EFFICIENT SEGMENTATION OF FLOWERS DATASET THROUGH DEEP LEARNING

Yee Wai Ewan Choi

University of Nottingham, School of Computer Science
psyyc9@nottingham.ac.uk

ABSTRACT

The paper aims to address the challenge of performing image segmentation in MatLab using Convolutional Neural Networks (CNNs) via two methods: Reusing an existing segmentation model (DeepLabV3+) and building our own CNN model, with training from scratch. Both methods involved the cleaning of data (deletion of images with no labels) to enhance the models' performance.
The re-trained model achieved a segmentation global accuracy of 96.5%, while the custom model showed a slightly lower global accuracy of 93.9%, demonstrating the effectiveness of customized architectures for specific datasets. These findings underscore the potential of tailored CNNs to significantly improve automated image segmentation tasks.

Index Terms – Semantic Segmentation, Computer Vision, Convolutional Neural Networks

1. INTRODUCTION

Image segmentation is a critical process in computer vision, with applications ranging from medical imaging [1] to autonomous driving, and even satellite imagery. In an image segmentation task, a network assigns a label (class) to each individual pixel of an image.
In the past, image segmentation was handled using a variety of classical image processing techniques [2]. These techniques were based on principles such as colour, texture and boundary detection, and often relied on hand-crafted features and heuristics. Nowadays, however, with vast improvements in technology, most image segmentation tasks are handled via the usage of deep learning, particularly using CNNs. Despite this, current methods often face challenges in many areas, especially with regards to long training times, overfitting and optimisation issues. [3]
This study hence attempts to address some of these challenges through the employment of 2 distinct methods in training a network: adapting and retraining an existing CNN model, and the creation of a custom model from scratch.
By evaluating the performance of these models on a dataset known as the 17 Category Flower Dataset [4] this research aims to uncover insights into the potential improvements in the methods used, and effectiveness of said methods.
The methodologies, results, implications, and future research directions are subsequently detailed, providing a comprehensive overview of the advances and ongoing needs in the field of automated image segmentation.

1.1 METHODOLOGY

We will not look at classical image segmentation techniques in this paper. Instead, we will focus solely on image segmentation via deep learning techniques.
There are 2 goals: using an existing segmentation model to segment the images in our dataset; and building our own CNN model to segment the same dataset. We worked with the Oxford Flower Dataset, 17 class version. The dataset was provided on Moodle and all images came at a size of [256 256 3]. All training was performed on MATLAB R2024a, and on an Apple Silicon MacBook Air M2. The specifications of the computer are provided in the evaluation section.

To preface, the original dataset provided on Moodle did not have labels for every image. So we first had to clean up the dataset and discard images that have no matching labels. To do this, I first generated lists of files without their extensions in both image and label directories with the following command in terminal:

```
ls images_256/*.jpg | xargs -n 1 basename -s .jpg | sort > images.txt
ls labels_256/*.png | xargs -n 1 basename -s .png | sort > labels.txt
comm -23 /images.txt /labels.txt > /non_matching.txt
```

This allowed us to identify the extra images (without labels) in the images folder. I then created a Bash script that automatically deleted the extra files in the image directory to clean up the dataset, leaving us with only 847 images with their matching labels:

```
while IFS= read -r line; do
  rm "${line}.jpg"
done < /tmp/non_matching_files.txt
```

The dataset was also split into training, test and validation sets in a 60%, 20%, 20% ratio, with a custom function used to randomly select each set to prevent bias. The training dataset was augmented through affine transformations, specifically random horizontal translations and reflections to increase the diversity of the training data. The transformations were generated using the randomAffine2d function, ensuring that any spatial adjustments to the images were mirrored accurately with the matching labels, to maintain the consistency of training data. By centralizing the output images through affineOutputView with BoundsStyle='centerOutput', we maintained uniform image dimensions, crucial for consistent model training. This augmentation process was vital in preventing model overfitting, significantly enhancing the model's ability to accurately segment images. [9]

The first method involved us making use of existing segmentation models, adapting and retraining where necessary. The CNN model chosen was DeepLabV3+ with a ResNet-18 backbone. DeepLabV3+ was chosen over other CNN models due to its advanced capabilities in handling multi-scale information via atrous spatial pyramid pooling and atrous convolution [5], making it particularly suited for our complex image dataset. ResNet-18 was chosen as the backbone due to its effective feature extraction capabilities [6] enabled by deep residual learning, which prevents the degradation problem often seen in traditional deep convolutional neural networks.

