SEMANTIC SEGMENTATION OF CROPS USING DEEPLEARNING

MUHAMMAD KHIZER ALI KHAN - 100063153
HAMMAD HASAN - 100062594
MAHMOUD SAMI ABOUSHANAB - 100060988

Contents

FIGURES

TABLES

Abstract

Recently, the world has witnessed a sudden shift towards automation with the improvement and accessibility of various autonomous systems. We see the manifestation of such systems in our daily lives; from robotic inventory managers in supermarkets to self-driven cars, which we earlier thought would not be possible. This change was made possible through the advent of better computer vision capabilities.

A robust computer vision capability is essential for most automatic systems. Convolutional Neural Networks (CNNs) and Deep learning are highly popular in the field of computer vision. While CNNs are good for Image classification, object detection, and image generation, Deep learning has proven itself as the most effective approach for image segmentation.

Picking crops, such as vegetables and fruits, from the farm is a laborious task, requiring a lot of human effort. Crops are usually grown on trees or on the ground, in clusters, around various positions in the vegetations. To pick the crops, we first require locating their position, identifying which ones are ready for picking, and carefully removing them from their vegetation without damaging it.

In this project, we focus on the *semantic segmentation of tomato, capsicum and chili plants in their respective farm environment*, using a self-developed dataset of images taken by a camera.

The objective of the project is to develop a robust model for semantic segmentation for three different types of crops: Tomatoes, Chillis, and Capsicum in their farm environment. The model would allow us to reduce the human effort by using it to train autonomous system so that I would carry out the exhaustive task of removing ripe crops from their respective vegetation.

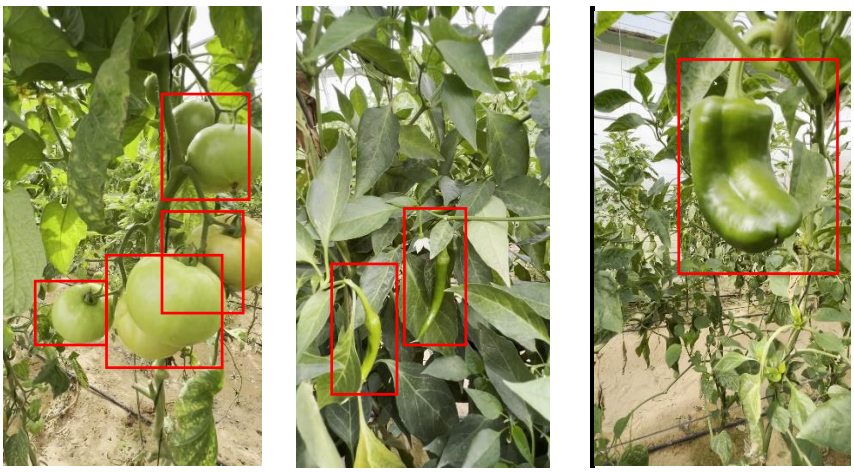


Figure 1 - Images of the three Classes

Literature Review

Semantic Segmentation andi Its Applications

Semantic segmentation involves the classification of each pixel of an image from a set of predefined classes. Basically, the goal is to take an image of size L x W x 3 and out a matric of W x H where each pixel of the output is identified by the number that represents that class.

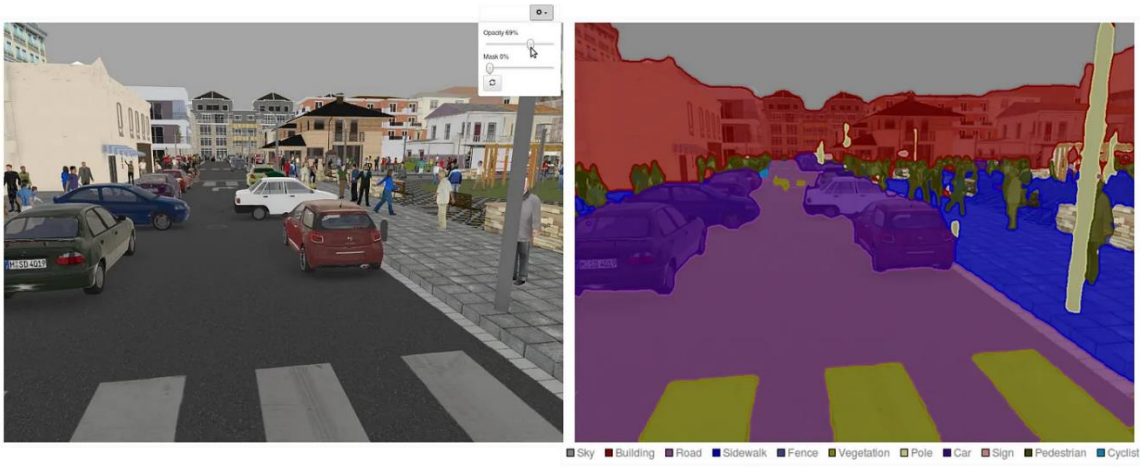


Figure 2 - An example of semantic segmentation [1]

For example, in the above figure, that scene is divided into different segments having the presence of pre-defined classes. In this case, the objects / instances such as Sky, Buildings, Roads, Sidewalks, Fence, Trees, Poles, People, Car and cyclist are separated from each other. Semantic Segmentation is significant application in the following state-of-the-art systems:

a) **Autonomous driving**: Semantic segmentation is used in the development of autonomous driving systems to detect and segment different objects on the road such as vehicles, pedestrians, traffic signs, and road markings. This helps in creating a 3D understanding of the scene and assists in decision-making.

b) **Medical image analysis**: Semantic segmentation is used in medical image analysis to segment different organs, tumors, and other abnormalities in the images. This helps in accurate diagnosis and treatment planning.

c) **Satellite imagery**: Semantic segmentation is used in analyzing satellite imagery to detect changes in land use, monitor natural disasters, and assess the impact of climate change on the environment.

d) **Augmented reality**: Semantic segmentation has wide application in augmented reality to separate different objects of a real environment and display virtual objects on top of the real objects. One such example is the Virtual Environment function

provided in the IKEA app to see the outlook of a selected furniture in your home setting.

Examples mentioned above are only a few examples that are currently in practice, however, with the advances in deep learning, the use of semantic segmentation is expected to develop more complex and innovative applications.

Deep Learning in Agronomics: Importance and Motivation

A semantic segmentation model capable of picking crops from their vegetation provides many benefits over a conventional human picker, such as it reduces the required time and cost r, provides consistency, reduces the dependence on human labor, and results in an overall improvement in the crop yield. However, semantic segmentation in Agronomics' is not limited to these advantages alone. Semantic models can be extended to enhance the capacity that would enable an algorithm to help farming in the following ways:

a) **Crop monitoring**: The growth of crops (size, texture, changes in size, rate of growth etc.) can be predicted by a segmentation model. By segmenting the images, crops can be classified into different classes, such as healthy, ripe or infected. This would help the farmer to identify the areas that require intervention for targeted prevention measures.

b) **Yield prediction**: Segmentation models can be enhanced to predict the yield for the next season, that would be helpful for the farmer to plan for its storage and sale.

c) **Irrigation management**: Water deficiency in the crops can be detected by good semantic segmentation models. By identifying the areas where the crops are more water deficient, farmers can adjust the farming conditions according to the need of those areas.

d) **Soil analysis**: The properties of soil can be evaluated, so that the region with poor soil condition can be treated with relevant fertilizers.

e) **Pest control:** By identifying which areas of the crops are infected by pests, farmers can apply relevant insecticides to reduce the damage and prevent infections in the future.

