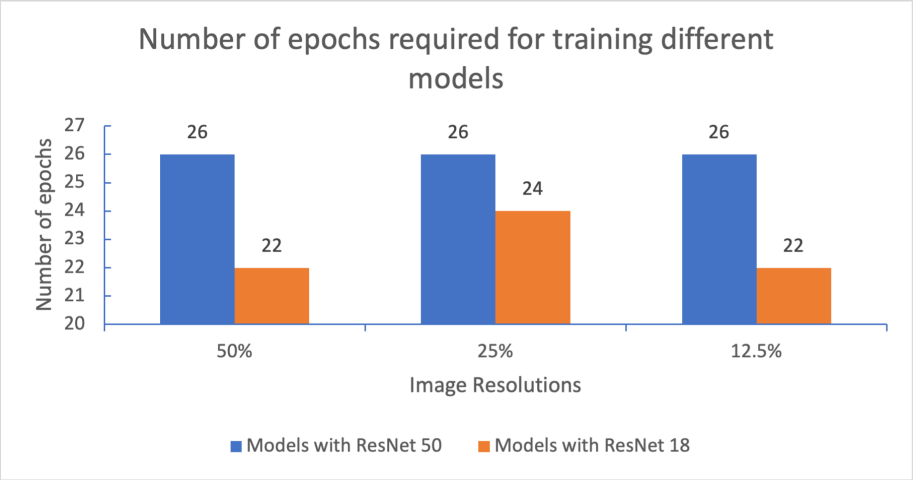

Figure 5.13: Comparing number of epochs required for training for different models using Cityscapes dataset

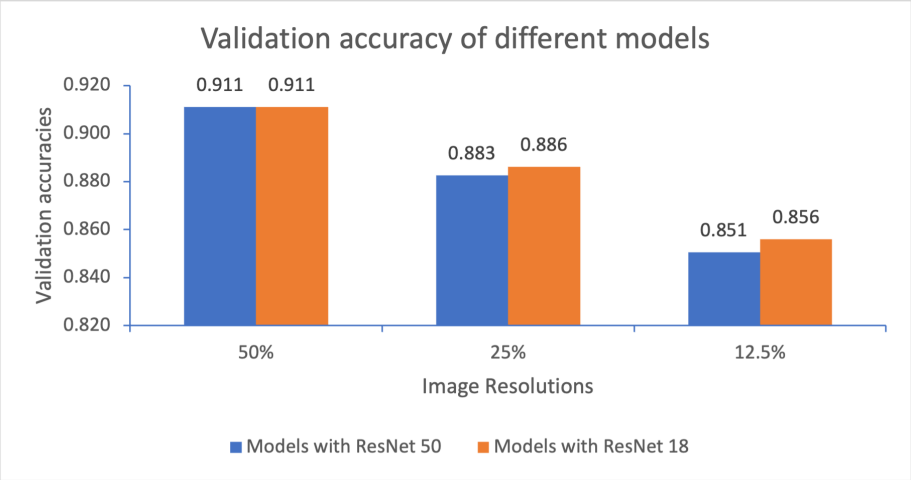

Figure 5.14: Comparing validation accuracies of different models using Cityscapes dataset

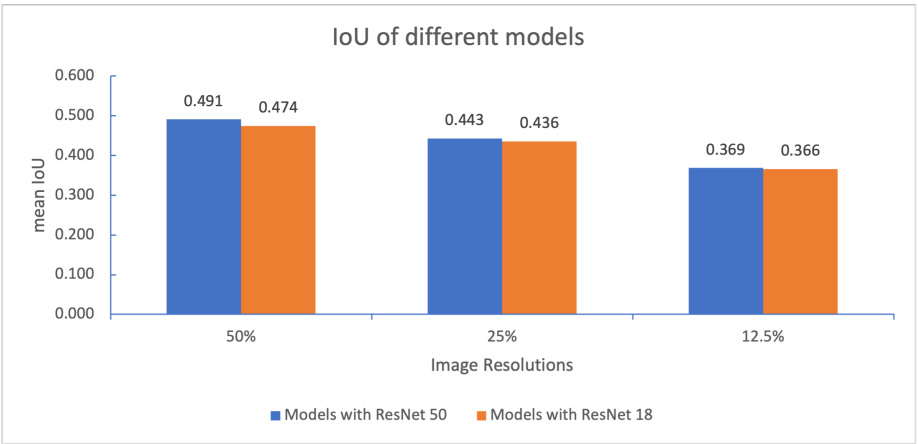

Figure 5.15: Comparing mean IoU of different models using Cityscapes dataset

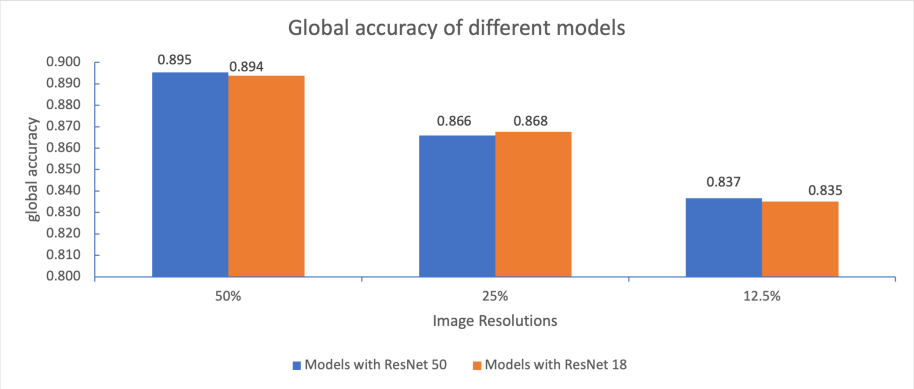

Figure 5.16: Comparing global accuracies of different models using Cityscapes dataset

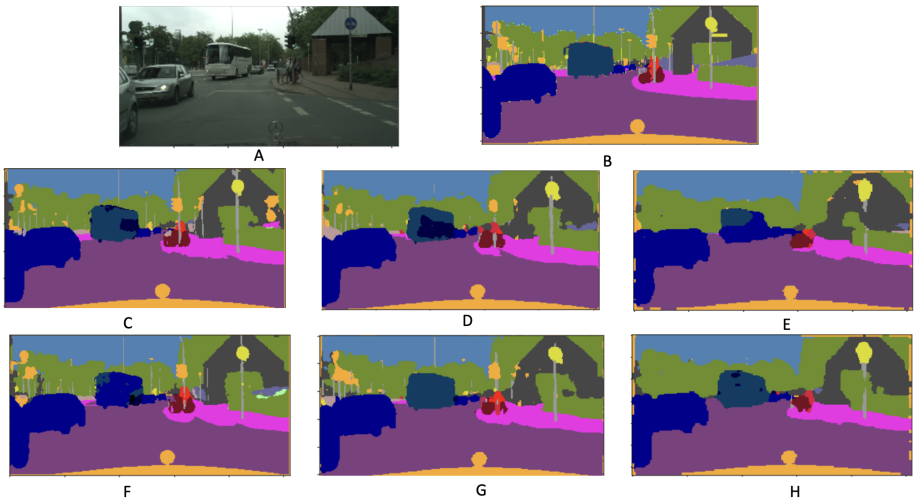

Figure 5.17: (A) Original image (B) ground truth mask; predicted masks trained by model with Resnet 18 backbone and resolutions (C) 1024×512(D) 512×256 (E) 256×128; predicted masks trained by model with Resnet 50 backbone and resolutions (F)1024×512 (G)512×256 (H)256×128

5.6.2 Image Resolution

Both the CamVid and Cityscapes datasets had the same the trend of results for the training and testing results on different image resolutions. Figures 5.6 show that reducing the image resolution from 100% to 25% reduced the required time per epoch during training by nearly 90%. Also, Figure 5.12 shows that the training duration decreased by nearly 50 – 60% as the resolution decreased from 50% to 25% for Cityscapes. However, the results for lower resolutions (i.e., 25% and 12.5%) did not exhibit the same trend as the resolution decreased. This could be attributed to the lazy loading operation of the data, which added more overhead cost during training that outweighed the training duration of the model on relatively low resolutions. The number of epochs required during training across the different image resolutions did not vary much, as shown in Figures 5.7 and 5.13. Despite the plummeting of the training duration due to drastically reducing the image resolution, the global accuracy of both datasets were only decreased by 3 – 7%, as shown in Figures 5.8, 5.9, 5.14, and 5.16. As compared to the accuracy results, the mean IoU was reduced more reduced as the resolution decreased. The mean IoU for the CamVid dataset was decreased by nearly 14%, while the mean IoU for the Cityscapes dataset was reduced by nearly 27%, as shown in Figures 5.10, and 5.15.

5.6.3 Backbone Architecture

We trained and tested DeepLabV3+ models with two backbone networks (i.e., ResNet 18 and ResNet 50) on both the CamVid and Cityscapes datasets for a further understanding. The results showed that the ResNet 50 models took almost double the time per epoch during training phase for the CamVid dataset, as compared to the ResNet 18 models, as shown in Figure 5.6. However, the Cityscapes dataset (see Figure 5.12) had the same trend for the image resolution of 50%, as the training duration of the ResNet 50 model was nearly 50% longer than that of the ResNet 18 model. This can be attributed to the fact that ResNet 50 network has nearly 2.5 times the parameters of the ResNet 18 network. However, for the image resolutions of 25% and 12.5% of the Cityscapes dataset, there was nearly no differece in the training duration between the ResNet 50 and ResNet 18 models. This trend of results can be due to the fixed cost of using data generators for the Cityscapes dataset, which greatly dominates the training duration at low image resolutions (i.e., 12.5% and 25%) and makes the training duration nearly the same across low image resolutions. The accuracy and mean IoU results of both backbone networks for the CamVid dataset did not show a significant difference, as shown in Figures 5.8, 5.9, 5.10, 5.9, 5.14, 5.16, and 5.15.