To train the model, we used the Stochastic Gradient Descent with Momentum (SGDM) optimiser to deal with the saddle points and local minima in our dataset. The initial learn rate was set to 1e-3 and decreased by 0.1 every 6 epochs thereafter under a piecewise learn schedule. This ensures fine-grained updates to the model weights as training progresses, aiding convergence to a better solution. Regularisation techniques applied included momentum at 0.9 to smooth the landscape, and L2 regularisation to penalise large weights, and produce a model that generalises better to unseen data. The mini-batch was set to 64 to maximise efficiency and balance performance, with data shuffling at every epoch to ensure diversity in training examples. Model performance was continuously monitored during training using a validation set. The training-progress plot in MATLAB provided real-time visualization of training and validation loss, alongside accuracy metrics.

After training, the model's performance was evaluated using MATLAB's inbuilt evaluateSemanticSegmentation method. Metrics such as pixel accuracy, mean IoU and class specific accuracy were used to assess the model's effectiveness in accurately segmenting flowers from the background across different images.

A custom loss function was also used to address class imbalance based on the modified cross-entropy formula. The function computes the loss by summing over all classes and only considers labeled (non-NaN) pixels.

$$L = \sum_{\{c=1\}}^{\{M\}} w_c \sum_{\{i\in \Omega\}} t_{\{i,c\}} \, log\big(y_{\{i,c\}}\big)$$

Recognizing the potential for model bias towards more frequently represented classes, we first calculated the total number of pixels associated with each class from our pixel-count table (tbl.PixelCount). We then computed the frequency of each class as the proportion of pixels for that class relative to the total number of pixels across all classes. Given the possibility of encountering classes with zero pixels, which could disrupt further calculations, we preemptively set any zero frequency values to 1e- 6. This adjustment ensured stability in our calculations. The weights for each class were then determined using the formula:

$$classWeights = \frac{1}{frequency}$$

The second method involves us creating our own CNN from scratch. Pre-processing and loading of the dataset was the same as before. The model employed was a custom deep convolutional neural network based on an encoder-decoder architecture, designed specifically for semantic segmentation tasks [5, 7]. The encoder part included successive convolutional layers with increasing filters of sizes 64, 128, and 256, each followed by batch normalization and ReLU activation functions, and max pooling for spatial downsampling. The decoder part used transposed convolutional layers for upsampling, combined with convolutional layers and sigmoid activations to produce segmentation maps. The model outputs a class label for each pixel, specifying whether it belongs to the background or to a flower. The model was then evaluated using the same evaluateSemanticSegmentation method.

Due to the characteristics of the network architecture, there was not a need for the custom loss function or the calculation of class weights for balancing of classes. The architecture emphasises on segmentation through convolution, pooling and upsampling layers, which progressively doubles the number of filters used as the network deepens, which should allow the network to learn detailed features from less represented class. Additionally the use of batch normalisation layer after convolutional layers help to stabilise the learning process which should make the model less reliant on class weight balancing. ReLU activations also contribute to learn complex patterns, hence we felt that there was no need for the class weights.

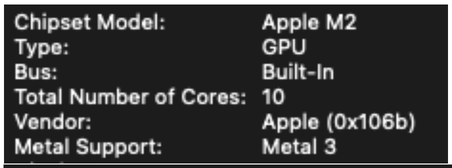
Additionally, when considering factors like the network depth, potential overfitting, and the aforementioned reasons, we hence decided there was no need to make use of the custom loss function nor the class weights calculations.

This time the network was trained using the Adam optimiser with an initial learn rate of 1e-4, which decreased by 0.1 every 6 epochs. Training was done over 8 epochs with mini-batch size of 64 to maximise efficiency and balance performance, with data shuffling at every epoch to ensure diversity in training examples. We used the Adam optimiser over the SGDM optimiser because of lower training times when used in this case. This is probably due to the fact that it has an adaptive learning rate and converges faster. [8]

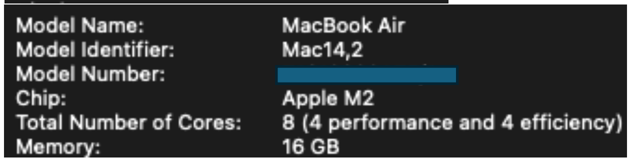
2. EVALUATION

To make results as fair and conclusive as possible, the trainings for both models were ran 5 times, and the average of various metrics were calculated.