Overall, semantic segmentation has a lot of many other application in Agronomics which can be used by the farming industry to optimize the yield and maintenance of crops with minimum effort, and at the same time monitoring its quality and health during the yield period.

Nevertheless, semantic segmentation for crops using ground images is a challenging task as it is plagued by problems such as occlusion, uneven illumination, and variability in the appearance (size, shape, color etc) of the same crop. Deep learning methods have the

capability to achieve very high performance by learning extensive features from large-scale dataset.

Most widely used deep learning architecture for semantic segmentation is an encoder-decoder structure[2]. The encoder extracts high level information from the images through down sampling by using a series of pooling and convolutions, whereas the decoder up-samples the images by applying convolutions. The performance of the encoder-decoder network can be improved by combining it with various modules [3], such as skip connections, in which the low-level information of the original images in the encoder part is provided to decoder layers at same of resolution. An example of a skip connection is U-Net model used for the segmentation of medical images [4].

SegNet uses pooling indices from the encoder network to perform non-linear up sampling in the decoder network. This eliminates the need for learning to up-sample and reduces the number of parameters. Hence it is used in semantic segmentation applications to produce useful results with relatively lesser computational resources as compared to other encoder-decoder frameworks [5].

FCN32 network uses a pixel-wise SoftMax loss function that computes the cross-entropy loss for each pixel and each class. This loss function encourages the network to predict the correct class label for each pixel. It has widely been used in the recognition and classification of fruits and vegetables on various datasets, such as Fruit-360 [6].

Dataset

Pictures of Tomatoes, Capsicum and Chili were collected using a camera in their respective farm environments.

Size and characteristics

The model has been trained on a total of 1085 images: 810 images of Tomatoes, 193 images of capsicum, and 82 images of chili. Here, we are dealing with a class imbalance problem, where the size of dataset of all three classes is significantly different from each other. Another peculiar feature about the dataset is that all of the samples are different shades of green, which are very well blended with the environment, which is also mostly green. This is a well-known problem of occlusion, where the instance of a class is difficult to predict.

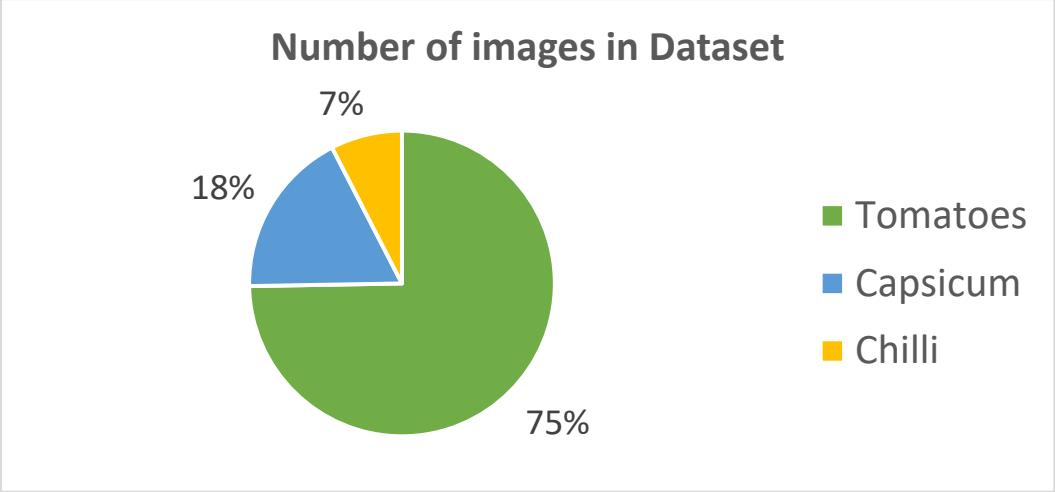


Figure 3 - Pie chart shows the class imbalance

Pre-processing techniques

The pre-processing of the images has been done in the following ways:

a) **Image collection**: Pictures of crops in their natural environment are collected. Pictures are taken in a way that the instances of class is present at multiple positions within the images. Pictures are different angles, magnification, distance, and light intensity to increase the robustness of the model that would be eventually prepared.

b) **Dataset cleaning**: Images that did not have any instance of class present were removed from the dataset.

c) **Image Labeling**: ImageLabeler tool of MATLAB was used to create the mask of the images. This step requires careful creation of boundaries around the classes which the model would use to learn the images. The images and their respective masks should have the same filename. This was achieved through a MATLAB code, attached as *Appendix "B*".

d) **Image Splitter:** The image splitter tool was used to divide the data in training, validation, and testing sets. As per the results of initial findings, the data was split in 90%, 5%, and 5% for training, validation, and testing purposes. The splitter code is attached as *Appendix "A".*

Figure 4 - Images and Labels using MATLAB ImageLabeler.

Methodology

Based on the literature review, the following deep learning architectures were found most suitable for the semantic segmentation of crops:

UNET: In the model training and testing step, we use a state-of-the-art deep learning framework called U-Net to perform semantic segmentation. U-Net is an encoder-decoder architecture that consists of a contracting path that captures context and a symmetric expanding path that enables precise localization. U-Net also uses skip connections that concatenate feature maps from different levels of the encoder and decoder to preserve spatial information and improve segmentation accuracy.

Efficient U-Net: Efficient Unet is a semantic segmentation model that combines the Unet architecture with the EfficientNet building blocks. It aims to achieve higher performance and lower computational complexity than the original Unet. The decoder part of Efficient Unet uses a modified version of the Unet++ structure, which consists of multiple dense skip connections between the encoder and decoder feature maps. The decoder also applies channel and spatial attention to the bottleneck feature maps using concurrent spatial and channel squeeze & excitation (scSE) blocks.

The output of the decoder is a pixel-wise prediction map that indicates the class label for each pixel in the input image [7-8].

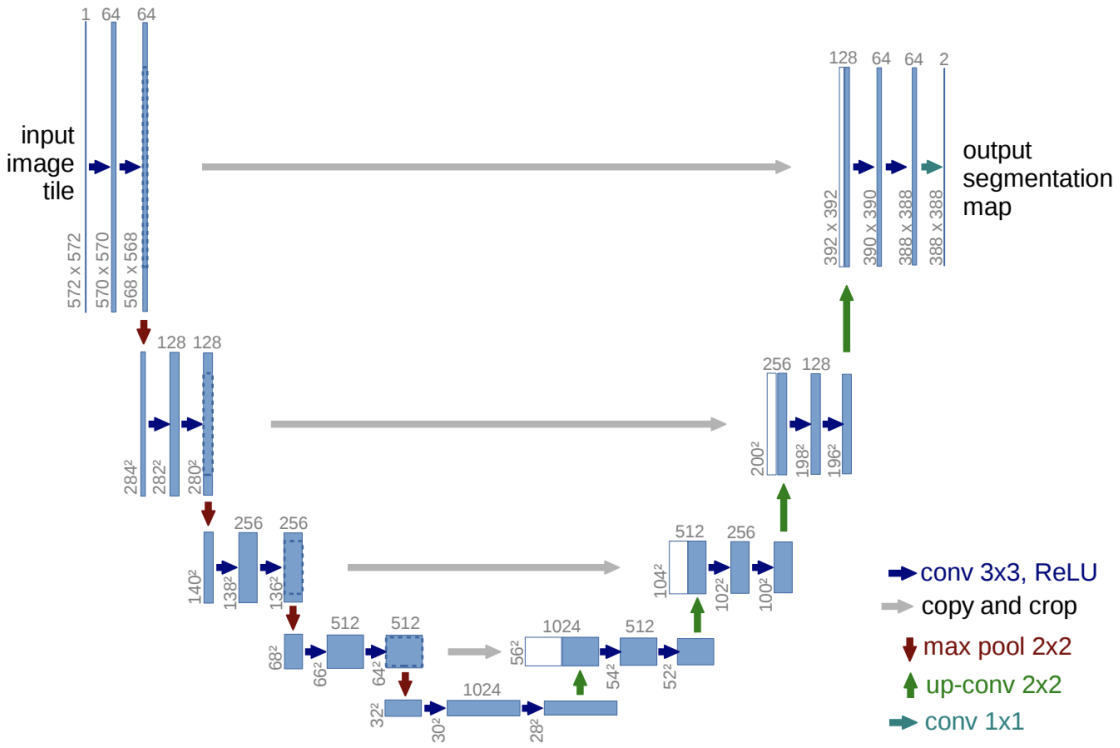


Figure 5 - U-Net architecture. Each blue box corresponds to a feature map [9]