5.6.4 Data Augmentation

Three data augmentation techniques were implemented to tackle the class imbalance issue in the CamVid dataset. The DeepLabV3+ model with ResNet 50 and an image size of 480×360 was trained on the original and augmented CamVid datasets for comparison. The results showed that data augmentation was able to improve the global accuracy by nearly 1%. Also, Figures 5.9 and 5.10 show that data augmentation model was able to achieve higher global accuracy and mean IoU on the testing dataset, as compared to the model that was trained on the original (i.e., non-augmented) dataset. Therefore, these results indicate that the data augmentation techniques were capable of reducing the mis-classification happening between semantically relevant classes, and improved the individual class accuracies.

Generally, the models did not exhibit sharp variations in the training and testing results due to the changes occurred. The results showed that the DeepLabV3+ models performed better on the CamVid dataset than the Cityscspes dataset. We believe that this trend can be attributed to the use of more number of classes for the Cityscapes dataset than the CamVid dataset. Also, the Cityscapes dataset has more variations, which made applying semantic segmentation to it a more tricky task.

5.7 Future Work

To improve the performance of the semantic segmentation, two problems can be tackled. We list these as future works.

1. Class imbalance problem: A common problem in semantic segmentation is class imbalance issue, which tends to reduce the classification accuracy of less weighted-classes. As per literature, the focal loss function [52] can be used to tackle the class imbalance issue. Focal loss applies a modulating term to the cross-entropy loss to focus the learning on hard mis-classified examples. It is a dynamically scaled cross entropy loss, where the scaling factor decays to zero as the confidence in the correct class increases. Focal loss has proven to be effective at balancing loss by increasing the loss on hard-to-classify classes. As a future work, we will evaluate the impact of the focal loss function on our DeepLabV3+ models.

2. Better performance metrics for boundary errors: Models getting some errors in the border is less severe than failing to recognise pixels within the objects. Borders are a bit ambiguous and even humans might get confused while classifying them. A weighted performance metrics that takes position of pixel into account while evaluating the model performance might better evaluate real world application. (This came up during the QA section of our presentation.)

5.8 Conclusion

In this milestone, we implemented several DeepLabV3+ model configurations on the Cityscapes dataset. We trained it on 50%, 25% and 12.5% of the original resolution while using the ResNet 18 and ResNet 50 backbone architectures. We also summarised all the results for both the Cityscapes and CamVid datasets for better analysis of the effects of resolutions and backbone architectures. We found that while decreasing the resolutions decreased the model performance measured in terms of gloal accuracy and mean IoU, the backbone architecture did not have much impact. The best DeepLabV3+ models were able to achieve nearly accuracies of 90 – 92% for both datasets, while the highest mean IoU achieved for the CamVid dataset was nearly 73% and for the Cityscapes dataset was nearly 50%.

So, in conclusion, in this project, we accomplished the following

1. We explored literature from two different problems — semantic segmentation and image colorization. We selected semantic segmentation for this project.

2. We explored two different architectures — UNet and DeepLabV3+.

3. Implemented DeepLabV3+ and trained it on CamVid and Cityscapes dataset.

4. Studied the effects of image resolution and different backbone networks on training time and model performance.

5. Explored data augmentation as a solution to class imbalance issue.

Appendix A

Learning curves

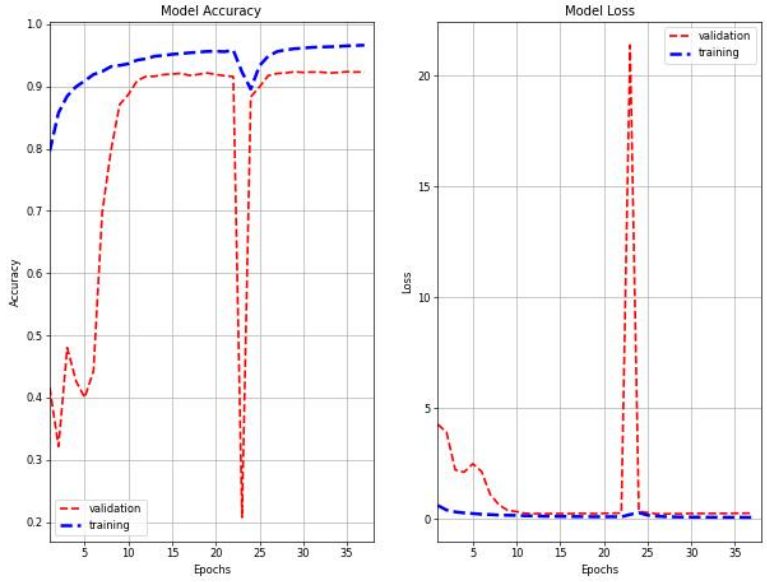

Figure A.1: Learning curve for model with ResNet 18 and image resolution of 960×720 on CamVid dataset

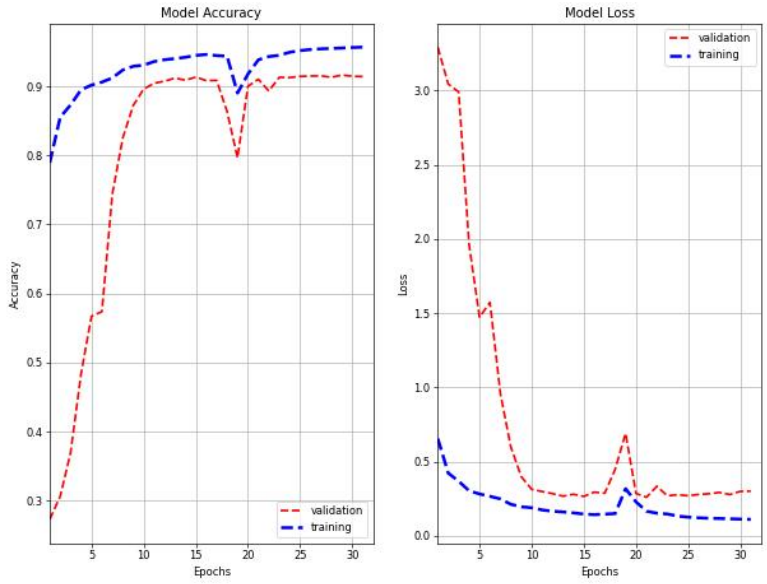

Figure A.2: Learning curve for model with ResNet 18 and image resolution of 480×360 on CamVid dataset

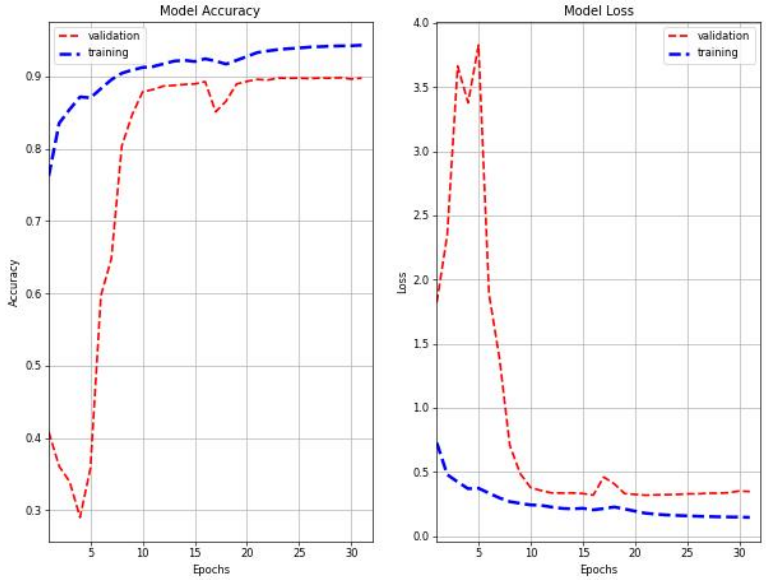

Figure A.3: Learning curve for model with ResNet 18 and image resolution of 240×180 on CamVid dataset

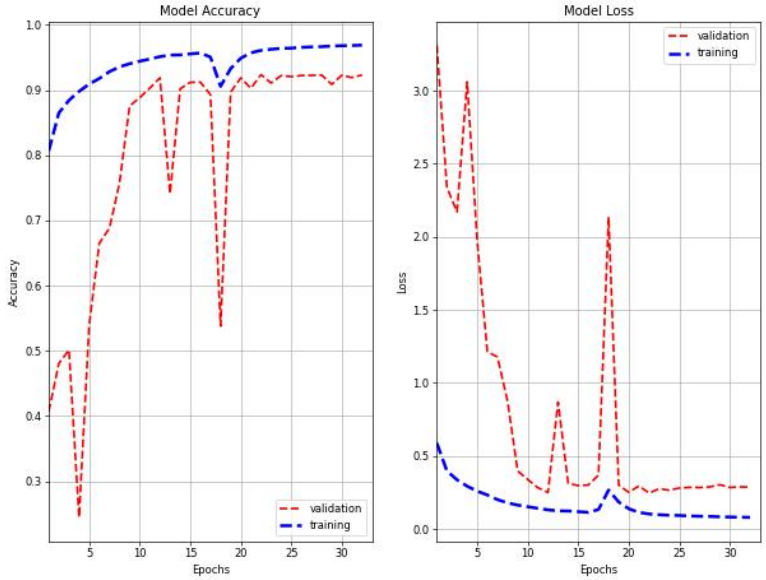

Figure A.4: Learning curve for model with ResNet 50 and image resolution of 960×720 on CamVid dataset

