Semantic Segmentation for Agriculture

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech

in

Instrumentation and Control Engineering

By

Kamalesh Palanisamy (110117039)

Rupp Santos (110117073)

Solayappan Ganesaan (110117084)

Sanjay Kumar (110117087)

INSTRUMENTATION AND CONTROL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI – 620015

MAY 2021

BONAFIDE CERTIFICATE

This is to certify that the project titled **Semantic Segmentation for Agriculture** is a bonafide record of the work done by

Kamalesh Palanisamy (110117039)

Rupp Santos (110117073)

Solayappan Ganesaan (110117084)

Sanjay Kumar (110117087)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Instrumentation and Control Engineering** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during the year 2020-21.

Dr.Umapathy	**Dr. Uma**
Guide	Head of the Department

Project Viva-voce held on ________________________

Internal Examiner	**External Examiner**

ABSTRACT

The ability to automatically monitor agricultural fields is an important capability towards creating more sustainable agricultural practices. Precise, high-resolution monitoring is a key prerequisite for targeted intervention and the selective application of agro-chemicals.

Weeds are one of the most significant factors that can reduce crop yield. Computer vision combined with image processing is an effective method for weed detection, and site-specific weed management has become an effective tool for weed control. The use of an encoder-decoder deep learning network for pixel-wise semantic segmentation of crop and weed was explored in this study. Different input representations including different color space transformations and color indices were compared to optimize the input of the network.

Aim

- To detect weed to crop ratio from the satellite images.
- Identify Regions of water stagnation from satellite images.

Keywords : CNN - Semantic Segmentation - Image Segmentation

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the following people for guiding us through this course and without whom this project and the results achieved from it would not have reached completion.

Dr.Umapathy, Assistant Professor, Department of Instrumentation and Control Engineering, for helping us and guiding us in the course of this project. Without his guidance, we would not have been able to successfully complete this project. His patience and genial attitude is and always will be a source of inspiration to us.

Dr. Uma, the Head of the Department, Department of Instrumentation and Control Engineering, for allowing us to avail the facilities at the department.

We are also thankful to the faculty and staff members of the Department of Instrumentation and Control Engineering, our individual parents and our friends for their constant support and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Agriculture is facing tremendous challenges from weeds, which appear everywhere randomly within the field, and compete with crops for water, nutrients, and sunlight, leading to a detrimental impact on crop yields and quality if uncontrolled properly. Numerous studies have demonstrated a powerful correlation between crop yield loss and weed competition. the assembly loss because of weeds may be up to 34%. to regulate weeds, different operations are adopted, among which chemical weeding has been the foremost widely used one since 1940s. However, conventional chemical weeding sprays herbicides uniformly to the full field, leading to the overuse of herbicides and further resulting in catastrophic environmental pollution problems.

1.1 Introduction to image segmentation

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of a picture into something that is more meaningful and easier to investigate. Image segmentation is usually accustomed locate objects and bounds (lines, curves, etc.) in images. More precisely, image segmentation is that the process of assigning a label to each pixel in a picture such pixels with the identical label share certain characteristics.

Now these characteristics can often lead to different types of image segmentation, which we can divide into the following: In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of a picture into something that is more meaningful and easier to investigate. Image segmentation is usually accustomed locate objects and bounds (lines, curves, etc.) in images. More precisely, image segmentation is that the process of assigning a label to each pixel in a picture such pixels with the identical label share certain characteristics.

Now these characteristics can often lead to different types of image segmentation, which we can divide into the following:

- Semantic Segmentation
- Instance Segmentation

1.1.1 Introduction to semantic and instance segmentation

Semantic segmentation is the task of classifying each and very pixel in an image into a class. The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as dense prediction.

One important thing to note is that we're not separating instances of the same class; we only care about the category of each pixel. In other words, if you have two objects of the same category in your input image, the segmentation map does not inherently distinguish these as separate objects. There exists a different class of models, known as instance segmentation models, which do distinguish between separate objects of the same class.

Semantic segmentation is different from instance segmentation which is that different objects of the same class will have different labels as in person1, person2 and hence different colours. The picture below very crisply illustrates the difference between instance and semantic segmentation.