FCN32: The convolutional part of FCN32 is based on a pre-trained VGG16 network, which has 13 convolutional layers and 3 fully connected layers. The fully connected layers are converted to convolutional layers with kernel size 7x7 and 1x1 respectively. The convolutional part produces a low-resolution feature map with 32 times down-sampling. The deconvolutional part is a single layer of transposed convolution (also called deconvolution or up-sampling) with kernel size 64x64 and stride 32. This layer samples the feature map to the same size as the input image and produces a class score for each pixel.

FCN32 is the simplest version of FCN. There are also more complex versions such as FCN16 and FCN8, which use skip connections from intermediate layers of the convolutional part to refine the segmentation results [10-11].

SegNet: The encoder network of SegNet is based on the 13 convolutional layers of the VGG16 network, which has 5 max-pooling layers that down-sample the input image by 32 times. The encoder network produces low-resolution feature maps that capture the semantic information of the images.

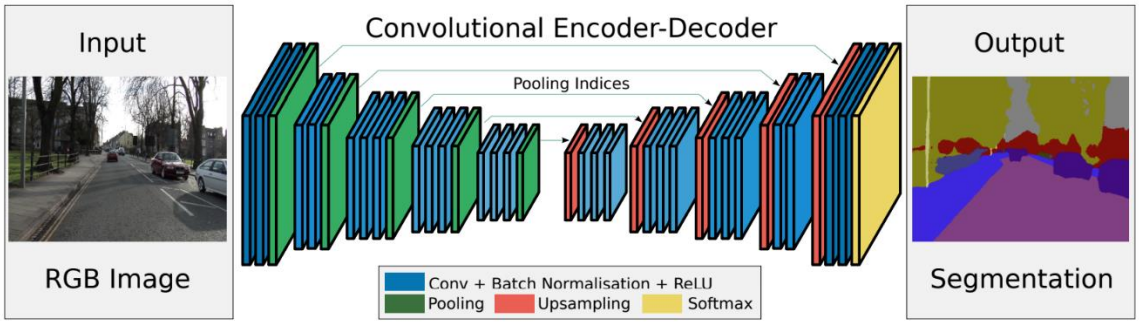


Figure 6 - Figure shows an illustration of SegNet architecture [12]

The decoder network is composed of 5 up-sampling layers that use the pooling indices computed in the corresponding encoder max-pooling layers to perform non-linear up-sampling. This preserves the boundary information of the objects and reduces the number of parameters to learn. The decoder network produces high-resolution feature maps that are then fed to a softmax layer to produce class probabilities for each pixel.

SegNet is similar to FCN32 in terms of architecture but differs in the way it performs up-sampling. SegNet uses pooling indices to up-sample, while FCN32 uses transposed convolution with a fixed filter. SegNet is more efficient in terms of memory and computational time than FCN32, and also produces sharper segmentation results [13-14].

Hyperparameters

Different hyperparameters, such *as batch_size, steps_per_epoch, and epochs* were fine-tuned to match with the models and datasets used for each training. Since the datasets for all three crops used are of different size, therefore the network parameters had to be changed for each model.

Metrics

Recall, precision, F1 score, and mean IU are four metrics are used to evaluate the performance of a classification model. A classification model is a model that predicts the class label of an input, such as if any email is spam or not.

a) **Recall**: Recall is a measure of how well the model can identify the positive class. It is calculated as the ratio of true positives (TP) to the total number of actual positives (TP + FN), where FN is the number of false negatives.

b) **Precision**: Precision is a measure of how accurate the model is when it predicts the positive class. It is calculated as the ratio of true positives (TP) to the total number of predicted positives. Precision value basically tells how many of the positives predicted by the model were positive.

c) **F1 Score**: F1 score is a metric that combines both precision and recall. It is calculated as the harmonic mean of precision and recall, which means it gives more weight to low values. F1 score uses *precision* and *recall* values to calculate the accuracy of the model. It is mathematically defined as follows:

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

F1 score is a good measure of accuracy *when dealing with imbalanced data* or when there is a high cost for false negatives or false positives.

d) **Mean IU**: Mean IU (Intersection over Union) is a metric that is used to evaluate the performance of a segmentation model. It is calculated as the average of the intersection over union (IU) values for each class in the image. The IU value for a class is calculated as the ratio of the area of overlap between the predicted and actual regions for that class to the area of union between them.

$$Mean\ IU = \frac{1}{N} * sum\ (IU)$$

for c = 1 to N, where N is the number of classes, and IU is the intersection over union value for class c. Mean IU is a better measure than pixel accuracy when dealing with imbalanced data or when there are multiple classes in an image.

Results and Discussion

Many different cases of the model were run on the dataset. First, the complete dataset of all three classes combined was run together to check the robustness of the model. The same combined dataset was then run with data augmentation to address the problem of small number of overall images. Both these models were run using Efficient UNet. Results of the combined dataset on Efficient UNet are shown below.

The same network was then used to train a model in which an equal number of images of each class (82) was used. The purpose of this was to check the performance of the model when class imbalance was resolved, however, this reduced the total number of images from 1085 to 246.

After analyzing the performance of different models on Efficient UNet, the following conclusions were made:

a) Class imbalance improved the results to some extent, but the improvements were not significant enough to disregard the additional number of labelled examples of tomato that were available.
b) Training on combined dataset did not produce good results as compared to dataset of models that were trained independently from each other.

Keeping the following findings in view, different models were then tested on UNet, SegNet, and FCN32 using datasets of each class separately using the complete dataset of each class without data augmentation.

Table 1 - A comparison of the number of learning parameters for the models used.

	FCN32	SegNet	U-Net
Total parameters	69,745,026	3,697,602	4,471,746
Trainable params	69,743,106	3,694,274	4,468,418
Non-trainable params	1,920	3,328	3,328

Table 2 - Evaluation metrics obtained using different models on Tomato dataset.

