Semantic Segmentation Models: A Comprehensive Review and Analysis

Federico Sandrinelli, Francesco Pittorino

***Abstract*—Semantic Segmentation is a challenging task in computer vision with numerous potential application, especially in the field of robotics and automation. Numerous models have been proposed to tackle this problem. The purpose of our work is to provide an overview of some cornerstone model in the literature with a focus on architectural features, performance and complexity. All the models are trained and tested on images related to the field of automatic driving.**

I. Introduction

Semantic image segmentation is the process of assigning each pixel of an image to a certain class. It can be seen as the generalization of object recognition, where the goal is to spot particular object in different context, semantic segmentation, instead, focuses on the image as a whole.

Semantic segmentation is needed in various fields, such as medicine, robotics and automotive, in particular for self-driving cars. To build autonomous agents that take actions in the real physical world, it is essential for them to understand their surrounding context with confidence. This can be achieved by the means of cameras whose captured images can be processed by image segmentation algorithms.

Since the rise of deep Convolutional Neural Networks (CNN) and the publication of open datasets such as ImageNet several models have been proposed to tackle the image segmentation task. In our work we aim to provide a comparison among some cornerstone models in the literature. The purpose of our paper is not to improve existing models performances by changing their architecture or by fine-tuning training parameters, but rather to asses their behaviour in the same setting. We will use the CamVid dataset to benchmark performance. Furthermore, we will discuss model complexity by analyzing size and inference speed.

The models we choose are U-Net, Pyramid Scene Parsing Networks, Feature Pyramid Networks, DenseASPP and the Transformer-based MaskFormer model. They will be evaluated in terms of performance on the selected dataset (mean Intersection-over-Union and mean F1-score) and complexity in terms of size and inference time.

The paper is organized as follows: in section II we describe the dataset while the models are described in depth in section III; in section IV is described the training and the data augmentation pipeline we used. In V is described the MaskFormer architecture and its fine-tuning. Results and conclusions are respectively in section VI and VII.

II. Dataset

The dataset we choose to benchmark the selected models is the CamVid dataset, introduced by Brostow in [1]. It is a collection of 701 images extracted from video recordings of a driving session from the perspective of an automobile. The dataset provides per-pixel annotations as ground truth for 32 different classes. The dataset is divided in three splits in the following way:

- Training: 367 samples;
- Validation: 101 samples;
- Test: 233 samples.

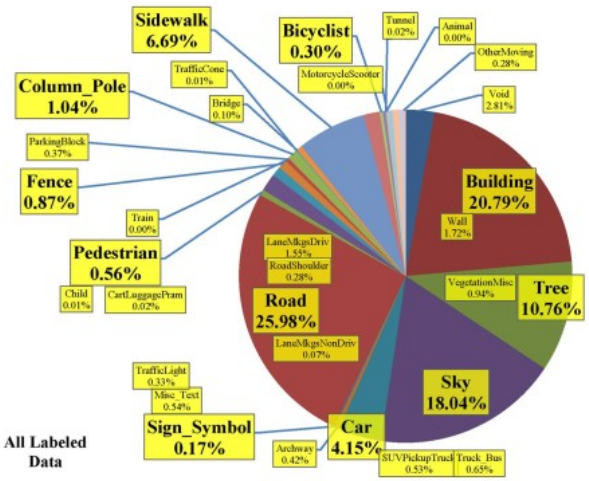


Fig. 1: Class distribution in the CamVid dataset. Image from [1].

The main issue of the dataset is that it is highly unbalanced, most of the available classes in fact appear in less than 1% of the samples, as can be seen in Figure 1.

For this reason we considered a smaller set of classes in our experiments. To train the models described in section III we kept six classes plus the background, which is everything that does not fit in the selected classes. The remaining classes were: 'sky', 'road', 'pavement', 'car', 'pedestrian' and 'bicyclist'. The choice was arbitrarily, we selected the classes representative of the principal road users and also classes that define the scenario (sky, road, pavement). For what concerns the fine-tuning of the MaskFormer we decided to keep more classes to increase the complexity of the task and see how the model responded. To the previously mentioned classes we added 'building', 'pole', 'sign symbol' and 'fence'. In Fig. 2 it is possible to see a sample from the dataset together with the ground truth segmentation map.