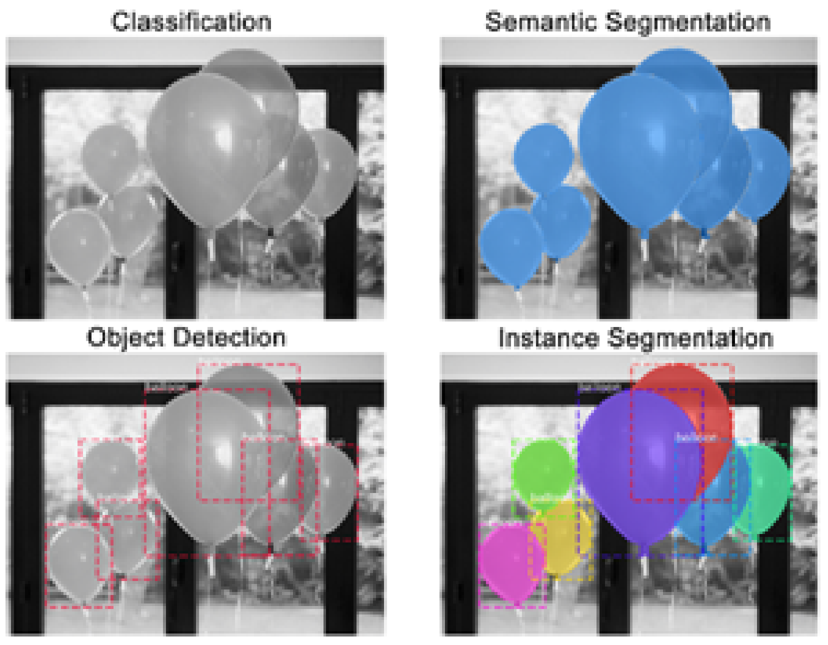


Figure 1.1: Different Image Segmentation Methods

1.2 Introduction to deep learning

Neural Networks (NN) have revolutionized the modern day-to-day life. Their significant impact is present even in our most basic actions, such as ordering products on-line via Amazon's Alexa or passing the time with on-line video games against computer agents. The NN effect is evident in many more occasions, for example, in medical imaging NNs are utilized for lesion detection and segmentation [1, 2], and tasks such as text-to-speech [3, 4] and text-to-image [5] have remarkable improvements thanks to this technology. In addition, the advancements they have caused in fields such as natural language processing (NLP) [6, 7, 8], optics [9, 10], image processing [11, 12] and computer vision (CV) [13, 14] are astonishing, creating a leap forward in technology such as autonomous driving [15, 16], face recognition [17, 18, 19], anomaly detection [20], text understanding [21] and art [22, 23], to name a few. Its influence is powerful and is continuing to grow.

1.2.1 Network Architectures

A deep learning architecture can be described as follows. Let $f_1, \ldots, f_L$ be given *univariate* activation functions for each of the L layers. Activation functions are nonlinear transformations of weighted data. A semi-affine activation rule is then defined by

$$f_l^{W,b} f_l \left(\sum_{j=1}^{N_l} W_{lj} X_j + b_l \right) = f_l(W_l X_l + b_l) \,,$$

which implicitly needs the specification of the number of hidden units N_l. Our deep predictor, given the number of layers L, then becomes the composite map

$$\hat{Y}(X) F(X) = \left(f_l^{W_1, b_1} \circ \ldots \circ f_L^{W_L, b_L} \right) (X) \,. \tag{1.1}$$

The fact that DL forms a universal 'basis' which we recognise in this formulation dates to Poincare and Hilbert is central. From a practical perspective, given a large enough data set of "test cases", we can empirically learn an optimal predictor.

Similar to a classic basis decomposition, the deep approach uses univariate activation functions to decompose a high dimensional X.