Testing Metrics for Tomato Class		
	FCN32	**SegNet**
IoU	0.6595	0.7575
Precision	0.9330	0.9425
Recall	0.6923	0.7943
F1 Score	0.8984	0.9298

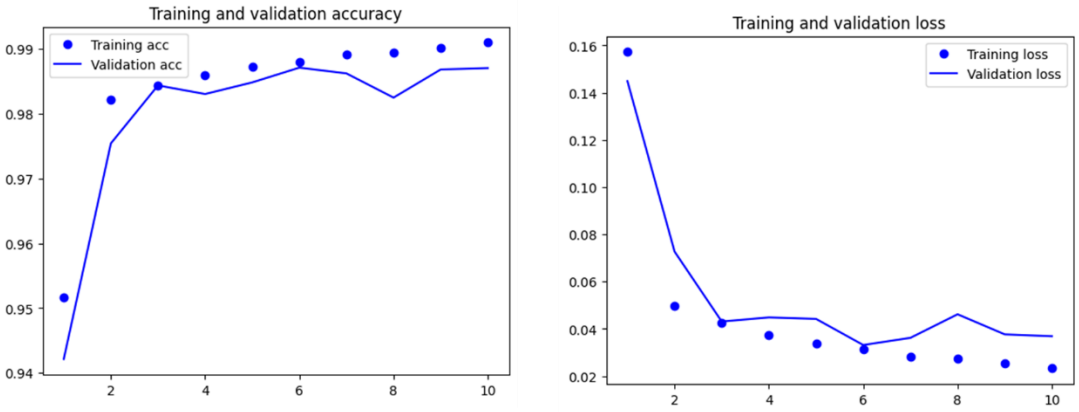
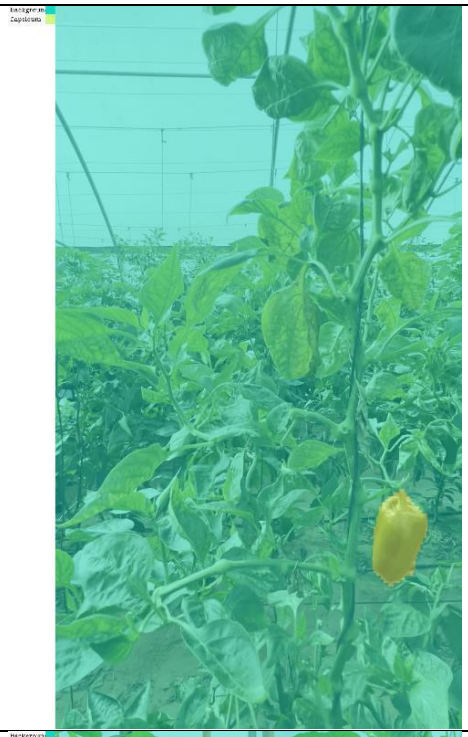


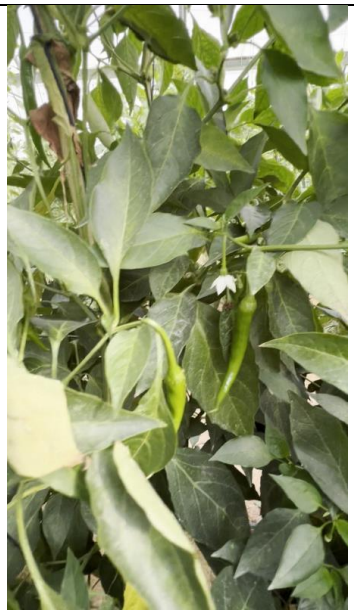
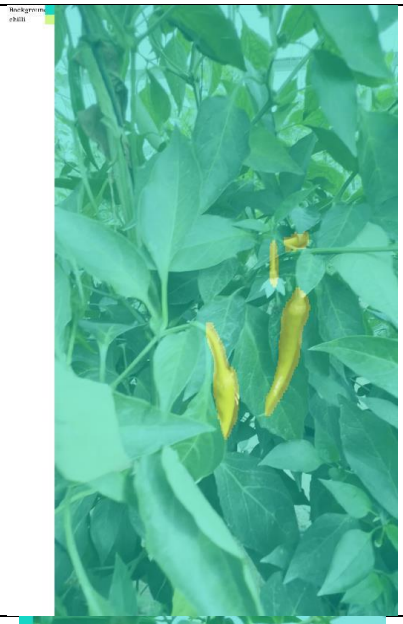
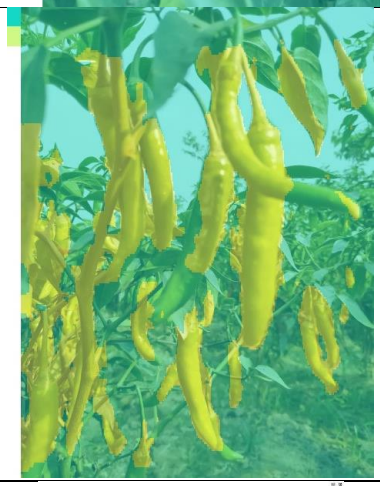
Figure 7 - Training and validation loss for the Tomato dataset using SegNet

Prediction

Overlay of predicted images for is shown below. The images for this purpose are selected from the testing dataset, as well as from random image from internet source. The results shows that the models predict very well on the testing images. The model also, in some

cases, tries to predict the intended class on random images from outside the dataset, which means that it has the capability to be enhanced for generalization.

Dataset / Model	Image	Predicted Overlay
Capsicum UNet Testing Dataset Image		
Capsicum UNet Random Images		

Chilli SegNet Testing dataset Image		
Chilli SegNet Random Image		
Tomato FCN32 Testing dataset Image		

Tomato SegNet Testing dataset Image	

Figure 8 - Summary of Outputs

From the results, it is evident that SegNet produces the optimum results. FCN32 also performs well on the Tomato dataset, however, the same model was found to be failing on the dataset of chili and capsicum. This is because FCN32 has 69 million trainable parameters, which require a large size of the dataset to train properly. It is anticipated that the chili and capsicum would also produce better results on FCN32 model if the dataset is increased for these two classes as well.

All different classes were trained separately using their dataset on different models. This increased the overall effort and time required to train various models, but it produced better results than the complete dataset.

Conclusion

Based on our results, we have come to the following conclusions:

a) Semantic segmentation of the crop is a challenge due to very high occlusions with respect of appearance as well as the type of environment.

b) Dataset of considerable size should be used when training crops for semantic segmentation.

c) Images having no instance of the desired class and images where the desired class is difficult to predict even with the human eye should be removed as it increases the difficulty in training.

d) Addressing class imbalance helps to produce better results, so it is better to use datasets where the labeled examples of each class are equal.

e) The datasets of each class should be trained independently for optimum results, as different classes are very similar to each other in appearance, and it confuses the model.

f) Data augmentation does not help to improve the results where the class imbalance is very high. It is preferrable to use Data augmentation for semantic segmentation when high computational resource is available as training of data takes a long time.

g) FCN32 was found to give the most accurate results (highest mean precision, recall, IU) for tomato dataset whereas SegNet produced the best results for Capsicum and Chili. This implies that FCN32 could give the best performance for all three different types of crops. However, the model fails for models having small datasets size.

Recommendation for future work

Based on our results, the following recommendations are made:

a) The classes should be trained independently, as they produce the best results. This is logical as there is no benefit of having a model that would predict all three classes with high accuracy, as it would not come across examples in the farm where all three classes are present simultaneously. Although it would be an achievement to do so, it might not be useful, considering the extensive effort that would be required.

b) Deep learning requires a large dataset to train, especially for the environment considered in this project. Hence as many as possible images should be used, which should also have good resolution which distinctly marks the boundaries of the crop with its surroundings.

c) The masks or labels created for the sample images should be done so very carefully, especially in regions around the boundary of the instance.

d) FCN32 should be evaluated for performance for capsicum and chili with larger dataset size.

References

[1] S. Malgonde, “Using convolutional neural networks for image segmentation — a quick intro,” Good Audience, 24-Dec-2017. [Online]. Available: https://blog.goodaudience.com/using-convolutional-neural-networks-for-image-segmentation-a-quick-intro-75bd68779225 [Accessed: 30-Apr-2023]

[2] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431-3440.