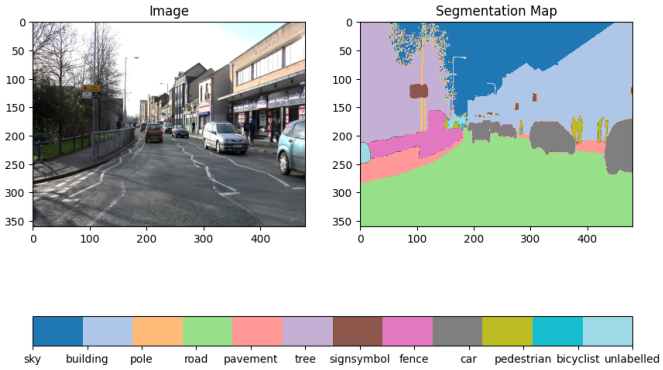


Fig. 2: Sample from the CamVid dataset.

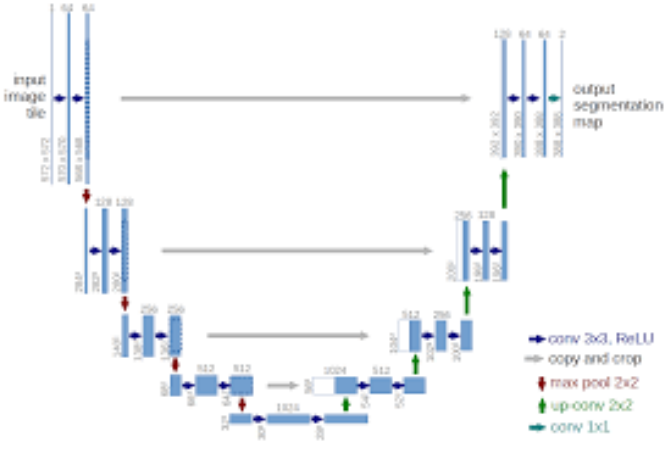


Fig. 3: Orinigal UNet architecture. Image from [4]

.

III. Models

The Segmentation Models [2] library provides an easy way to experiment with different segmentation models. It allows to choose from four different models, U-Net, PSP-Net, FPN-Net and LinkNet and a variety of, possibly pre-trained, backbones. We decided to use U-Net, PSP-Net and FPN-Net for our work. Furthermore, we implemented in TensorFlow the DenseASPP model [3] proposed by Yang.

A. U-Net

U-net architecture is one of the most popular deep learning models used for image segmentation tasks. The original architecture was proposed by Ronneberger et al. in [4] in the field of biomedical imagin. In particular, the development was conducted trying to address two main problems: the modest number of available training examples and the need for a really precise localization capability.

U-Net is made up of a contracting path on the left and of an expanding path on the right side.

- The **contracting path** has a quite typical CNN architecture. Two 3x3 unpadded convolutions are applied subsequently, each one followed by a ReLU activation function and a 2x2 max pooling operation with stride 2 for downsampling. Every downsampling step results in twice as many feature channels.
- The **expanding path** is made up by: an upsampling of the feature map, a 2x2 convolution (also known as a "up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The concatenation between same level feature maps is implemented by skip connection, so it doesn't increase the number of parameters of the model

The main idea here is to use upsampling operators to increase the resolution of the output and combine this with high resolution features from the contracting path. In this way the network is able to localize each class with high precision. The expansive path is more or less symmetric to the contracting path, and results in the characteristic u-shaped architecture.