Let $Z^{(l)}$ denote the lth layer, and so $X = Z^{(0)}$. The final output Y can be numeric or categorical. The explicit structure of a deep prediction rule is then

$$\begin{aligned}
\hat{Y}(X) &= W^{(L)} Z^{(L)} + b^{(L)} \\
Z^{(1)} &= f^{(1)} \left(W^{(0)} X + b^{(0)} \right) \\
Z^{(2)} &= f^{(2)} \left(W^{(1)} Z^{(1)} + b^{(1)} \right) \\
\ldots & \\
Z^{(L)} &= f^{(L)} \left(W^{(L-1)} Z^{(L-1)} + b^{(L-1)} \right) .
\end{aligned}$$

Here $W^{(l)}$ is a weight matrix and $b^{(l)}$ are threshold or activation levels. Designing a good predictor depends crucially on the choice of univariate activation functions $f^{(l)}$. The $Z^{(l)}$ are hidden features which the algorithm will extract.

Put differently, the deep approach employs hierarchical predictors comprising of a series of L nonlinear transformations applied to X. Each of the L transformations is referred to as a *layer*, where the original input is X, the output of the first transformation is the first layer, and so on, with the output $\hat{Y}$ as the first layer. The layers 1 to L are called *hidden layers*. The number of layers L represents the *depth* of our routine.

Figure 1.2 illustrates a number of commonly used structures; for example, feed-forward architectures, auto-encoders, convolutional, and neural Turing machines. Once you have learned the dimensionality of the weight matrices which are non-zero, there's an implied network structure.

Stacked GLM. From a statistical viewpoint, deep learning models can be viewed as stacked Generalized Linear Models [**polson2017**]. The expectation over dependent variable in GLM is computed using affine transformation (linear regression model) followed by a non-linear univariate function (inverse of the link function). GLM is given by

$$E(y \mid x) = g^{-1}(w^T x).$$

Choice of link function is defined by the target distribution $p(y \mid x)$. For example when p is binomial, we choose g^{-1} to be sigmoid $1/(1 + \exp\left(-w^T x\right))$.

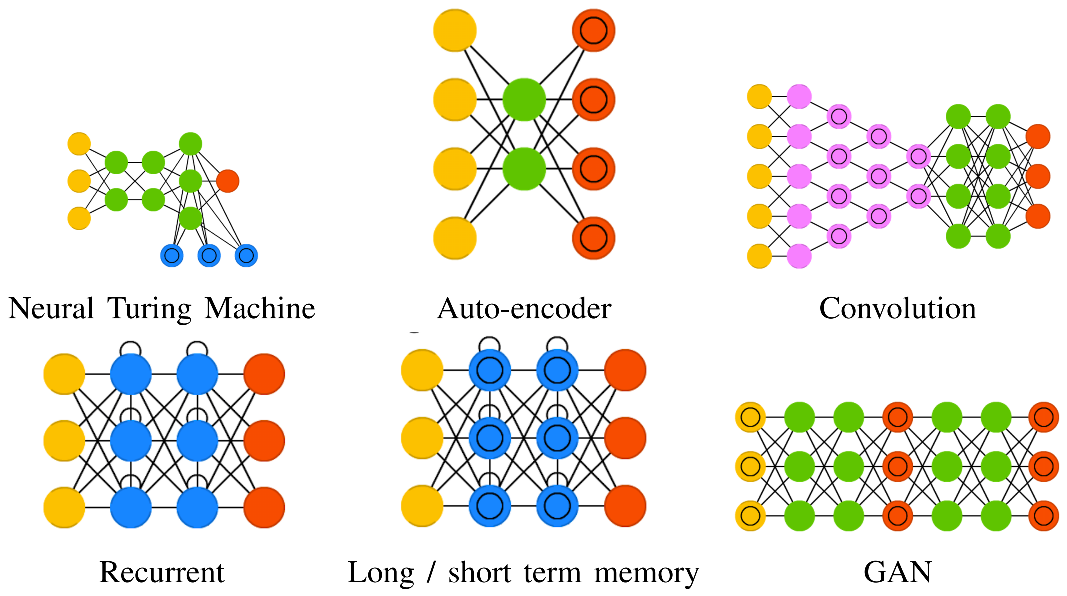


Figure 1.2: Commonly used deep learning architectures. Each circle is a neuron which calculates a weighted sum of an input vector plus bias and applies a non-linear function to produce an output. Yellow and red colored neurons are input-output cells correspondingly. Pink colored neurons apply weights inputs using a kernel matrix. Green neurons are hidden ones. Blue neurons are recurrent ones and they append its values from previous pass to the input vector. Blue neuron with circle inside a neuron corresponds to a memory cell. Source: `http://www.asimovinstitute.org/neural-network-zoo`.