[3] P. Karthik et al., “Semantic segmentation for plant phenotyping using advanced deep learning pipelines,” Multimedia Tools and Applications 81 (2022): 4535-4547.

[4] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015, pp. 234-241.

[5] 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 39 (2017): 2481-2495.

[6] H. S. Gill et al., “Fruit recognition from images using deep learning applications,” Multimedia Tools and Applications 81 (2022): 33269-33290.

[7] Silva, João Lourenço, Miguel Nobre Menezes, Tiago Rodrigues, Beatriz Silva, Fausto J. Pinto, and Arlindo L. Oliveira. "Encoder-decoder architectures for clinically relevant coronary artery segmentation." arXiv preprint arXiv:2106.11447 (2021).

[8] He, K. (2020). EfficientNet-UNet [Source code]. https://github.com/he44/EfficientNet-UNet

[9] Figure 22 : O. Ronneberger, P. Fischer and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015.

[10] https://github.com/wkentaro/pytorch-fcn/blob/main/torchfcn/models/fcn32s.py.

[11] Chen, K. (2018) Keras-FCN [Source code]. https://github.com/kevinddchen/Keras-FCN

[12] V. Badrinarayanan, A. Kendall and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481-2495, Dec. 2017

[13] Silva, R., Santos, J., & Cardoso, J. S. (2020). Encoder-decoder architectures for clinically relevant coronary artery segmentation. In 2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS). IEEE.
[14] He, K. (2020). EfficientNet-UNet [Source code]. https://github.com/he44/EfficientNet-UNet

Appendices

Appendix “A”

Dataset Splitter Code used in MATLAB

```
path_to_images = fullfile(pwd, 'images\')
path_to_labels = fullfile(pwd, 'labels\')
path_to_output = fullfile(pwd, '\')
% Define the percentage split for each set
train_percentage = 0.70;
val_percentage =0.15;
test_percentage = 0.15;

% Get a list of all the images and labels
images = dir(fullfile(path_to_images, '*.png'));
labels = dir(fullfile(path_to_labels, '*.png'));

% Shuffle the list of images and labels
rng(123); % set the random seed for reproducibility
rand_order = randperm(length(images));
images = images(rand_order);
labels = labels(rand_order);

% Calculate the number of images for each set
num_images = length(images);
num_train = round(train_percentage * num_images);
num_val = round(val_percentage * num_images);
num_test = num_images - num_train - num_val;

% Create the folders for each set
mkdir(path_to_output, 'train_images');
mkdir(path_to_output, 'train_labels');
mkdir(path_to_output, 'val_images');
mkdir(path_to_output, 'val_labels');
mkdir(path_to_output, 'test_images');
mkdir(path_to_output, 'test_labels');

% Copy the images and labels to their respective folders
for i = 1:num_train
    % Copy images to train folder
    src = fullfile(path_to_images, images(i).name);
    dst = fullfile(path_to_output, 'train_images', images(i).name);
    copyfile(src, dst);

    % Copy labels to train folder
    src = fullfile(path_to_labels, labels(i).name);
    dst = fullfile(path_to_output, 'train_labels', labels(i).name);
    copyfile(src, dst);
end

for i = (num_train+1):(num_train+num_val)
    % Copy images to validation folder
    src = fullfile(path_to_images, images(i).name);
    dst = fullfile(path_to_output, 'val_images', images(i).name);
    copyfile(src, dst);
```

```
    % Copy labels to validation folder
    src = fullfile(path_to_labels, labels(i).name);
    dst = fullfile(path_to_output, 'val_labels', labels(i).name);
    copyfile(src, dst);
end

for i = (num_train+num_val+1):num_images
    % Copy images to test folder
    src = fullfile(path_to_images, images(i).name);
    dst = fullfile(path_to_output, 'test_images', images(i).name);
    copyfile(src, dst);

    % Copy labels to test folder
    src = fullfile(path_to_labels, labels(i).name);
    dst = fullfile(path_to_output, 'test_labels', labels(i).name);
    copyfile(src, dst);
end
```

Documentation:

-Place this code in the same folder as the images and labels.

-Make sure that folders are named as "images" and "labels".

-Set the following parameters according to your requirements.

```
train_percentage = 0.70;
val_percentage =0.15;
test_percentage = 0.15;
```

-Run the code and you will get the following folders as output: `train_images, train_labels, val_images, val_labels, test_images, test_labels.`

Appendix “B”

File renaming code to match images and labels file

```
clc

clear

load gTruth.mat

folder = 'C:\Users\100057573\Downloads\Tomato Images\Folder_4\Folder_4\PixelLabelData\';%folder with labels

dest = 'C:\Users\100057573\Downloads\Tomato Images\Folder_4\Folder_4\renamedLabels\';%destination to save renamed labels

mkdir renamedLabels

fullfile = [folder '*.png']; %change the extension as per the requirements

a = dir(fullfile);


for i = 1:length(a)

   realName = gTruth.DataSource{i}

%% IMP:CHECK THE NAME BEFORE RUNNING THE WHOLE SCRIPT so that the name is same as the image

   unrealImage = imread([char(gTruth(1).LabelData{i,1})]);

   imwrite(unrealImage,[dest realName(69:end-4) '.png']); %% I have selected only those characters that make up the name. Yours will be different as per your directories.

end
```

Documentation:

-As MATLAB names the labeled output pictures differently than their groundTruth name.

-This code can change the names of the labels to their real images name.

-This name changing helps the “keras_segmentation” library know where the is label for respective label.

Appendix “C”