B. Pyramid Scene Parsing Networks

Pyramid Scene Parsing Networks (PSPNet) are a CNN architecture proposed for scene analysis by Zhao et al in [5]. In image segmentation, a lot of mistakes are partially or fully associated with the lack of capability to understand the contextual information at a global scale. For instance there exist classes whose presence in the scene is correlated. Scene parsing performance can therefore be significantly increased by using a deep network with an appropriate global-scene-level prior. The PSPNet relies on the use of the Pyramid Pooling Module, which is applied to the feature map extracted by the CNN backbone architecture. In the Pyramid Pooling Module, pooling operations with windows of different sizes are applied. In the implementation used, four levels of pooling are employed:

- global pooling on the entire image, producing a single value,
- pooling on subdivisions 2×2, producing 4 values.
- pooling on subdivisions 3×3, producing 9 values.
- pooling on subdivisions 6×6, producing 36 values.

Notice that the pooling kernel size are defined as N/i where N is the width/height of the image while i is the number of subdivision for that pooling operation. Each pooling result is then re-scaled to the original size of the input feature map by bilinear interpolation. The outputs are concatenated with the original feature map. A representation of PSPNet can be seen in Fig. 4.

C. Feature Pyramid Networks

Feature Pyramid Networks (FPNs) are a feature extractor architecture originally proposed for object detection by Lin et al. in [6] that can be easily adapted to perform image segmentation tasks. The goal of the authors was to exploit the capacity of feature pyramids to represent object at different scales but avoiding the computational burden of computing image pyramids. To do so they exploited the intrinsic property of CNNs to compute an in-network feature hierarchy of feature

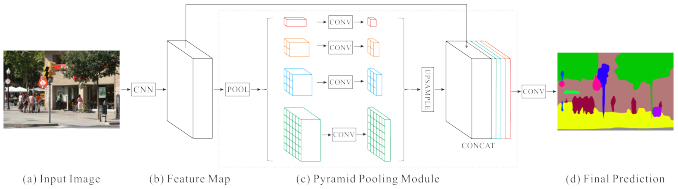


Fig. 4: PSPNet architecture. Image from [5]

.

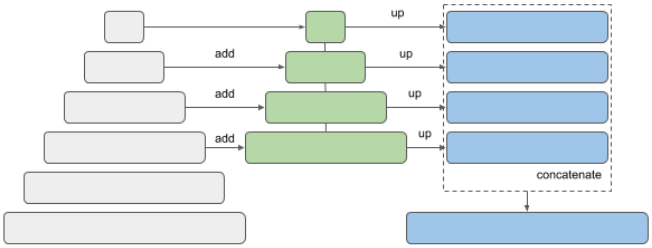


Fig. 5: High level representation of FPNs. The backbone is on the left, while the up-sampling module is in the middle. The output of the FPN are the features represented in blue.

maps of different spatial resolutions. As a result they proposed an architecture that combines the representations produced by a CNN at different scales which is invariant with respect to the choice of the backbone.

From an high level perspective FPNs are composed by two modules:

- a **bottom-up pathway**: the backbone used for feature extraction which produces feature maps with increasingly small resolution and higher semantic;
- a **top-down pathway**: the up-sampling pipeline of the feature map produced by the convolutional network to produce representation with both strong semantic and high resolution.

The two modules works together to produce pyramidal features. The intermediate representations produced in the bottom-up pathway are summed to the relative feature map produced by the top-down pathway by lateral connections. 3x3 convolutions are applied to the combined feature maps to produce the final feature maps used for the task. An high level representation of FPNs can be seen in Fig. 5.

D. DenseASPP

The DenseASPP architecture proposed by Yang [3] was inspired by ASPP network by Chen [7]. Cheng proposed to combine feature maps produced by atrous convolution with different dilation rates (d) to encode multi-scale information and build stronger classifiers.

DenseASPP follows this approach while tackling the challenge of image segmentation for high resolution images in driving scenes. Those images are challenging because they could contain elements at different scale levels (e.g. a pedestrian close to the car who looks bigger than a truck faraway from the camera). This requires to use atrous convolution with

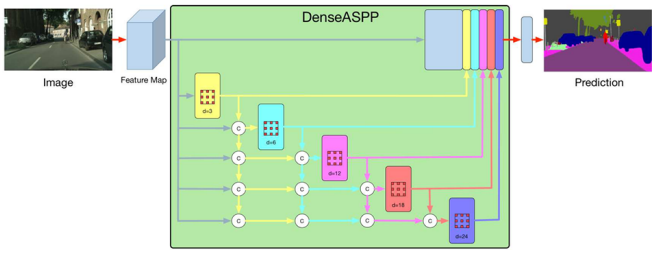


Fig. 6: DenseASPP architecture. Image from [3]