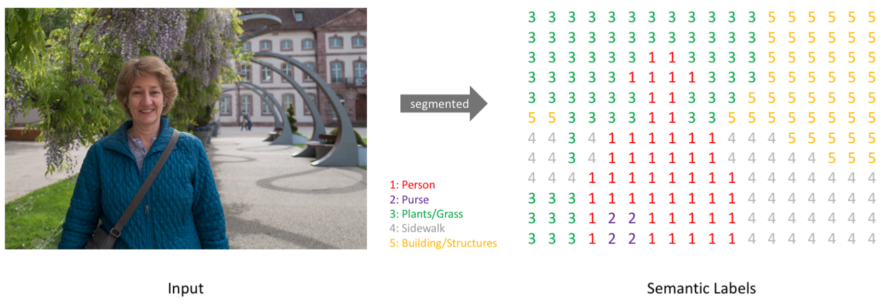
Recently deep architectures (indicating non-zero weights) include convolutional neural networks (CNN), recurrent NN (RNN), long short-term memory (LSTM), and neural Turing machines (NTM). pascanu and montufar provided results on the advantage of representing some functions compactly with deep layers. poggio extended theoretical results on when deep learning can be exponentially better than shallow learning. Bryant implemented an algorithm to estimate the non-smooth inner link function. In practice, deep layers allow for smooth activation functions to provide "learned" hyper-planes which find the underlying complex interactions and regions without having to see an exponentially large number of training samples.

Commonly used activation functions are sigmoidal ($\cosh$ or $\tanh$), heaviside gate functions $I(x > 0)$, or rectified linear units (ReLU) $\max\{\cdot, 0\}$. ReLU's especially have been found (schmidt.et.al) to lend themselves well to rapid dimension reduction. A deep learning predictor is a data reduction scheme that avoids the curse of dimensionality through the use of univariate activation functions. One particular feature is that the weight matrices $W_l \in \mathrm{Re}^{N_l \times N_{l-1}}$ are matrix valued. This gives the predictor great flexibility to uncover nonlinear features of the data – particularly so in finance data as the estimated hidden features $Z^{(l)}$ can be given the interpretation of portfolios of payouts. The choice of the dimension N_l is key, however, since if a hidden unit (aka columns of W_l) is dropped at layer l it kills all terms above it in the layered hierarchy.

CHAPTER 2

Semantic Segmentation

Pixel-wise image segmentation is a well-studied problem in computer vision. The task of semantic image segmentation is to classify each pixel in the image. Deep learning and convolutional neural networks (CNN) have been extremely ubiquitous in the field of computer vision. CNNs are popular for several computer vision tasks such as Image Classification, Object Detection, Image Generation, etc. Like for all other computer vision tasks, deep learning has surpassed other approaches for image segmentation Our goal is to take either a RGB color image (height x width x 3) or a grayscale image (height x width x 1) and output a segmentation map where each pixel contains a class label represented as an integer (height x width x 1).



3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Figure 2.1: Low-resolution prediction map

Similar to how we treat standard categorical values, we'll create our target by one-hot encoding the class labels - essentially creating an output channel for each of the possible classes. A prediction can be collapsed into a segmentation map (as shown in the first image) by taking the (argmax) of each depth-wise pixel vector. We can easily inspect a target by overlaying it onto the observation. When we overlay a single channel of our target (or prediction), we refer to this as a mask which illuminates the regions of an image where a specific class is present.