Models’ summary

```
Model: "FCN32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 416, 608, 3)]     0

 zero_padding2d (ZeroPadding  (None, 418, 610, 3)      0
 2D)

 conv2d (Conv2D)             (None, 416, 608, 64)      1792

 batch_normalization (BatchN  (None, 416, 608, 64)     256
 ormalization)

 activation (Activation)     (None, 416, 608, 64)      0

 max_pooling2d (MaxPooling2D  (None, 208, 304, 64)     0
 )

 zero_padding2d_1 (ZeroPaddi  (None, 210, 306, 64)     0
 ng2D)

 conv2d_1 (Conv2D)           (None, 208, 304, 128)     73856

 batch_normalization_1 (Batc  (None, 208, 304, 128)    512
 hNormalization)

 activation_1 (Activation)   (None, 208, 304, 128)     0

 max_pooling2d_1 (MaxPooling  (None, 104, 152, 128)    0
 2D)

 zero_padding2d_2 (ZeroPaddi  (None, 106, 154, 128)    0
 ng2D)

 conv2d_2 (Conv2D)           (None, 104, 152, 256)     295168

 batch_normalization_2 (Batc  (None, 104, 152, 256)    1024
 hNormalization)

 activation_2 (Activation)   (None, 104, 152, 256)     0

 max_pooling2d_2 (MaxPooling  (None, 52, 76, 256)      0
 2D)

 zero_padding2d_3 (ZeroPaddi  (None, 54, 78, 256)      0
 ng2D)

 conv2d_3 (Conv2D)           (None, 52, 76, 256)       590080
```

```
batch_normalization_3 (Batc  (None, 52, 76, 256)      1024
hNormalization)

activation_3 (Activation)   (None, 52, 76, 256)       0

max_pooling2d_3 (MaxPooling  (None, 26, 38, 256)      0
2D)

zero_padding2d_4 (ZeroPaddi  (None, 28, 40, 256)      0
ng2D)

conv2d_4 (Conv2D)           (None, 26, 38, 256)       590080

batch_normalization_4 (Batc  (None, 26, 38, 256)      1024
hNormalization)

activation_4 (Activation)   (None, 26, 38, 256)       0

max_pooling2d_4 (MaxPooling  (None, 13, 19, 256)      0
2D)

conv2d_5 (Conv2D)           (None, 13, 19, 4096)      51384320

dropout (Dropout)           (None, 13, 19, 4096)      0

conv2d_6 (Conv2D)           (None, 13, 19, 4096)      16781312

dropout_1 (Dropout)         (None, 13, 19, 4096)      0

conv2d_7 (Conv2D)           (None, 13, 19, 2)         8194

conv2d_transpose (Conv2DTra  (None, 448, 640, 2)      16384
nspose)

reshape (Reshape)           (None, 286720, 2)         0

activation_5 (Activation)   (None, 286720, 2)         0

=================================================================
Total params: 69,745,026
Trainable params: 69,743,106
Non-trainable params: 1,920
_________________________________________________________________
```

```
Model: "SegNET"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 416, 608, 3)]     0

 zero_padding2d_5 (ZeroPaddi  (None, 418, 610, 3)      0
 ng2D)

 conv2d_8 (Conv2D)           (None, 416, 608, 64)      1792

 batch_normalization_5 (Batc  (None, 416, 608, 64)     256
 hNormalization)

 activation_6 (Activation)   (None, 416, 608, 64)      0

 max_pooling2d_5 (MaxPooling  (None, 208, 304, 64)     0
 2D)

 zero_padding2d_6 (ZeroPaddi  (None, 210, 306, 64)     0
 ng2D)

 conv2d_9 (Conv2D)           (None, 208, 304, 128)     73856

 batch_normalization_6 (Batc  (None, 208, 304, 128)    512
 hNormalization)

 activation_7 (Activation)   (None, 208, 304, 128)     0

 max_pooling2d_6 (MaxPooling  (None, 104, 152, 128)    0
 2D)

 zero_padding2d_7 (ZeroPaddi  (None, 106, 154, 128)    0
 ng2D)

 conv2d_10 (Conv2D)          (None, 104, 152, 256)     295168

 batch_normalization_7 (Batc  (None, 104, 152, 256)    1024
 hNormalization)

 activation_8 (Activation)   (None, 104, 152, 256)     0

 max_pooling2d_7 (MaxPooling  (None, 52, 76, 256)      0
 2D)

 zero_padding2d_8 (ZeroPaddi  (None, 54, 78, 256)      0
 ng2D)

 conv2d_11 (Conv2D)          (None, 52, 76, 256)       590080
```

```
batch_normalization_8 (Batc  (None, 52, 76, 256)      1024
hNormalization)

activation_9 (Activation)   (None, 52, 76, 256)       0

max_pooling2d_8 (MaxPooling  (None, 26, 38, 256)      0
2D)

zero_padding2d_10 (ZeroPadd  (None, 28, 40, 256)      0
ing2D)

conv2d_13 (Conv2D)          (None, 26, 38, 512)       1180160

batch_normalization_10 (Bat  (None, 26, 38, 512)      2048
chNormalization)

up_sampling2d (UpSampling2D  (None, 52, 76, 512)      0
)

zero_padding2d_11 (ZeroPadd  (None, 54, 78, 512)      0
ing2D)

conv2d_14 (Conv2D)          (None, 52, 76, 256)       1179904

batch_normalization_11 (Bat  (None, 52, 76, 256)      1024
chNormalization)

up_sampling2d_1 (UpSampling  (None, 104, 152, 256)    0
2D)

zero_padding2d_12 (ZeroPadd  (None, 106, 154, 256)    0
ing2D)

conv2d_15 (Conv2D)          (None, 104, 152, 128)     295040

batch_normalization_12 (Bat  (None, 104, 152, 128)    512
chNormalization)

up_sampling2d_2 (UpSampling  (None, 208, 304, 128)    0
2D)

zero_padding2d_13 (ZeroPadd  (None, 210, 306, 128)    0
ing2D)

conv2d_16 (Conv2D)          (None, 208, 304, 64)      73792

batch_normalization_13 (Bat  (None, 208, 304, 64)     256
chNormalization)

conv2d_17 (Conv2D)          (None, 208, 304, 2)       1154
 reshape_1 (Reshape)         (None, 63232, 2)          0

 activation_11 (Activation)  (None, 63232, 2)          0

=================================================================
Total params: 3,697,602
Trainable params: 3,694,274
Non-trainable params: 3,328
_________________________________________________________________
```

```
Model: "UNET"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 416, 608, 3)]     0

zero_padding2d_14 (ZeroPadding2D)  (None, 418, 610, 3)  0

conv2d_18 (Conv2D)           (None, 416, 608, 64)      1792

batch_normalization_14 (BatchNormalization)  (None, 416, 608, 64)  256

activation_12 (Activation)   (None, 416, 608, 64)      0

max_pooling2d_10 (MaxPooling2D)  (None, 208, 304, 64)  0

zero_padding2d_15 (ZeroPadding2D)  (None, 210, 306, 64)  0

conv2d_19 (Conv2D)           (None, 208, 304, 128)     73856

batch_normalization_15 (BatchNormalization)  (None, 208, 304, 128)  512

activation_13 (Activation)   (None, 208, 304, 128)     0

max_pooling2d_11 (MaxPooling2D)  (None, 104, 152, 128)  0

zero_padding2d_16 (ZeroPadding2D)  (None, 106, 154, 128)  0

conv2d_20 (Conv2D)           (None, 104, 152, 256)     295168

batch_normalization_16 (BatchNormalization)  (None, 104, 152, 256)  1024

activation_14 (Activation)   (None, 104, 152, 256)     0

max_pooling2d_12 (MaxPooling2D  (None, 52, 76, 256)    0
```

```
max_pooling2d_12 (MaxPooling2D  (None, 52, 76, 256)  0
)

zero_padding2d_17 (ZeroPadding  (None, 54, 78, 256)  0
2D)

conv2d_21 (Conv2D)              (None, 52, 76, 256)  590080

batch_normalization_17 (BatchN  (None, 52, 76, 256)  1024
ormalization)

activation_15 (Activation)      (None, 52, 76, 256)  0

max_pooling2d_13 (MaxPooling2D  (None, 26, 38, 256)  0
)

zero_padding2d_19 (ZeroPadding  (None, 28, 40, 256)  0
2D)

conv2d_23 (Conv2D)              (None, 26, 38, 512)  1180160

batch_normalization_19 (BatchN  (None, 26, 38, 512)  2048
ormalization)

up_sampling2d_3 (UpSampling2D)  (None, 52, 76, 512)  0

concatenate (Concatenate)       (None, 52, 76, 768)  0


zero_padding2d_20 (ZeroPadding  (None, 54, 78, 768)  0
2D)

conv2d_24 (Conv2D)              (None, 52, 76, 256)  1769728

batch_normalization_20 (BatchN  (None, 52, 76, 256)  1024
ormalization)

up_sampling2d_4 (UpSampling2D)  (None, 104, 152, 25  0
                                6)

concatenate_1 (Concatenate)     (None, 104, 152, 38  0
                                4)

zero_padding2d_21 (ZeroPadding  (None, 106, 154, 38  0
2D)                             4)

conv2d_25 (Conv2D)              (None, 104, 152, 12  442496
                                8)

batch_normalization_21 (BatchN  (None, 104, 152, 12  512
ormalization)                   8)
```

```
up_sampling2d_5 (UpSampling2D)  (None, 208, 304, 128)  0         ['batch_normalization_21[0][0]']

concatenate_2 (Concatenate)     (None, 208, 304, 192)  0         ['up_sampling2d_5[0][0]',
                                                                  'max_pooling2d_10[0][0]']

zero_padding2d_22 (ZeroPadding  (None, 210, 306, 192)  0         ['concatenate_2[0][0]']
2D)

conv2d_26 (Conv2D)              (None, 208, 304, 64)   110656    ['zero_padding2d_22[0][0]']

batch_normalization_22 (BatchN  (None, 208, 304, 64)   256       ['conv2d_26[0][0]']
ormalization)

conv2d_27 (Conv2D)              (None, 208, 304, 2)    1154      ['batch_normalization_22[0][0]']

reshape_2 (Reshape)             (None, 63232, 2)       0         ['conv2d_27[0][0]']

activation_17 (Activation)      (None, 63232, 2)       0         ['reshape_2[0][0]']

==================================================================================================
Total params: 4,471,746
Trainable params: 4,468,418
Non-trainable params: 3,328
__________________________________________________________________________________________________
```