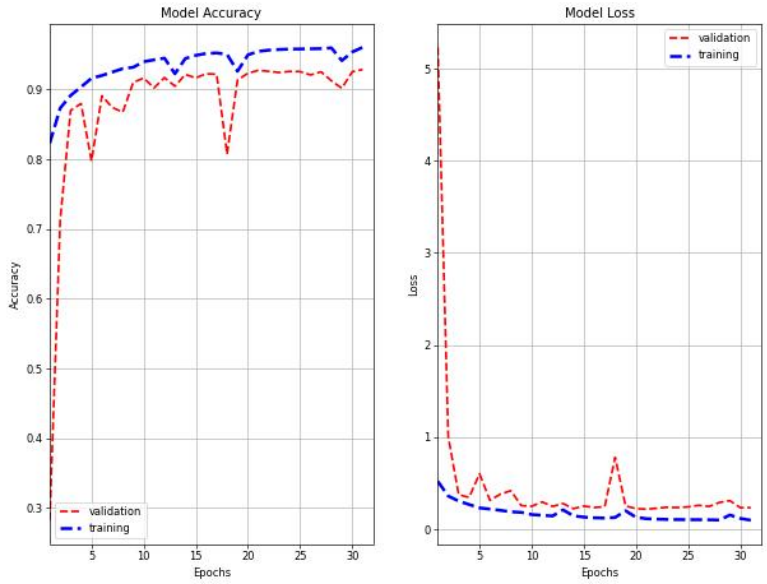

Figure A.5: Learning curve for model with image augmentation and ResNet 50 and Image resolution of 480×360 on Camvid dataset

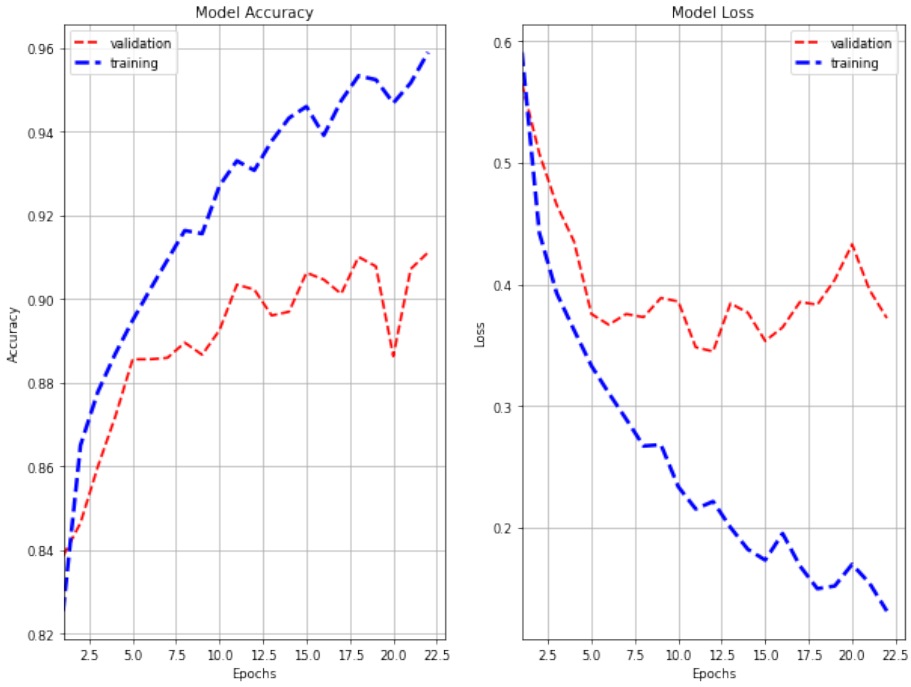

Figure A.6: Learning curve for model with image augmentation and ResNet 18 and image resolution of 1024×512 trained on Cityscapes dataset

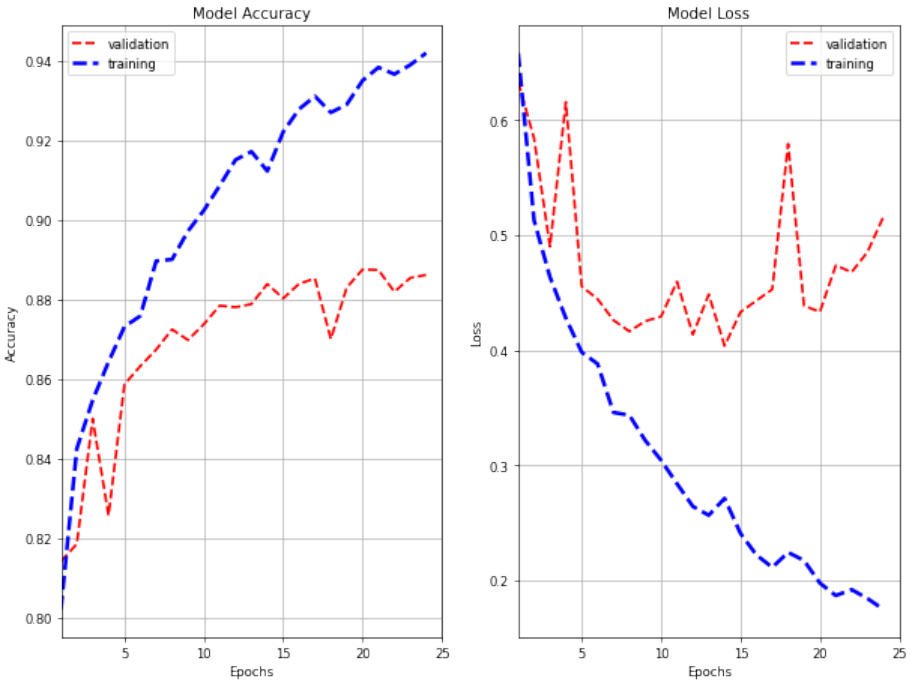

Figure A.7: Learning curve for model with image augmentation and ResNet 18 and image resolution of 512×256 trained on Cityscapes dataset

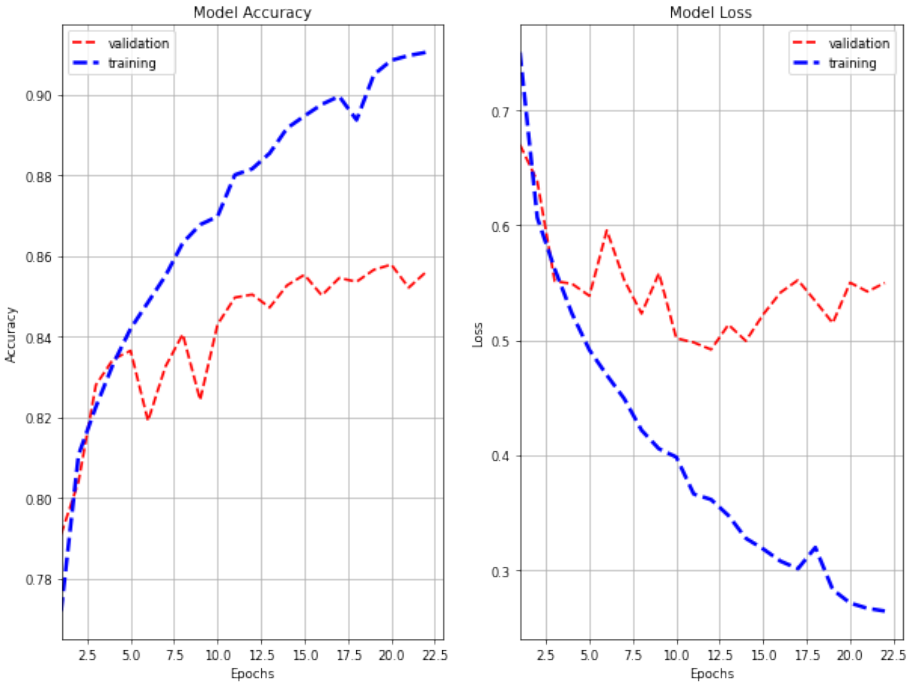

Figure A.8: Learning curve for model with image augmentation and ResNet 18 and image resolution of 256×128 trained on Cityscapes dataset