All training was performed on an Apple Silicon MacBook Air running the latest version of MacOS (14.4.1 Sonoma). The specifications of the computer are shown below.



```
Chipset Model:          Apple M2
Type:                   GPU
Bus:                    Built-In
Total Number of Cores:  10
Vendor:                 Apple (0x106b)
Metal Support:          Metal 3
```



```
Model Name:             MacBook Air
Model Identifier:       Mac14,2
Model Number:           [redacted]
Chip:                   Apple M2
Total Number of Cores:  8 (4 performance and 4 efficiency)
Memory:                 16 GB
```

To obtain a fair result, training was re-run 5 times across both methods, and the mean runtime was calculated for the pre-trained and our own CNN.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
DeepLabV3+	29:39	34:34	33:00	28:20	27:20	32:28
Our CNN	40:08	42:22	39:09	39:44	40:31	40:16

Table 1: Average Runtimes of the 2 different segmentation models

As we can see, there is a significant difference (approx. 7:48) in the average training times between the DeepLabV3+ model and our own CNN. Run 2 for the DeepLabV3+ model and our CNN model were the slowest. This was because during the runs, other programs were being run on the computer, which ate up system resources which slowed down training time. Additionally, because training was performed on an Apple Silicon computer, MATLAB is unable to train on the GPU (currently not supported). MATLAB currently only supports training on CUDA-enabled Nvidia GPUs, which this computer does not have. Hence this has led to significantly longer training times as compared to if the networks were trained on a Nvidia GPU.

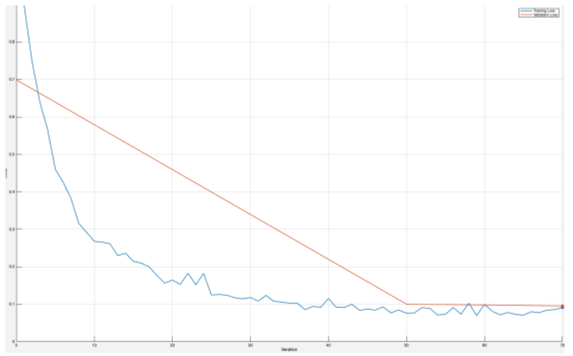
We will now show the average metrics obtained from training across the 5 runs using different models.

	GlobalAccuracy	MeanAccuracy	Mean IOU	WeightedIOU	MeanBF Score
DeepLabv3+	96.5%	96.3%	91.7 %	92.9%	91.4%
CNN	93.9%	93.2%	86.8 %	88.6%	70.4%

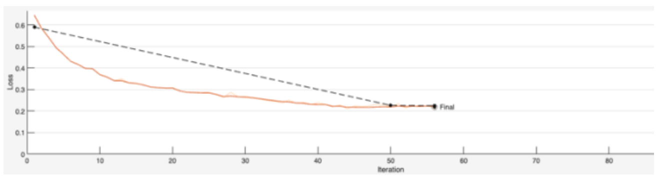
Table 2: Average Class Metrics obtained across 5 runs for both models.

As seen in the table above, the DeepLabV3+ model outperforms our CNN in every metric. This is because DeepLabV3+ uses atrous convolutions and a more complicated architecture (as opposed to our CNN's simpler structure of standard convolutional layers, batch normalization, ReLU activations, and max pooling, followed by a series of transposed convolutional layers for upsampling) which allows it to capture more context while still maintaining the fine details. Additionally, DeepLabV3+ is a model with more depth with complex connections and layers, which allows it to capture and model more features in our dataset, as opposed to our simpler CNN model. There are many other reasons, such as choice of optimiser, class imbalance etc. that could also affect the results.

We will now show the graph of training against validation loss. Because the trends of the graphs produced are very similar, indicating minimal variation, we have only added the graphs from run 4 to provide clear representation of the data while avoiding redundancy.

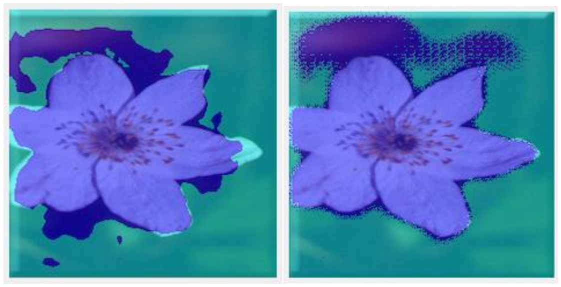


Graph 1: Plot of training against validation loss in run 4 of DeepLabV3+ model.