Appendix “D”

```
pip install keras-segmentation

from google.colab import drive
drive.mount('/content/drive')

train_images_dir ='/content/drive/My Drive/Project/validation/train_images
'
train_labels_dir ='/content/drive/My Drive/Project/validation/train_labels
'
test_images_dir ='/content/drive/My Drive/Project/validation/test_images'
test_labels_dir ='/content/drive/My Drive/Project/validation/test_labels'
val_images_dir ='/content/drive/My Drive/Project/validation/val_images'
val_labels_dir ='/content/drive/My Drive/Project/validation/val_labels'

import os

print(len(os.listdir(train_images_dir)))
print(len(os.listdir(train_labels_dir)))
print(len(os.listdir(test_images_dir)))
print(len(os.listdir(test_labels_dir)))
print(len(os.listdir(val_images_dir)))
print(len(os.listdir(val_labels_dir)))
```

Using Pre-Existing Model

```
from keras_segmentation.models.unet import unet

model = unet(n_classes=4)
```

Training

```
history = model.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    checkpoints_path="/checkpoints/name",
    auto_resume_checkpoint=True,
    validate=True,
    verify_dataset=False,
    val_images=val_images_dir,
    #load_weights="/content/drive/My Drive/deep_learning_project/vgg_pspne
t.h5",
    val_annotations=val_labels_dir,
    epochs=10,
    do_augment=False,
```

```
)
 #
 # load_weights is a parameter
 # do_augment = True,
 # Define the path to the directory where you want to save the model
save_path = "/content/drive/My Drive/deep_learning_project/"

# Save the model to the defined path
model.save(save_path + "unet1.h5")


print(history.history.keys())
```

Parallel running

```
from keras_segmentation.models.fcn import fcn_32
from keras_segmentation.models.segnet import segnet
from keras_segmentation.models.unet import unet

tomato = fcn_32(n_classes=2)

chilli = segnet(n_classes=2)

capsicum = unet(n_classes=2)
```

Model Summaries:

```
tomato.summary()
chilli.summary()
capsicum.summary()
```

Parallel Running: (SegNet)

```
from keras_segmentation.models.segnet import segnet

tomato = segnet(n_classes=2)

chilli = segnet(n_classes=2)

capsicum = segnet(n_classes=2)
```

Parallel Running: (FCN & SegNet)

```
from keras_segmentation.models.fcn import fcn_32
from keras_segmentation.models.segnet import segnet

tomato = fcn_32(n_classes=2)

chil_cap = segnet(n_classes=3)
```

Training Using (FCN &SegNet):

```
tomato.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"tomato_fcn32.h5",
    ignore_zero_class=False,
    do_augment=False,
)
chil_cap.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"chil_cap_segnet.h5",
    ignore_zero_class=False,
    do_augment=False,
)
```

Training Using (SegNet):

```
tomato.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
```

```
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"segnettomato.h5",
    ignore_zero_class=False,
    do_augment=False,
)
chilli.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"segnetchilli.h5",
    ignore_zero_class=False,
    do_augment=False,
)
capsicum.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"segnetcapsicum.h5",
    ignore_zero_class=False,
    do_augment=False,
)
```

Training Using (FCN & SegNet & UNet):

```
tomato.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"tomato_fcn32.h5",
```

```
    ignore_zero_class=False,
    do_augment=False,
)
chilli.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"segnetchilli.h5",
    ignore_zero_class=False,
    do_augment=False,
)
capsicum.train(
    train_images=train_images_dir,
    train_annotations=train_labels_dir,
    auto_resume_checkpoint=False,
    validate=False,
    verify_dataset=False,
    batch_size = 0,
    steps_per_epoch = 0,
    epochs=0,
    load_weights=r"capsicum_unet.h5",
    ignore_zero_class=False,
    do_augment=False,
)
```

Graphs

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
```

```
plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Testing Image:

```
out = tomato.predict_segmentation(
    inp=r"C:\Users\Hammad Hasan\Deep Learning Project\test\tomato.png",
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\output.pn
g"
)
```

Printing the Test Image's Label:

```
%matplotlib inline

import matplotlib
import matplotlib.pyplot as plt
plt.imshow(out)
print(out)
```

Predicting Using (FCN & SegNet):

```
# Perform semantic segmentation on the image
import numpy as np
image = r"C:\Users\Hammad Hasan\Deep Learning Project\test\tomato.png"
out = tomato.predict_segmentation(
    inp=image,
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\out.png"
,
    overlay_img=True,
    show_legends=True,
    class_names=["None", "Tomato"]
)
out2 = chil_cap.predict_segmentation(
    inp=image,
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\out2.png"
 ,
    overlay_img=True,
    show_legends=True,
    class_names=["None", "Chili", "Capsicum"]
```

```
)
# Extract the pixels corresponding to the "Tomato" class label
tomato_pixels = np.array(list(zip(*np.where(out == 1))))
# Extract the pixels corresponding to the "Chili" class label
chili_pixels = np.array(list(zip(*np.where(out2 == 1))))
# Extract the pixels corresponding to the "Capsicum" class label
capsicum_pixels = np.array(list(zip(*np.where(out2 == 2))))
print(out2)
tomato_pixels_count = len(tomato_pixels)
chili_pixels_count = len(chili_pixels)
capsicum_pixels_count = len(capsicum_pixels)


print("Tomato pixels count:", tomato_pixels_count)
print("Chili pixels count:", chili_pixels_count)
print("Capsicum pixels count:", capsicum_pixels_count)


total_pixels = tomato_pixels_count + chili_pixels_count + capsicum_pixels_
count
tomato_probability = round((tomato_pixels_count / total_pixels)*100,2)
chili_probability = round((chili_pixels_count / total_pixels)*100,2)
capsicum_probability = round((capsicum_pixels_count / total_pixels)*100,2)


print("Tomato probability:", tomato_probability,'%')
print("Chili probability:", chili_probability,'%')
print("Capsicum probability:", capsicum_probability,'%')

# Print the pixel values
print("Tomato pixels:")
print(np.array([np.asarray(out[tomato_pixels[i][0]][tomato_pixels[i][1]])
for i in range(len(tomato_pixels))]))

print("Chili pixels:")
print(np.array([np.asarray(out2[chili_pixels[i][0]][chili_pixels[i][1]]) f
or i in range(len(chili_pixels))]))

print("Capsicum pixels:")
print(np.array([np.asarray(out3[capsicum_pixels[i][0]][capsicum_pixels[i][
1]]) for i in range(len(capsicum_pixels))]))

from IPython.display import Image
Image(r'C:\Users\Hammad Hasan\Deep Learning Project\test\out2.png')
```