.

high dilation rates, however as d increases it becames more and more inexpressive. To this end, the authors proposed an architecture composed by a base model (the backbone) and a series of atrous convolutional layers with different dilation rates. The output of each atrous convolution and the feature map produced by the backbone are concatened to produce the feature vector used for segmentation. An illustration of DenseASPP can be seen in Fig. 6.

IV. Training

All the models were trained under the same conditions in a Google Colab environment with access to L4 GPUs. We use Adam optimizer with 0.0001 learning rate on batches of 8 images for 40 epochs.

All the models implemented with the Segmentation Models library were trained end-to-end with two different backbones with pre-trained weights from ImageNet: DenseNet121 and ResNet101. For the DenseASPP network instead we only made use of DenseNet121 as backbone, like in the original paper.

A. Loss Function

As loss function, we used a combination of the Dice loss and the binary focal loss.

The Dice loss is a common choice for image segmentation tasks because it directly optimizes for the overlap between the predicted segmentation and the ground truth. The Dice coefficient, also known as the Sørensen–Dice index, measures the similarity between two sets. For binary segmentation, the Dice coefficient is defined as follows:

$$\text{Dice Coefficient} = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

where A represents the set of predicted pixels, and B represents the set of ground truth pixels. The Dice loss is derived from the Dice coefficient and is given by:

$$\text{Dice Loss} = 1 - \frac{2 \sum_i p_i g_i}{\sum_i p_i + \sum_i g_i}$$

Here, p_i is the predicted probability for pixel i, and g_i is the ground truth label for pixel i. This loss function encourages maximizing the overlap between the predicted and ground

truth pixels while minimizing the number of false positives and false negatives.

The binary focal loss is instead designed to address the issue of class imbalance by giving more focus to hard-to-classify examples. This is particularly important in cases where some classes are underrepresented. The binary focal loss is defined as:

$$\text{Focal Loss}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t)$$

where p_t is the model's estimated probability for the true class, α_t is a balancing factor for the class, and γ is a focusing parameter that adjusts the rate at which easy examples are down-weighted. For binary classification, p_t is defined as:

$$p_t = \begin{cases} p & \text{if } y = 1, \\ 1-p & \text{otherwise.} \end{cases}$$

The binary focal loss helps to balance the contribution of classes that are easy to classify and those that are hard to classify, ensuring that the model pays more attention to difficult or minority classes.

By combining the Dice loss and the binary focal loss, we aim to leverage the advantages of both loss functions. The combined loss function can be expressed as:

$$\text{Combined Loss} = \lambda \cdot \text{Dice Loss} + (1-\lambda) \cdot \text{Focal Loss}$$

where λ is a weighting factor that balances the contributions of the Dice loss and the binary focal loss. This combined approach allows the model to optimize for both the overlap between predicted and ground truth segments and the accurate classification of underrepresented classes. As a result, the segmentation model becomes more robust and performs better across a variety of challenging scenarios

B. Number of Epochs and Early Stopping

Each of the model was set to train for 40 epochs. It was added an early stopping callback that stops the models training if the loss on the validation set is not getting better after 10 epochs. Then the weights that produced the smallest loss during training are retained.

C. Evaluation Measures

In order to evaluate models performance, we considered the mean F1-Score and the mean Intersection Over Union. F1-Score is computed, for each class, as the harmonic mean of precision and recall, providing a balance between these two metrics. For image segmentation, precision and recall of a class can be defined as follows:

- **Precision**: The ratio of correctly predicted positive pixels to the total predicted positive pixels.
- **Recall**: The ratio of correctly predicted positive pixels to the total actual positive pixels.

Mathematically, precision and recall are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

where:

- TP (True Positives) is the number of correctly predicted positive pixels.
- FP (False Positives) is the number of incorrectly predicted positive pixels.
- FN (False Negatives) is the number of positive pixels that were incorrectly predicted as negative.