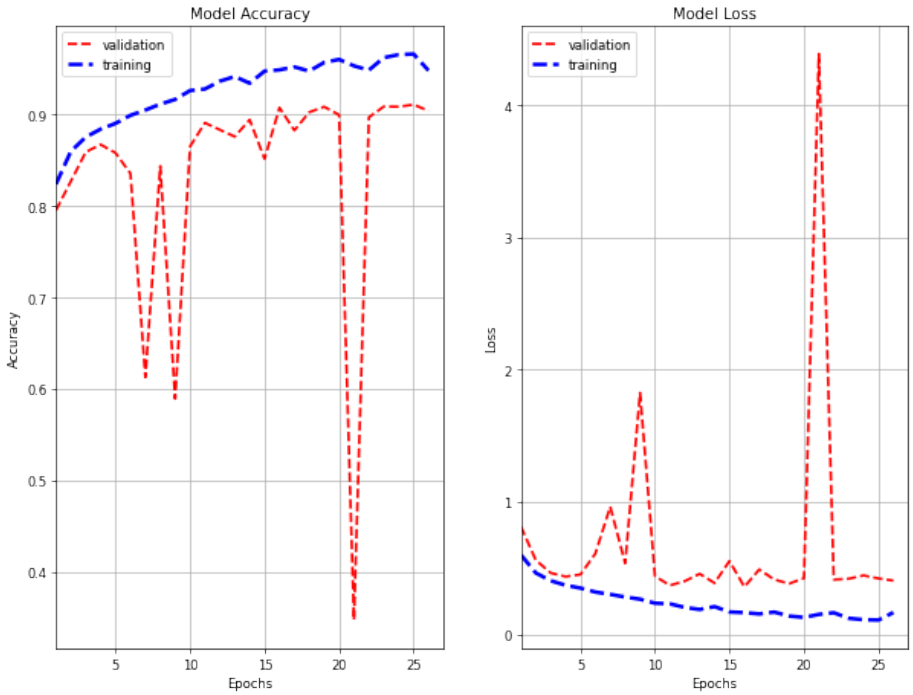

Figure A.9: Learning curve for model with image augmentation and ResNet 50 and Image resolution of 1024×512 trained on Cityscapes dataset

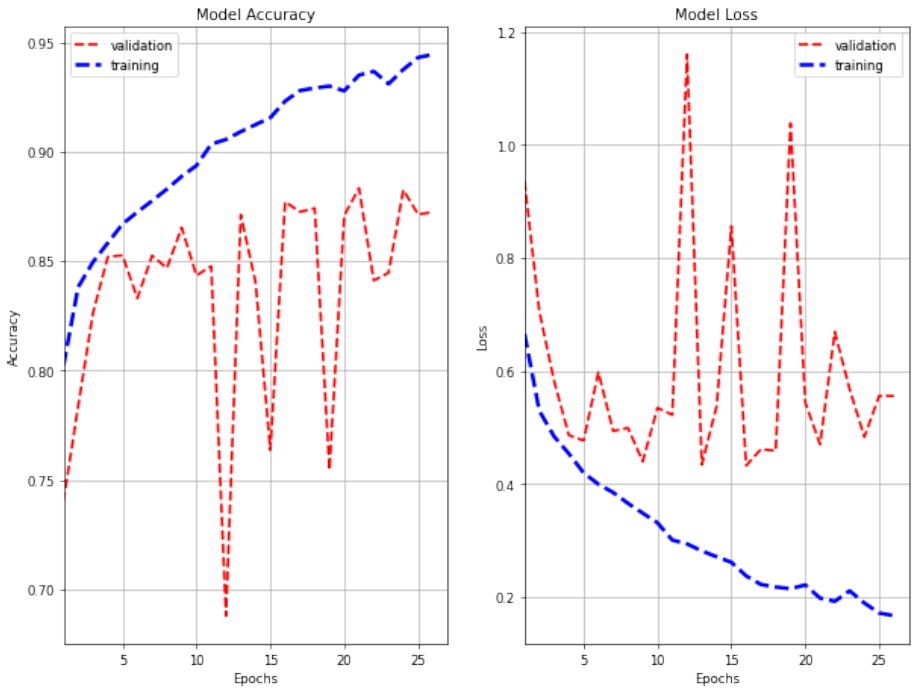

Figure A.10: Learning curve for model with image augmentation and ResNet 50 and Image resolution of 512×256 trained on Cityscapes dataset

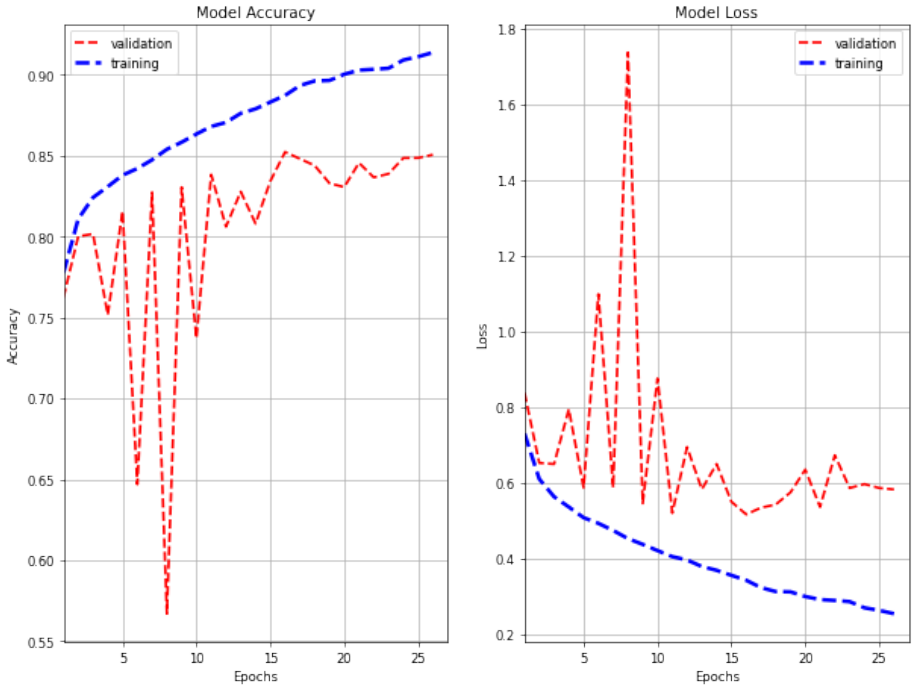

Figure A.11: Learning curve for model with image augmentation and ResNet 50 and Image resolution of 256×128 trained on Cityscapes dataset

Appendix B

Confusion matrix

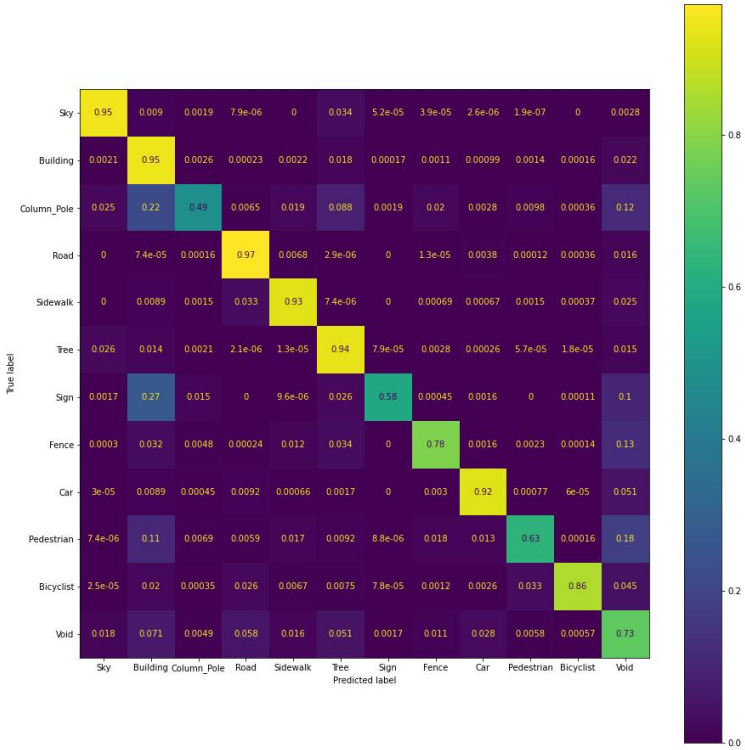

Figure B.1: Confusion matrix for model with ResNet 18 and image resolution of 960×720 on CamVid dataset

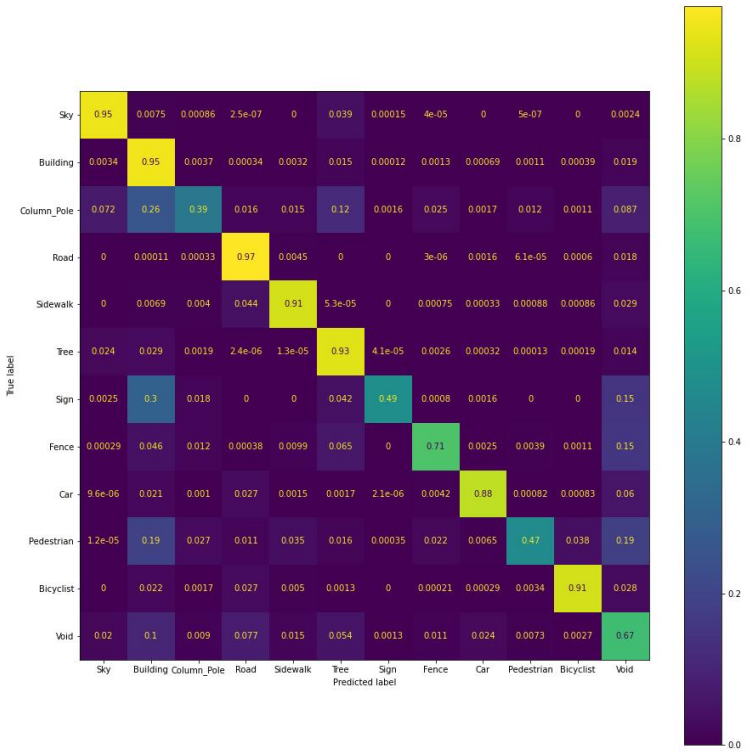

Figure B.2: Confusion matrix for model with ResNet 18 and image resolution of 480×360 on CamVid dataset