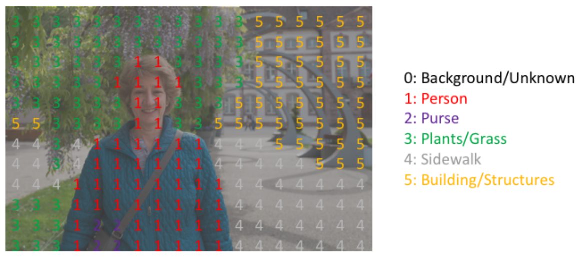


Figure 2.2: Masked Image

2.1 Traditional methods for semantic segmentation

Before DNN is proposed, features and classification methods refer to the most important topics. In the computer vision and image processing area, feature is a piece of information which is relevant for solving the computational tasks. In general, this is the same sense as feature in machine learning and pattern recognition. Variety of features are used for semantic segmentation, such as Pixel color, Histogram of oriented gradients (HOG) (Dalal and Triggs 2005; Bourdev et al. 2010), Scale-invariant feature transform (SIFT) (Lowe 2004), Local Binary Pattern (LBP) (He and Wang 1990), SURF (Bay et al. 2008), Harris Corners (Derpanis 2004), Shi-Tomasi (Shi et al. 1994), Sub-pixel Corner (Medioni and Yasumoto 1987), SUSAN(Smith and Brady 1997), Features from Accelerated Segment Test (FAST) (Rosten and Drummond 2005) and Textons (Zhu et al. 2005), just to name a few.

Approaches in image semantic segmentation include unsupervised and supervised ones. To be specific, the simple one is thresholding methods which are widely used in gray images. Gray images are very common in medical area where the collection equipment is usually X-ray CT scanner or MRI (Magnetic Resonance Imaging) equipment (Zheng et al. 2010; Hu et al. 2001; Xu et al. 2010). Overall, thresholding methods are quite effective in this area.

K-means clustering refers to an unsupervised method for clustering. The k-means algorithm requires the number of clusters to be given beforehand. Initially, k centroids are randomly placed in the feature space. Furthermore, it assigns each data point to the nearest centroid, successively moves the centroid to the center of the cluster, and continues the process until the stopping criterion is reached (Hartigan and Hartigan 1975). The segmentation problem can be treated as an energy model. It derives from compressionbased method which is implemented in Mobahi et al. (2010). Intuitively, edge is important information for segmentation. There are also many edgebased detection researches (Kimmel and Bruckstein 2003; Osher and Paragios 2003; Barghout 2014; Pedrycz et al. 2008; Barghout and Lee 2003; Lindeberg and Li 1997). Besides, edge-based approaches and region-growing methods (Nock and Nielsen 2004) are also other branches. Support vector machine (SVMs): SVMs are well-studied binary classifiers which preform well on many tasks. The training data is represented as (xi, yi) where xi is the feature vector and yi 1, 1 the binary label for training example i 1,..., m. Where w is a weight vector and b is the bias factor. Solving SVM is an optimization problem described as Eq. 5.

Slack variables can solve linearly inseparable problems. Besides, kernel method is adopted to deal with inseparable tasks through mapping current dimensional features to higher dimension. Markov Random Network (MRF) is a set of random variables having a Markov property described by an undirected graph. Also, it is an undirected graphical model. Let x be the input, and y be the output. MRF learns the distribution P(y, x). In contrast to MRF, A CRF (Russell et al. 2009) is essentially a structured extension of logistic regression, and it models the conditional probabilities P(Y —X). These two models and their variations are widely used and have reached the best performance in segmentation (http://host.robots.ox.ac.uk/pascal/VOC/voc2010/results/index.html; He et al. 2004; Shotton et al. 2006).

2.2 Deep Learning based Semantic Segmentation

2.2.1 Fully Convolution Neural Network(FCN)

FCN is a popular algorithm for doing semantic segmentation. This model uses various blocks of convolution and max pool layers to first decompress an image to 1/32th of its original size. It then makes a class prediction at this level of granularity. Finally it uses up sampling and deconvolution layers to resize the image to its original dimensions. These models typically don't have any fully connected layers. The goal of down sampling steps is to capture contextual information while the goal of up sampling is to recover spatial information. Also there are no limitations on image size. The final image is the same size as the original image. To fully recover the fine grained spatial information lost in down sampling, skip connections are used. A skip connection is a connection that bypasses at least one layer. Here it is used to pass information from the down sampling step to the up sampling step. Merging features from various resolution levels helps combining context information with spatial information.