The F1 Score is then computed as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

In our study the mean F1-Score is the unweighted average value of F1 score for each class

The Intersection over Union (IoU), also known as the Jaccard Index, instead measures the overlap between the predicted segmentation and the ground truth. IoU is defined as the ratio of the intersection of the predicted and ground truth segmentation masks to their union:

$$\text{IoU} = \frac{|\text{Predicted} \cap \text{Ground Truth}|}{|\text{Predicted} \cup \text{Ground Truth}|} \tag{4}$$

The mean IoU is then the average of the Intersection over Union computed on all the classes.

We decided to use them both since F1-Score provides insight into the model's ability to correctly identify relevant pixels while avoiding irrelevant ones, which is crucial in applications where false positives and false negatives have different implications (e.g., for our dataset, a false negative pedestrian), while IoU offers a direct measure of overall segmentation quality.

D. Data Augmentation

Due to the modest size of our training set we implemented a data augmentation pipeline to prevent overfitting. The pipeline applies several transformations, with some degree of probability, thus it is not deterministic, ensuring a high degree of variability to the training data. The transformations applied to the images are:

- Horizontal Flipping, with probability 0.5;
- Shifting and scaling (up to 50%) with probability 1;
- Padding and Random Cropping are used to ensure that the size of the output image matches the network requirements;
- Additive Gaussian Noise with probability 0.2;
- Perspective transformation with probability 0.5;
- One transformation among CLAHE, Random Brightness and Random Gamma with probability 0.9;
- One transformation among Sharpening, Blurring and Motion Blur with probability 0.9;
- One among Random Contrast and Hue Saturation with probability 0.9.

An example of augmented image can be seen in Fig. 7.

Fig. 7: Example of the application of the data augmentation pipeline (right) to an original image (left).

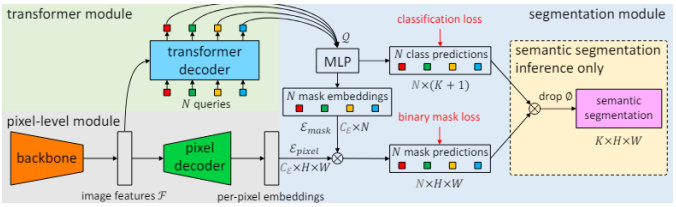


Fig. 8: The MaskFormer architecture. Image from [10].

V. Vision Tranformers

Transformers [8] have become the standard architecture for Natural Language Processing tasks since their introduction in 2017. However, the use of the attention mechanism in computer vision tasks remained marginal for quite a few years. In 2020, however, inspired by the success of the architecture, Dosovitskiy et al. introduced the Vision Transformer (Vit) [9]. The core of their work was applying the transformer architecture directly to images, they did so by splitting images in patches of 16x16, treating them as tokens in NLP tasks. The patches are flattened, linearly projected and a positional encoding is added to each of them. The rest of the ViT architecture is exactly the one of a standard Transformer model. By making use of a huge amount of training data ViT achieved state of the art performance on a variety of datasets and tasks.

A. MaskFormer

In the context of image segmentation Cheng et al. proposed the MaskFormer [10] a new model based on Transformers that provides a unified framework for semantic, instance, and panoptic segmentation tasks. In their work they observed that "mask classification is sufficiently general to solve both semantic- and instance-level segmentation tasks in a unified manner using the exact same model, loss, and training procedure" (p. 1 of [10]). Their major contribution was to show that semantic segmentation can be formulated as a mask classification problem and to propose an inference strategy that provides a task-dependent output.

The MaskFormer architecture is composed of three main modules:

- **Pixel-level module**: it consist of a backbone that produces a low resolution feature map and of a decoder that upsample such feature map to generate per pixel embedding. The authors used a ResNet101 based encoder and a FPN decoder.
- **Transformer module**: standard transformer decoder that uses images features and learnable positional embeddings to compute per-segment embeddings.
- **Segmentation module**: performs classification with a linear classifier and a softmax activation on the per-segment embeddings, producing class probability predictions. Mask embeddings are generated from per-segment embeddings. Mask predictions are obtained via dot product between mask embeddings and per-pixel embeddings.