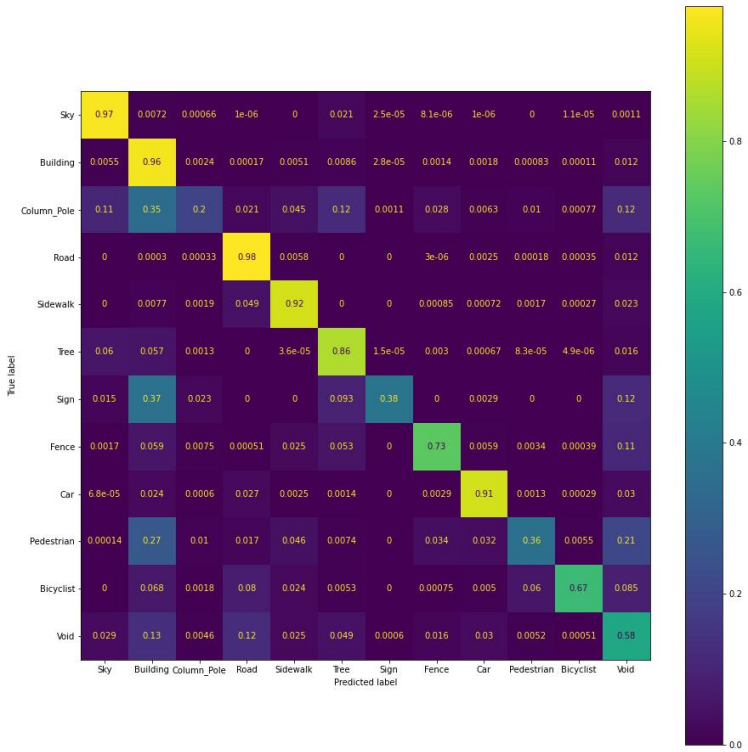

Figure B.3: Confusion matrix for model with ResNet 18 and image resolution of 240×180 on CamVid dataset

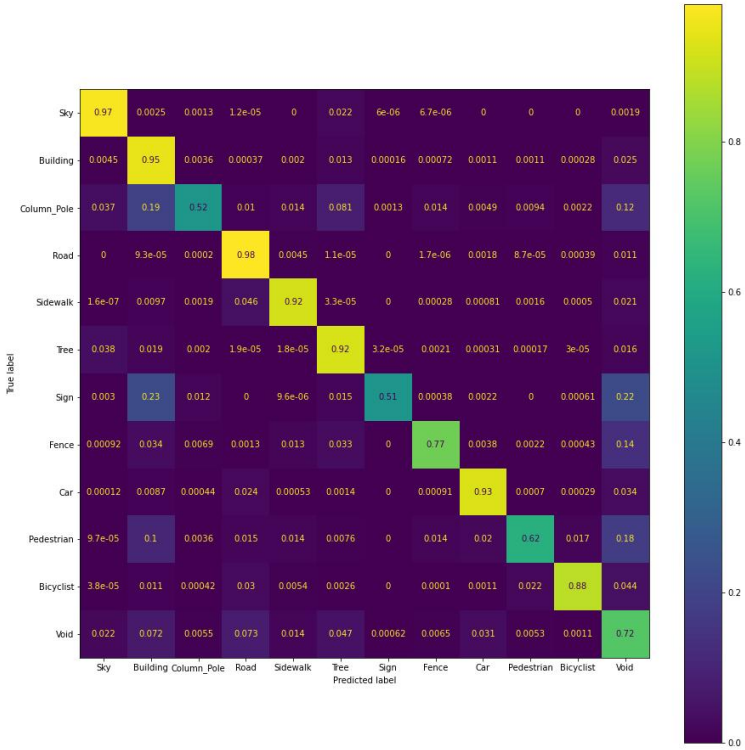

Figure B.4: Confusion matrix for model with ResNet 50 and image resolution of 960×720 on CamVid dataset

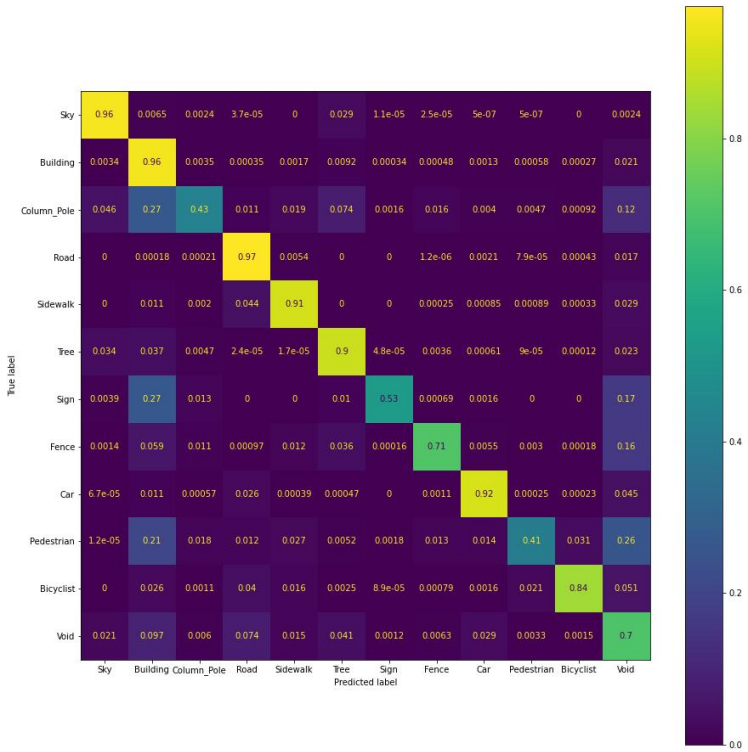

Figure B.5: Confusion matrix for model with ResNet 50 and image resolution of 480×360 on CamVid dataset

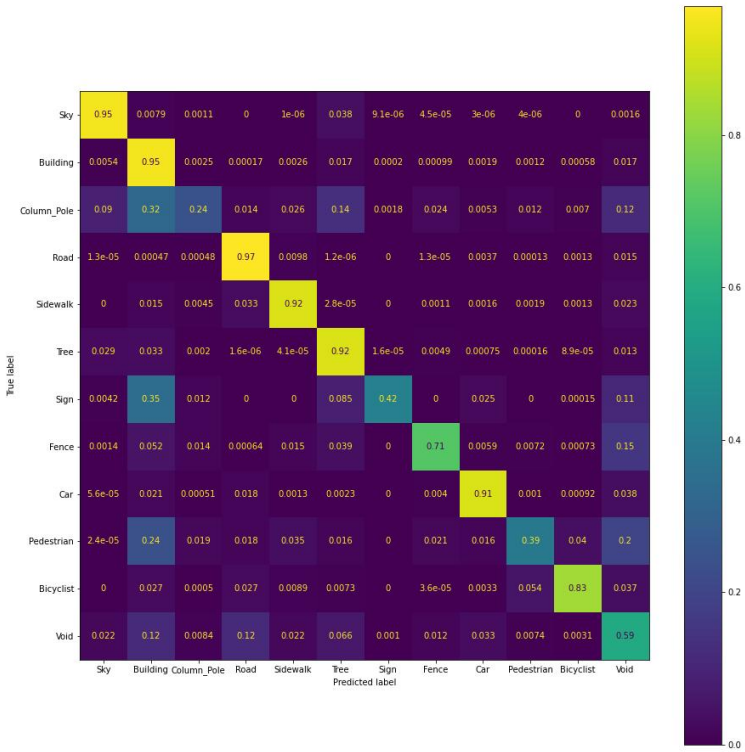

Figure B.6: Confusion matrix for model with ResNet 50 and image resolution of 240×180 on CamVid dataset

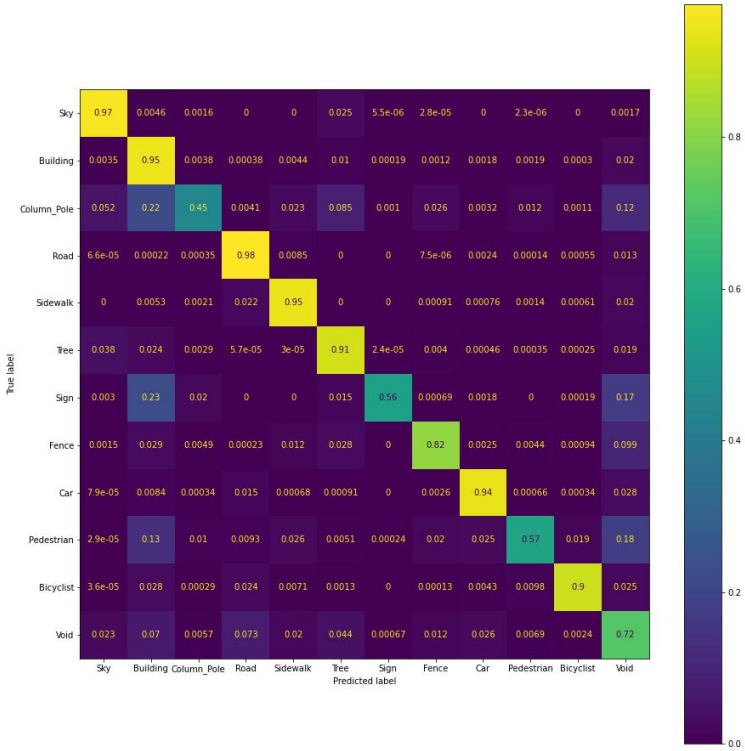

Figure B.7: Confusion matrix for model with image augmentation and ResNet 50 and Image resolution of 480×360 on CamVid dataset