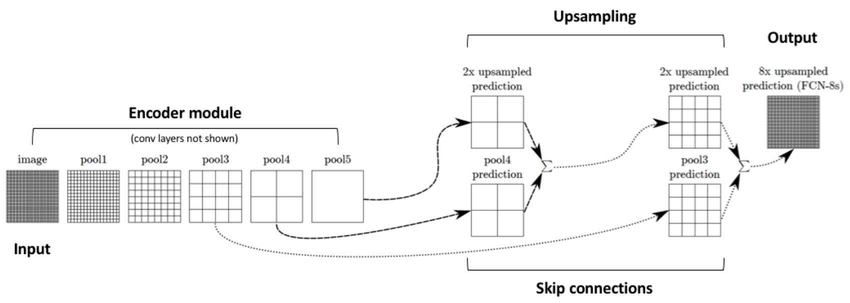


Figure 2.3: Skip Connection

These skip connections from earlier layers in the network (prior to a downsampling operation) should provide the necessary detail in order to reconstruct accurate shapes for segmentation boundaries. Indeed, we can recover more fine-grain detail with the addition of these skip connections.

2.2.2 U-Net

The U-Net architecture is built upon the Fully Convolutional Network (FCN) and modified in a way that it yields better segmentation in medical imaging.

Compared to FCN-8, the two main differences are:

- U-net is symmetric
- the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum.

These skip connections intend to provide local information to the global information while upsampling. Because of its symmetry, the network has a large number of feature maps in the upsampling path, which allows to transfer information. U-Net architecture is separated in 3 parts:

- The contracting/downsampling path
- Bottleneck.
- The expanding/upsampling path

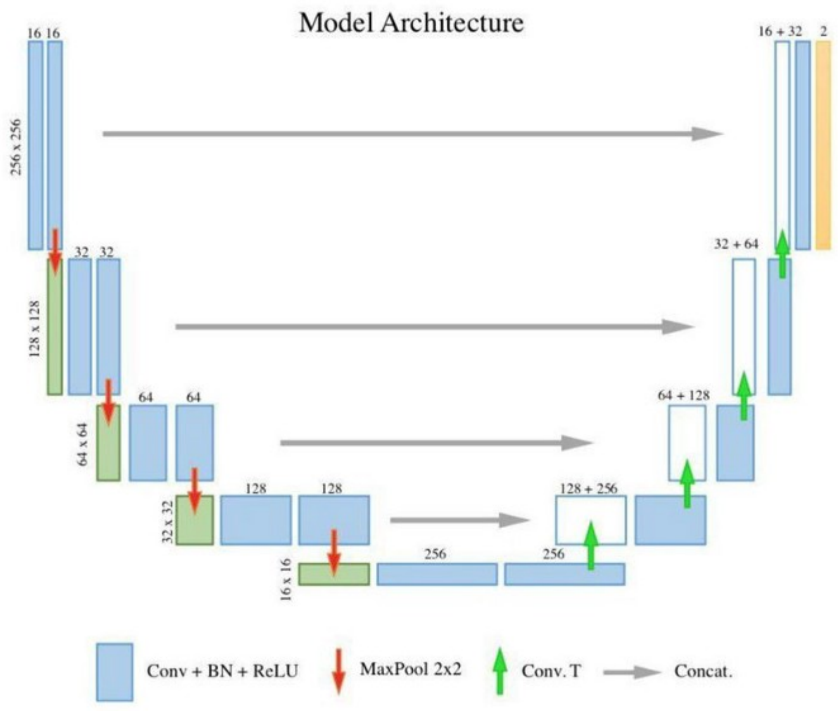


Figure 2.4: U-Net Architecture

2.2.3 Mask RCNN

Faster RCNN is a very good algorithm that is used for object detection. Faster R-CNN consists of two stages. The first stage, called a Region Proposal Network (RPN), proposes candidate object bounding boxes. The second stage, which is in essence Fast R-CNN, extracts features using RoIPool from each candidate box and performs classification and bounding-box regression. The features used by both stages can be shared for faster inference.