Predicting Using 3 Models Running in Parallel:

```
# Perform semantic segmentation on the image
import numpy as np
image = r"C:\Users\Hammad Hasan\Deep Learning Project\test\tomato.png"
out = tomato.predict_segmentation(
    inp=image,
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\out.png"
,
    overlay_img=True,
    show_legends=True,
    class_names=["None", "Tomato"]
)
out2 = chilli.predict_segmentation(
    inp=image,
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\out2.png"
 ,
    overlay_img=True,
    show_legends=True,
    class_names=["None", "Chili"]
)
out3 = capsicum.predict_segmentation(
    inp=image,
    out_fname=r"C:\Users\Hammad Hasan\Deep Learning Project\test\out3.png"
 ,
    overlay_img=True,
    show_legends=True,
    class_names=["None", "Capsicum"]
)
# Extract the pixels corresponding to the "Tomato" class label
tomato_pixels = np.array(list(zip(*np.where(out == 1))))
# Extract the pixels corresponding to the "Chili" class label
chili_pixels = np.array(list(zip(*np.where(out2 == 1))))
# Extract the pixels corresponding to the "Capsicum" class label
capsicum_pixels = np.array(list(zip(*np.where(out3 == 1))))
tomato_pixels_count = len(tomato_pixels)
chili_pixels_count = len(chili_pixels)
capsicum_pixels_count = len(capsicum_pixels)


print("Tomato pixels count:", tomato_pixels_count)
print("Chili pixels count:", chili_pixels_count)
print("Capsicum pixels count:", capsicum_pixels_count)
```

```
total_pixels = tomato_pixels_count + chili_pixels_count + capsicum_pixels_
count
tomato_probability = round((tomato_pixels_count / total_pixels)*100,2)
chili_probability = round((chili_pixels_count / total_pixels)*100,2)
capsicum_probability = round((capsicum_pixels_count / total_pixels)*100,2)


print("Tomato probability:", tomato_probability,'%')
print("Chili probability:", chili_probability,'%')
print("Capsicum probability:", capsicum_probability,'%')

# Print the pixel values
print("Tomato pixels:")
print(np.array([np.asarray(out[tomato_pixels[i][0]][tomato_pixels[i][1]])
for i in range(len(tomato_pixels))]))

print("Chili pixels:")
print(np.array([np.asarray(out2[chili_pixels[i][0]][chili_pixels[i][1]]) f
or i in range(len(chili_pixels))]))

print("Capsicum pixels:")
print(np.array([np.asarray(out3[capsicum_pixels[i][0]][capsicum_pixels[i][
1]]) for i in range(len(capsicum_pixels))]))

from IPython.display import Image
Image(r'C:\Users\Hammad Hasan\Deep Learning Project\test\out2.png')
```

Prediction using Checkpoints

```
from keras_segmentation.predict import predict

predict(
      checkpoints_path="/checkpoints/name",
      inp="/content/drive/My Drive/deep_learning_project/capsicum.png",
      out_fname="/content/drive/My Drive/deep_learning_project/output.png"
)
```

Output training and testing parameters

```
print(model.evaluate_segmentation( inp_images_dir=train_images_dir  , anno
tations_dir=train_labels_dir ) )

print(model.evaluate_segmentation( inp_images_dir=test_images_dir  , annot
ations_dir=test_labels_dir ) )
```

Documentation (For Appendix D):

```
from keras_segmentation.models.model_utils import get_segmentation_model

model = get_segmentation_model(img_input ,  out )
```

model_utils can be chosen as one of the following: “fcn”, “segnet”, “pspnet”, “unet”.

Get_segmentation_model can be chosen from the following table under the “model_name” column, make sure to chose the corresponding segmentation model using the model_utils value.

Table 3 - Available Models

model_name	Base Model	Segmentation Model
fcn_8	Vanilla CNN	FCN8
fcn_32	Vanilla CNN	FCN8
fcn_8_vgg	VGG 16	FCN8
fcn_32_vgg	VGG 16	FCN32
fcn_8_resnet50	Resnet-50	FCN32
fcn_32_resnet50	Resnet-50	FCN32
fcn_8_mobilenet	MobileNet	FCN32
fcn_32_mobilenet	MobileNet	FCN32
pspnet	Vanilla CNN	PSPNet
pspnet_50	Vanilla CNN	PSPNet
pspnet_101	Vanilla CNN	PSPNet
vgg_pspnet	VGG 16	PSPNet
resnet50_pspnet	Resnet-50	PSPNet
unet_mini	Vanilla Mini CNN	U-Net
unet	Vanilla CNN	U-Net
vgg_unet	VGG 16	U-Net
resnet50_unet	Resnet-50	U-Net
mobilenet_unet	MobileNet	U-Net
segnet	Vanilla CNN	Segnet
vgg_segnet	VGG 16	Segnet
resnet50_segnet	Resnet-50	Segnet
mobilenet_segnet	MobileNet	Segnet

-For model_name.train(), the following input parameters are available.

```
        train_images,
        train_annotations,
        input_height=None,
        input_width=None,
        n_classes=None,
        verify_dataset=True,
        checkpoints_path=None,
        epochs=5,
        batch_size=2,
        validate=False,
        val_images=None,
        val_annotations=None,
        val_batch_size=2,
        auto_resume_checkpoint=False,
        load_weights=None,
        steps_per_epoch=512,
        val_steps_per_epoch=512,
        gen_use_multiprocessing=False,
        ignore_zero_class=False,
        optimizer_name='adadelta',
        do_augment=False,
        augmentation_name="aug_all"
```

-Using these, the hyperparameters can be tuned, and some other features like verifying dataset, data augmentation, resuming from checkpoints or loading weights can be used.

The following changes were made to the “train.py” file from “keras_segmentation” library:

```
    if not validate:
        return model.fit_generator(train_gen, steps_per_epoch,
                                   epochs=epochs, callbacks=callbacks)
    else:
        return model.fit_generator(train_gen,
                                   steps_per_epoch,
                                   validation_data=val_gen,
                                   validation_steps=val_steps_per_epoch,
                                   epochs=epochs, callbacks=callbacks,
                                   use_multiprocessing=gen_use_multiprocessing)
```

So, that the validation and training losses could be plotted.

The following changes were made to the “predict.py” file from “keras_segmentation” library:

```
    precision = tp / (tp + fp + 0.000000000001)
    recall = tp / (tp + fn + 0.000000000001)
    iou = tp / (tp + fp + fn + 0.000000000001)
    cl_wise_score = iou
    n_pixels_norm = n_pixels / np.sum(n_pixels)
    frequency_weighted_IU = np.sum(cl_wise_score*n_pixels_norm)
    mean_IU = np.mean(cl_wise_score)
    mean_precision = np.mean(precision)
    mean_recall = np.mean(recall)
    f1_score = 2 * (mean_precision * mean_recall) / (mean_precision +
mean_recall)

    return {
        "frequency_weighted_IU": frequency_weighted_IU,
        "mean_IU": mean_IU,
        "class_wise_IU": cl_wise_score,
        "precision": precision,
        "recall": recall,
        "mean_precision": mean_precision,
        "mean_recall": mean_recall,
        "f1_score": f1_score
    }
```

These give us the IoU, Precision, Recall, and F1-Score metrics.