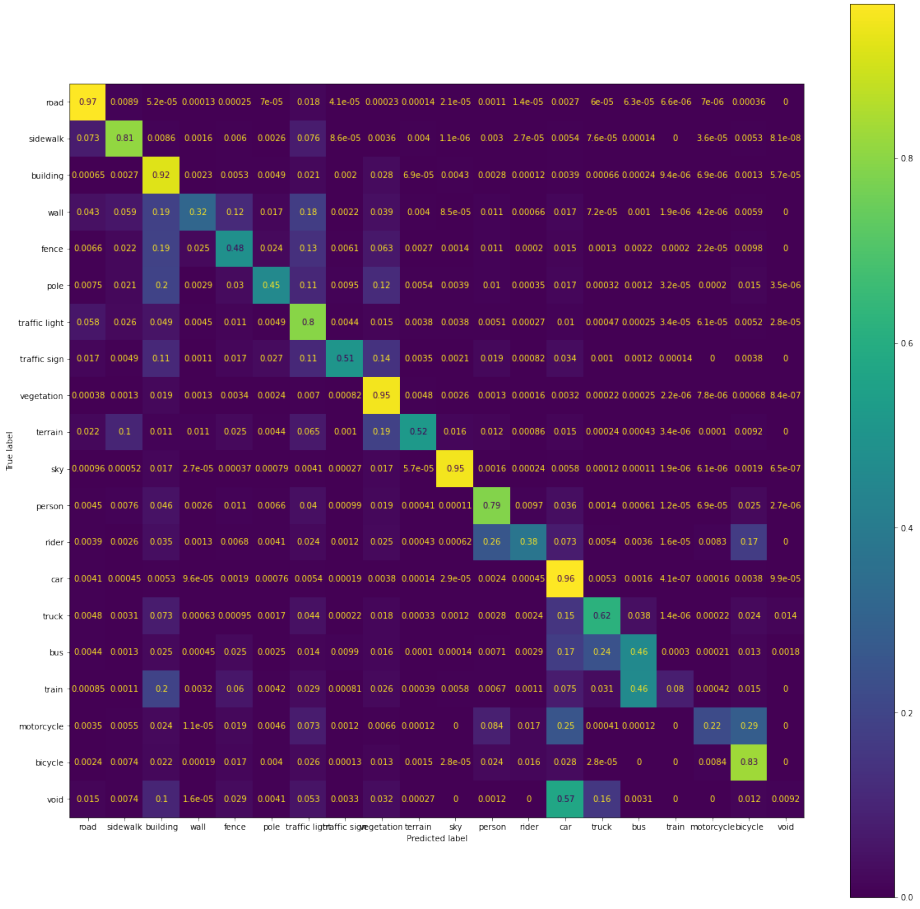

Figure B.8: Confusion matrix for model with ResNet 18 and image resolution of 1024×512 trained on Cityscapes dataset

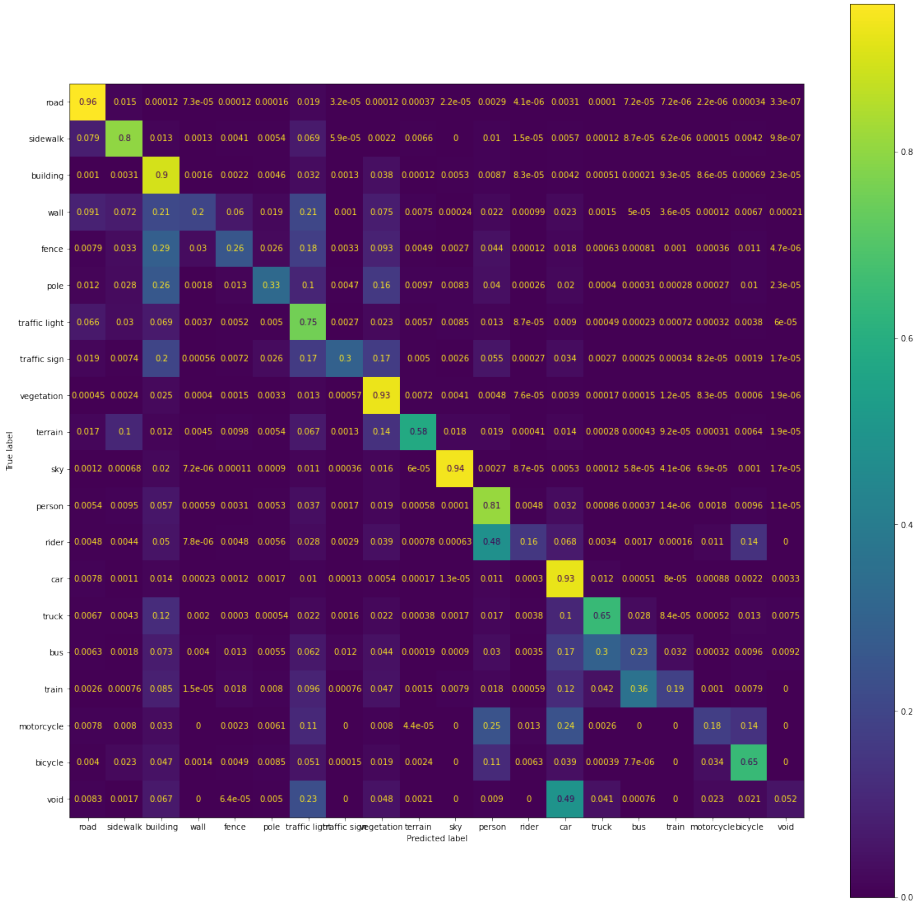

Figure B.9: Confusion matrix for model with ResNet 18 and image resolution of 512×256 trained on Cityscapes dataset

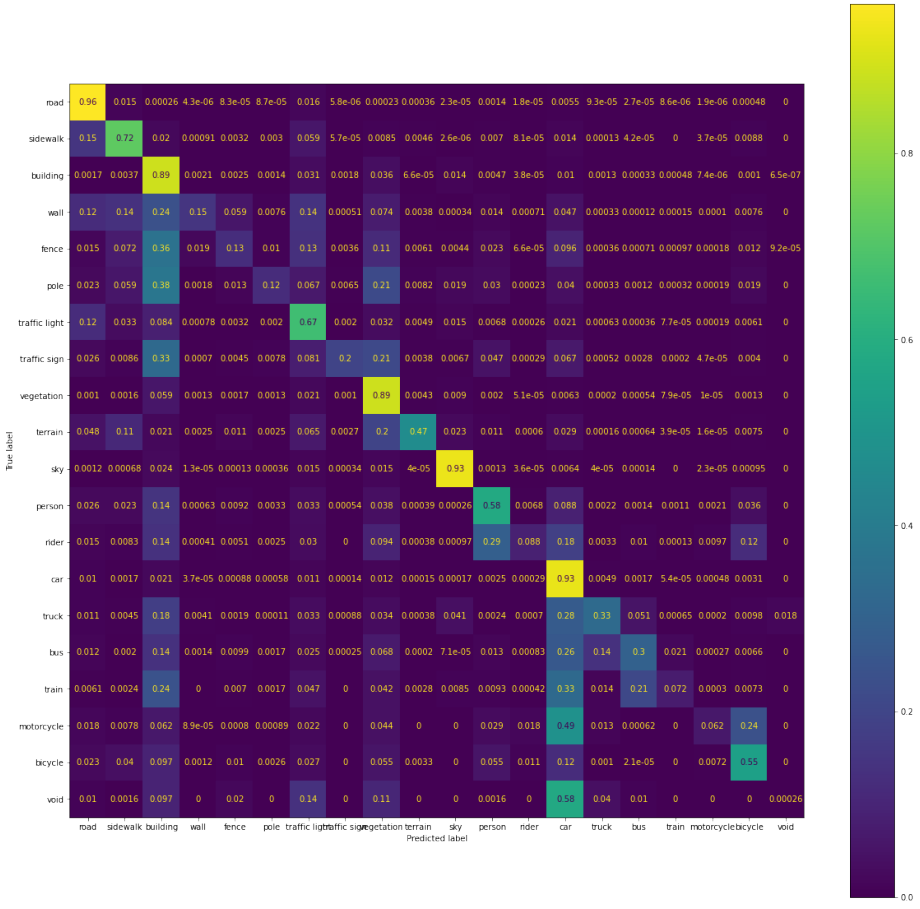

Figure B.10: Confusion matrix for model with ResNet 18 and image resolution of 256×128 trained on Cityscapes dataset