A representation of the MaskFormer architecture can be seen in Fig. [10].

B. Fine Tuning

In our experiment we fine tuned the MaskFormer Swin Base implementation of the MaskFormer architecture, following a tutorial by Nielse Rogge on GitHub [11]. As mentioned in Section II we fine tuned the model on the CamVid dataset on 11 classes: 'sky', 'building', 'pole', 'road', 'pavement', 'tree', 'signsymbol', 'fence', 'car', 'pedestrian' and 'bicyclist'. We used an higher number of classes with respect to the one used for models discussed in Section III to challenge the model.

We trained the model on an L4 GPU in Google Colab and because of the computational resources needed to train a model of this kind we only trained it for 15 epochs using a learning rate of 0.00005 and Adam optimizer, without further exploring other training configurations. Despite the naive approach the model exhibited great flexibility adapting to the new dataset. In just 15 epochs it reached a MeanIoU of 79.6% on the validation set with a simple data augmentation pipeline.

Results will be discussed in a wider perspective in the next section.

VI. Results

In this section we will discuss the performance of the models we described in section III and of the MaskFormer. We will compare the model performance on the validation set and see how the best one behaves on the CamVid test set. Moreoveor, we will discuss complexity in terms of number of parameters, size (MB) and inference time to get a comprehensive view of the of models under examination.

A. Performance

U-Net, PSP, FPN and DenseASPP were trained using only a subset of the available classes, which is different from the one used for the MaskFormer, thus, their performance are not directly comparable.

The performances of the traditional convolutional models on the validation set are reported in Table I.

The FPN with the ResNet101 backbone achieves the highest Mean IoU and Mean F1-score on the validation set, respectively 80.4% and 87.1%. Its performances were evaluated on the CamVid test set where it achieved a Mean IoU of 73.4% and a Mean F1-Score of 79.9%.

Models	Loss	Mean IoU	Mean F1
Densenet121			
Unet	0.15832	0.79962	0.86142
PSPnet	0.20622	0.7614	0.82729
FPN	0.15488	0.79915	0.86755
DenseASPP	0.38218	0.59854	0.68521
Resnet101			
Unet	0.20726	0.74939	0.82479
PSP	0.24814	0.7215	0.79111
FPN	**0.15062**	0.**8042**	**0.87135**

TABLE I: Performance of models described in section III on the validation set

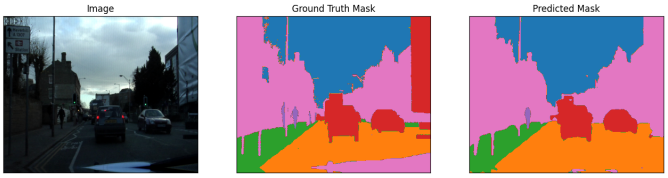


Fig. 9: An example of segmentation performed by the FPN model on a sample from the CamVid test set.

The application of FPN model on a sample image can be seen in Fig. 9.

The MaskFormer model achieved a 79.6% Mean IoU on the validation set and a 72.8% Mean IoU on the test set. Just like in the case of FPN the performances on the test split are way poorer than the performances on the validation split which seems to be way more challenging for the models.

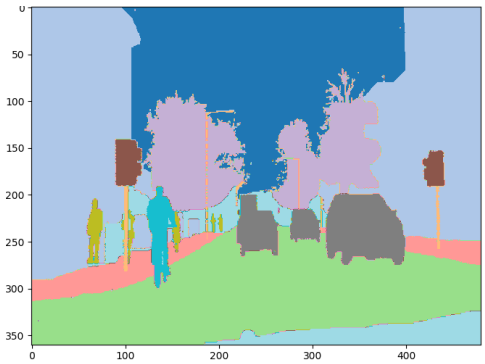
If we want to compare the tests performed on the traditional models like FPN with the test with the MaskFormer it is important to keep in mind that the set of classes considered differs, as explained in section II. For this reason, even if the Mean IoU scores can not be directly compared we can say that the transformer based model is showing its strength, achieving roughly the same score of FPN on a more complex task. An example of inference with the MaskFormer model is given in Fig. 10.