Graph 2: Plot of training against validation loss in run 4 of our CNN

In graph 2, we can see that training and validation loss start at a lower value than graph 1. This indicates that our model has a better initial performance compared to DeepLabV3+. However this is not indicative of overall performance, as the initial learning rate for DeepLabV3+ is higher than our CNN. Additionally, DeepLabV3+ model is more complex than our CNN, hence the initial difference in loss could simply be attributed to the model learning to optimise. We also used different optimisers (SGDM vs Adam) which can affect how quickly the model converges. This can be seen in the test segmented images produced by the two models.



Figures 1 (Left) and 2 (Right): Sample segmented images from DeepLabV3+ (Left) and our CNN (right).

As we can see, the DeepLabV3+ image is much more precise and captures the fine details around the flower petals and boundaries. The flower region also seems to be a smooth, continuous mask with well-defined edges. The background also seems to be segmented cleanly with sharp boundaries separating the flower from the background.

Our CNN model does manage to segment the flower from the background as well, however the segmentation is course and lacks fine details around the flower petals and boundaries. There are also misclassified pixels and gaps which produce a rough segmentation mask. Though the background is segmented well, there is still some noise around the edges.

From these sample images, we can infer that the DeepLabV3+ model, being a state-of-the-art architecture specifically designed for semantic segmentation tasks, is able to produce more accurate and detailed segmentation masks compared to the CNN created from scratch.
The CNN from scratch, while capable of segmenting the flower and background regions, struggles with capturing fine details and maintaining accurate boundaries which can be attributed to its simpler structure.

3. DISCUSSION

As we can see from the results, it strongly indicates that DeepLabV3+ performs better than our CNN built from scratch. More testing needs to be done, however, as the networks were only trained on this particular dataset – the Oxford Flower Dataset. The models' performance could vary across different types of datasets with different types of images. Hence more testing with other types of data is still required in order to obtain a clearer picture of the performance of DeepLabV3+ model against our CNN from scratch.

4. CONCLUSION

This paper has gone through the selected dataset and methods used to segment said dataset using Deep Learning. Different metrics were used to compare and contrast between using a pre-trained network as compared to creating and using a CNN from scratch. The results has also managed to show some insight into why the pre-trained model performs better at semantic segmentation. However we have also determined that a larger dataset, and more variations of images need to be used to concretely decide that the pre-trained network is superior to the CNN from scratch – we can only deduce their performances based on the dataset used.

11. REFERENCES

[1] Guo, Y. *et al.* (2017) *A review of semantic segmentation using Deep Neural Networks - International Journal of Multimedia Information Retrieval*, *SpringerLink*. Available at: https://link.springer.com/article/10.1007/s13735-017-0141-z

[2] Nilsback, M.-E. and Zisserman, A. (no date) *17 category Flower Dataset*, *Visual Geometry Group - University of Oxford*. Available at: https://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html

[3] Qi, J., Yang, H. and Kong, Z. (2022) *A review of traditional image segmentation methods*, *SPIE Digital Library*. Available at: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12451/124511P/A-review-of-traditional-image-segmentation-methods/10.1117/12.2656653.short?SSO=1

[4] Bandyopadhyay, H. (2021) *Image segmentation: Deep Learning vs traditional [guide]*, *Image Segmentation: Deep Learning vs Traditional [Guide]*. Available at: https://www.v7labs.com/blog/image-segmentation-guide#h5

[5] Chen, L.-C. *et al.* (2018) *Encoder-decoder with atrous separable convolution for Semantic Image segmentation*, *arXiv.org*. Available at: https://doi.org/10.48550/arXiv.1802.02611

[6] *Papers with code - cityscapes test benchmark (real-time semantic segmentation) The latest in Machine Learning*. Available at: https://paperswithcode.com/sota/real-time-semantic-segmentation-on-cityscapes?p=multi-path-segmentation-network

[7] Norelyaqine, A., Azmi, R. and Saadane, A. (2023) *Architecture of deep convolutional encoder-decoder networks for building footprint semantic segmentation*, *Scientific Programming*. Available at: https://www.hindawi.com/journals/sp/2023/8552624/

[8] Wang, B. *et al.* (2024) *On the convergence of Adam under non-uniform smoothness: Separability from SGDM and beyond*, *arXiv.org*. Available at: https://arxiv.org/abs/2403.15146

[9] Ruiz, D.V., Krinski, B.A. and Todt, E. (2019) *A novel data augmentation technique applied to salient ...*, *ANDA: A Novel Data Augmentation Technique Applied to Salient Object Detection*. Available at: https://arxiv.org/pdf/1910.01256