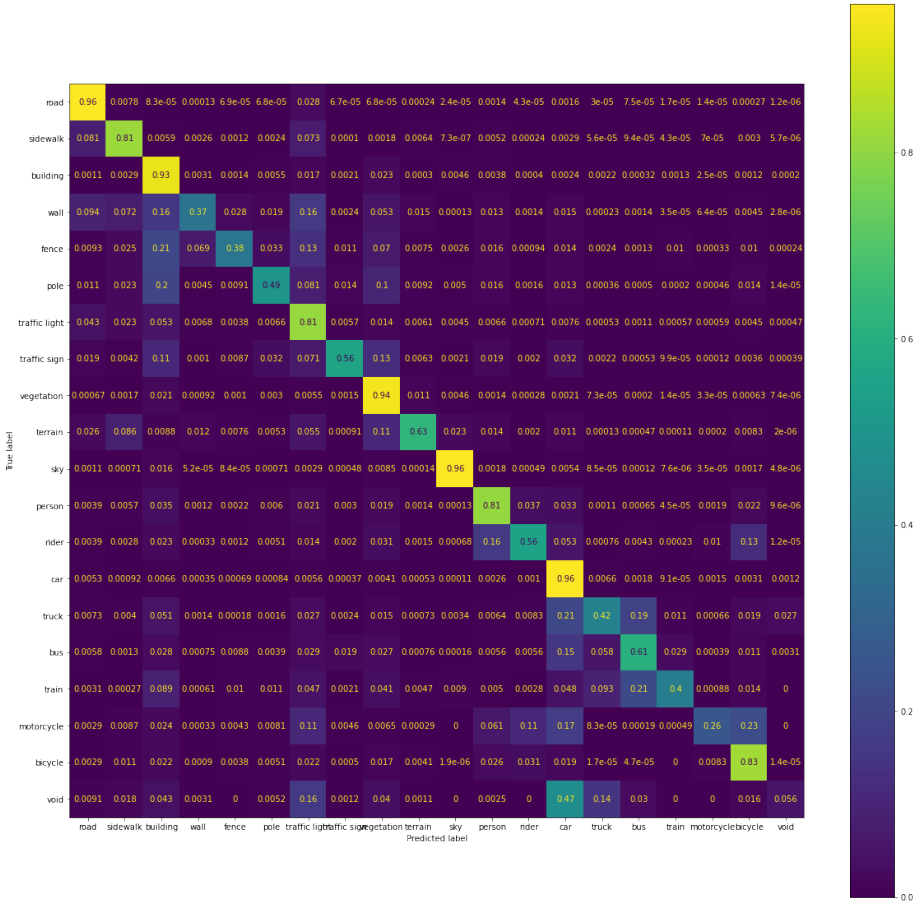

Figure B.11: Confusion matrix for model with ResNet 50 and image resolution of 1024×512 trained on Cityscapes dataset

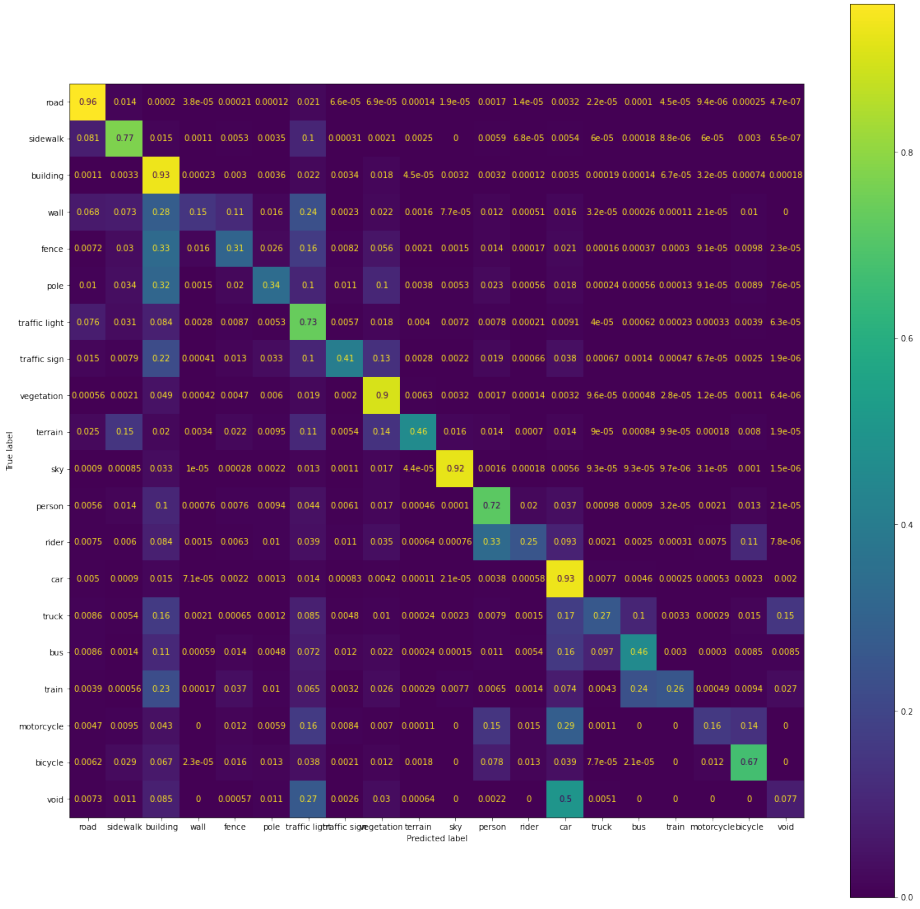

Figure B.12: Confusion matrix for model with ResNet 50 and image resolution of 512×256 trained on Cityscapes dataset

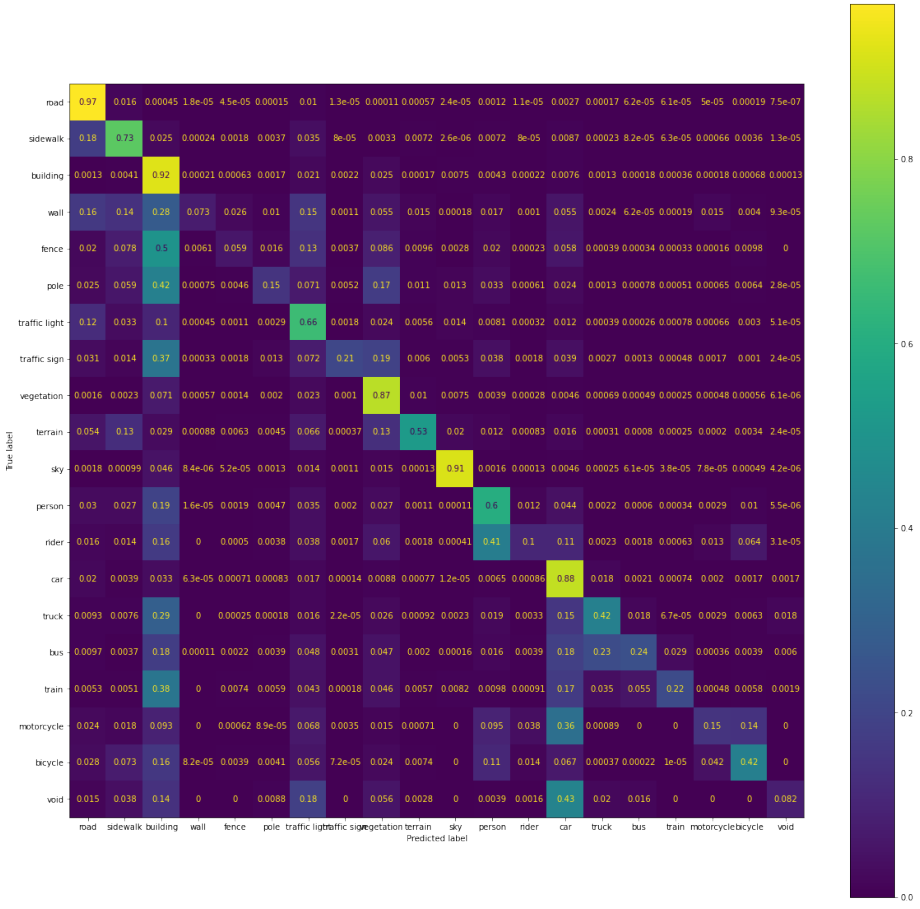

Figure B.13: Confusion matrix for model with ResNet 50 and image resolution of 256×128 trained on Cityscapes dataset

Appendix C

Classification report

Table C.1: Classification report for model with ResNet 18 and image resolution of 960×720

	precision	recall	f1-score	support
Sky	0.93	0.97	0.95	991348
Building	0.88	0.96	0.92	1346548
Column_Pole	0.54	0.2	0.29	60990
Road	0.94	0.98	0.96	1642361
Sidewalk	0.9	0.92	0.91	395195
Tree	0.87	0.86	0.87	615402
Sign	0.83	0.38	0.52	6468
Fence	0.8	0.73	0.77	98187
Car	0.88	0.91	0.9	233632
Pedestrian	0.65	0.36	0.46	42422
Bicyclist	0.92	0.67	0.78	28045
Void	0.8	0.58	0.67	630602
accuracy			0.9	6091200
macro avg	0.83	0.71	0.75	6091200
weighted avg	0.89	0.9	0.89	6091200

Table C.2: Classification report for model with ResNet 50 and image resolution of 960×720

	precision	recall	f1-score	support
Sky	0.95	0.97	0.96	15905564
Building	0.94	0.95	0.94	21562128
Column_Pole	0.71	0.52	0.6	971275
Road	0.96	0.98	0.97	26223662
Sidewalk	0.94	0.92	0.93	6322324
Tree	0.88	0.92	0.9	9857465
Sign	0.82	0.51	0.63	104349
Fence	0.9	0.77	0.83	1570224
Car	0.89	0.93	0.91	3737055
Pedestrian	0.78	0.62	0.69	678905
Bicyclist	0.9	0.88	0.89	448973
Void	0.81	0.72	0.76	10077276
accuracy			0.92	97459200
macro avg	0.87	0.81	0.84	97459200
weighted avg	0.92	0.92	0.92	97459200