B. Complexity

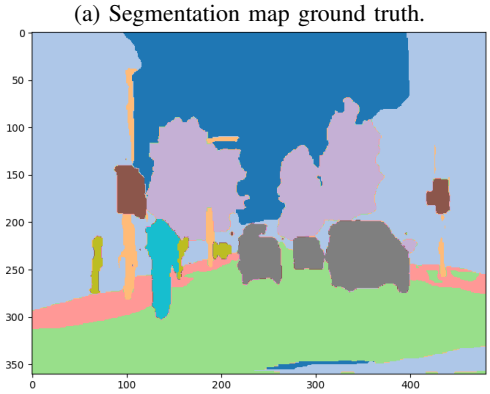
We evaluated the complexity of the tested models considering: the number of parameters, the size in MB and the time required to perform inference on the CamVid test set on a CPU. Measurements for the CNN based models described in section III are reported in Table II.

From the table it is possible to see that the ResNet101 based FPN model is the slowest at inference time while the faster is DenseNet121 implementation of Unet. The smaller model is the PSPnet with DenseNet backbone. This model is also pretty fast at inference, making it the best trade-off in terms of performance and complexity.

Regarding the MaskFormer model, it has more than 101 millions parameters for a total of 392.12 MB, making it the bigger model among all, almost twice the size of the second



(a) Segmentation map ground truth.



(b) Segmentation map prediction.

Fig. 10: Example of segmentation map prediction with the MaskFormer model on a sample of the CamVid test set.

Models	Number of Parameters	Size (MB)	Inference Time (s)
Densenet121			
Unet	12145847	46.33	**0.15326**
PSPnet	**3191367**	12.1	2.36693
FPN	9918663	37.84	10.82945
Resnet101			
Unet	**51606336**	198.86	0.16285
PSP	3854032	14.7	1.59952
FPN	45963088	175.34	**13.3669**
DenseASPP	15738695	60.04	3.43895

TABLE II: Complexity measures of CNN based models including number of parameters, size, and inference time.

biggest model (ResNet101-Unet). CPU inference with the MaskFormer model took 12.8 minutes on the CamVid test set. This result is pretty close to the one for FPN with ResNet101 backbone. This makes sense as it is used in the pixel-level module in the MaskFormer model, as detailed in V-A.

VII. Conclusion

We compared different architectures on a semantic segmentation task on the CamVid dataset.

The highest mIoU was achieved by the FPN model with ResNet101 backbone. However, the traditional CNN based models have shown similar performances on the dataset we choose for this benchmark. The complexity analysis shows

that some model (i.e. DenseNet121-PSPnet) are more efficient than others in terms of speed and size while still providing satisfactory results.

The MaskFormer model has proven to be a powerful architecture, capable to adapt to new task with few iterations of fine-tuning. Moreover, inference speed is comparable to the one of simpler models.

REFERENCES

[1] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. xx, no. x, pp. xx–xx, 2008.

[2] P. Iakubovskii, "Segmentation models." https://github.com/qubvel/segmentation$_m odels$, 2019.

[3] M. Yang, K. Yu, C. Zhang, Z. Li, and K. Yang, "Denseaspp for semantic segmentation in street scenes," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3684–3692, 2018.

[4] P. F. Olaf Ronneberger and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[5] X. Q. X. W. J. J. Hengshuang Zhao, Jianping Shi, "Pyramid scene parsing network," 2017.

[6] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[7] L. 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," *CoRR*, vol. abs/1606.00915, 2016.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020.

[10] B. Cheng, A. G. Schwing, and A. Kirillov, "Per-pixel classification is not all you need for semantic segmentation," *CoRR*, vol. abs/2107.06278, 2021.

[11] N. Rooge, "Transformers tutorials." https://github.com/NielsRogge/Transformers-Tutorials.