Table C.3: Classification report for model with ResNet 18 and Image resolution of 480 × 360

	precision	recall	f1-score	support
Sky	0.96	0.95	0.96	3968442
Building	0.91	0.95	0.93	5385849
Column_Pole	0.57	0.39	0.46	242489
Road	0.96	0.97	0.96	6571544
Sidewalk	0.94	0.91	0.92	1580256
Tree	0.84	0.93	0.88	2461101
Sign	0.71	0.49	0.58	26099
Fence	0.83	0.71	0.76	392633
Car	0.91	0.88	0.9	934526
Pedestrian	0.71	0.47	0.57	169604
Bicyclist	0.82	0.91	0.86	112210
Void	0.78	0.67	0.72	2520047
accuracy			0.91	24364800
macro avg	0.83	0.77	0.79	24364800
weighted avg	0.91	0.91	0.91	24364800

Table C.4: Classification report for model with ResNet 50 and image resolution of 480 × 360

	precision	recall	f1-score	support
Sky	0.96	0.95	0.96	3968442
Building	0.91	0.95	0.93	5385849
Column_Pole	0.57	0.39	0.46	242489
Road	0.96	0.97	0.96	6571544
Sidewalk	0.94	0.91	0.92	1580256
Tree	0.84	0.93	0.88	2461101
Sign	0.71	0.49	0.58	26099
Fence	0.83	0.71	0.76	392633
Car	0.91	0.88	0.9	934526
Pedestrian	0.71	0.47	0.57	169604
Bicyclist	0.82	0.91	0.86	112210
Void	0.78	0.67	0.72	2520047
accuracy			0.91	24364800
macro avg	0.83	0.77	0.79	24364800
weighted avg	0.91	0.91	0.91	24364800

Table C.5: Classification report for model with ResNet 18 and image resolution of 240 × 180

	precision	recall	f1-score	support
Sky	0.96	0.95	0.96	3968442
Building	0.91	0.95	0.93	5385849
Column_Pole	0.57	0.39	0.46	242489
Road	0.96	0.97	0.96	6571544
Sidewalk	0.94	0.91	0.92	1580256
Tree	0.84	0.93	0.88	2461101
Sign	0.71	0.49	0.58	26099
Fence	0.83	0.71	0.76	392633
Car	0.91	0.88	0.9	934526
Pedestrian	0.71	0.47	0.57	169604
Bicyclist	0.82	0.91	0.86	112210
Void	0.78	0.67	0.72	2520047
accuracy			0.91	24364800
macro avg	0.83	0.77	0.79	24364800
weighted avg	0.91	0.91	0.91	24364800

Table C.6: Classification report for model with ResNet 50 and image resolution of 240 × 180

	precision	recall	f1-score	support
Sky	0.95	0.95	0.95	991348
Building	0.89	0.95	0.92	1346548
Column_Pole	0.48	0.24	0.32	60990
Road	0.94	0.97	0.96	1642361
Sidewalk	0.9	0.92	0.91	395195
Tree	0.83	0.92	0.87	615402
Sign	0.72	0.42	0.53	6468
Fence	0.82	0.71	0.76	98187
Car	0.87	0.91	0.89	233632
Pedestrian	0.61	0.39	0.48	42422
Bicyclist	0.75	0.83	0.79	28045
Void	0.78	0.59	0.67	630602
accuracy			0.9	6091200
macro avg	0.8	0.73	0.75	6091200
weighted avg	0.89	0.9	0.89	6091200

Bibliography

[1] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, "Recsys challenge 2018: Automatic music playlist continuation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, pp. 527–528, 2018.

[2] B. Brost, R. Mehrotra, and T. Jehan, "The music streaming sessions dataset," in *Proceedings of the 2019 Web Conference*, ACM, 2019.

[3] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, M. Jagersand, and H. Zhang, "A comparative study of real-time semantic segmentation for autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 587–597, 2018.

[4] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, *et al.*, "Speeding up semantic segmentation for autonomous driving," 2016.

[5] I. Papadeas, L. Tsochatzidis, A. Amanatiadis, and I. Pratikakis, "Real-time semantic image segmentation with deep learning for autonomous driving: A survey," *Applied Sciences*, vol. 11, no. 19, p. 8802, 2021.

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*, pp. 3354–3361, IEEE, 2012.

[7] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

[8] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[9] J. Jin, A. Fatemi, W. M. P. Lira, F. Yu, B. Leng, R. Ma, A. Mahdavi-Amiri, and H. Zhang, "Raidar: A rich annotated image dataset of rainy street scenes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2951–2961, 2021.

[10] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2174–2182, 2017.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[12] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, pp. 1520–1528, 2015.

[13] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[14] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[15] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[16] "Pixellabeldatastore."

[17] M. Siam, S. Valipour, M. Jagersand, and N. Ray, "Convolutional gated recurrent networks for video segmentation," in *2017 IEEE international conference on image processing (ICIP)*, pp. 3090–3094, IEEE, 2017.

[18] D. Nilsson and C. Sminchisescu, "Semantic video segmentation by gated recurrent flow propagation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6819–6828, 2018.

[19] D. G. Nurdialit, "Image colorization," Apr 2021.

[20] Wikipedia contributors, "Grayscale — Wikipedia, the free encyclopedia," 2021. [Online; accessed 5-February-2022].

[21] S. Anwar, M. Tahir, C. Li, A. Mian, F. S. Khan, and A. W. Muzaffar, "Image colorization: A survey and dataset," *arXiv preprint arXiv:2008.10774*, 2020.

[22] H. Caesar, J. Uijlings, and V. Ferrari, "Coco-stuff: Thing and stuff classes in context," in *Computer vision and pattern recognition (CVPR), 2018 IEEE conference on*, IEEE, 2018.

[23] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, (Cambridge, MA, USA), p. 487–495, MIT Press, 2014.

[24] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[25] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[27] Z. Cheng, Q. Yang, and B. Sheng, "Deep colorization," in *Proceedings of the IEEE international conference on computer vision*, pp. 415–423, 2015.

[28] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European conference on computer vision*, pp. 649–666, Springer, 2016.

[29] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, "Scribbler: Controlling deep image synthesis with sketch and color," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5400–5409, 2017.

[30] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification," *ACM Trans. Graph.*, vol. 35, jul 2016.

[31] P. Vitoria, L. Raad, and C. Ballester, "Chromagan: Adversarial picture colorization with semantic class distribution," in *The IEEE Winter Conference on Applications of Computer Vision*, pp. 2445–2454, 2020.

[32] C. M. Staff, "Autonomous vehicles could benefit health if cars are electric shared, year=2020, url=https://cvmbs.source.colostate.edu/autonomous-vehicles-could-benefit-health-if-cars-are-electric-shared/, note=accessed: 2022-02-24."

[33] DARPA, "Defense advanced research project agency urban challenge, year=2007, url=https://www.darpa.mil/about-us/timeline/darpa-urban-challenge, note=accessed: 2022-02-24."

[34] T. Litman, "Autonomous vehicle implementation predictions implications for transport planning," 2022. accessed: 2022-02-24.

[35] NHTSA, "National highway traffic safety administration crash report," 2020. accessed: 2022-02-24.

[36] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818, 2018.

[37] Y. Heryadi, E. Irwansyah, E. Miranda, H. Soeparno, K. Hashimoto, *et al.*, "The effect of resnet model as feature extractor network to performance of deeplabv3 model for semantic satellite image segmentation," in *2020 IEEE Asia-Pacific Conference on Geoscience, Electronics and Remote Sensing Technology (AGERS)*, pp. 74–77, IEEE, 2020.

[38] C. S. Perone, E. Calabrese, and J. Cohen-Adad, "Spinal cord gray matter segmentation using deep dilated convolutions," *Scientific reports*, vol. 8, no. 1, pp. 1–13, 2018.

[39] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[40] H. Lamba, "Understanding semantic segmentation with unet," Feb 2019.

[41] I. Ahmed, M. Ahmad, F. A. Khan, and M. Asif, "Comparison of deep-learning-based segmentation models: Using top view person images," *IEEE Access*, vol. 8, pp. 136361–136373, 2020.

[42] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009. Video-based Object and Event Analysis.

[43] "Papers with code - improving semantic segmentation via video propagation and label relaxation."

[44] F. Ramzan, M. U. G. Khan, A. Rehmat, S. Iqbal, T. Saba, A. Rehman, and Z. Mehmood, "A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks," *Journal of medical systems*, vol. 44, no. 2, pp. 1–16, 2020.

[45] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "License — cityscapes dataset." `https://www.cityscapes-dataset.com/license/`. [Online; accessed 14-April-2022].

[46] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[47] D. Becker, "Cityscapes image pairs— kaggle." https://www.kaggle.com/datasets/dansbecker/cityscapes-image-pair. [Online; accessed 14-April-2022].

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[49] A. Rastogi, "Resnet50," Mar 2022.

[50] M. Cordts, "cityscapesscripts/cityscapesscripts/helpers/labels.py." https://github.com/mcordts/cityscapesScripts/blob/master/cityscapesscripts/helpers/labels.py. [Online; accessed 14-April-2022].

[51] K. Team, "Keras documentation: Multiclass semantic segmentation using deeplabv3+."

[52] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.