Mask R-CNN is conceptually simple: Faster R-CNN has two outputs for each candidate object, a class label and a bounding-box offset; to this we add a third branch that outputs the object mask — which is a binary mask that indicates the pixels where the object is in the bounding box. But the additional mask output is distinct from the class and box outputs, requiring extraction of much finer spatial layout of an object. To do this Mask RCNN uses the Fully Convolution Network (FCN).

So in short we can say that Mask RCNN combines the two networks — Faster RCNN and FCN in one mega architecture. The loss function for the model is the total loss in doing classification, generating bounding box and generating the mask.

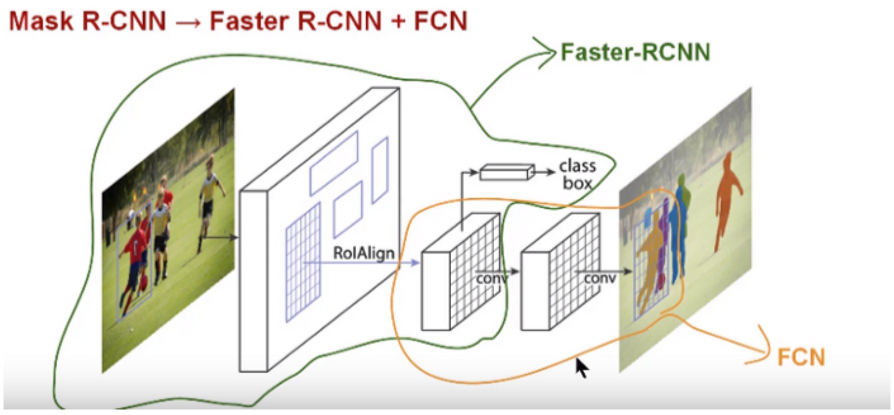


Figure 2.5: Mask RCNN

2.3 Performance Evaluation

There are two main criteria in evaluating the performance of semantics segmentation: accuracy, or in other words, the success of an algorithm; and computation complexity in terms of speed and memory requirements. In this section, we analyse these two criteria separately.

2.3.1 Accuracy

Measuring the performance of segmentation can be complicated, mainly because there are two distinct values to measure. The first is classification, which is simply determining the pixel-wise class labels; and the second is localisation, or finding the correct set of pixels that enclose the object. Different metrics can be found in the literature to measure one or both of these values. The following is a brief explanation of the principal measures most commonly used in evaluating semantic segmentation performance.

- *ROC-AUC*: ROC stands for the Receiver-Operator Characteristic curve, which summarises the trade-off between true positive rate and false-positive rate for a predictive model using different probability thresholds; whereas AUC stands for the area under this curve, which is 1 at maximum. This tool is useful in interpreting binary classification problems, and is appropriate when observations are balanced between classes. However, since most semantic segmentation image sets [24, 25, 26, 27, 28, 29, 30] are not balanced between the classes, this metric is no longer used by the most popular challenges.

- *Pixel Accuracy*: Also known as *global accuracy* [31], pixel accuracy (PA) is a very simple metric which calculates the ratio between the amount of properly classified pixels and their total number. Mean pixel accuracy (mPA), is a version of this metric which computes the ratio of correct pixels on a per-class basis. mPA is also referred to as *class average accuracy* [31].

$$PA = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k} t_j}, \qquad mPA = \frac{1}{k}\sum_{j=1}^{k}\frac{n_{jj}}{t_j} \tag{2.1}$$

  where n_{jj} is the total number of pixels both classified and labelled as class *j*. In other words, n_{jj} corresponds to the total number of *True Positives* for class *j*. t_j is the total number of pixels labelled as class *j*.

- *Intersection over Union* (IoU): Also known as the Jaccard Index, IoU is a statistic used for comparing the similarity and diversity of sample sets. In semantics segmentation, it is the ratio of the intersection of the pixel-wise classification results with the ground truth, to their union.

$$IoU = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k}(n_{ij} + n_{ji} + n_{jj})}, \qquad i \neq j \tag{2.2}$$

  where, n_{ij} is the number of pixels which are labelled as class *i*, but classified as class *j*. In other words, they are *False Positives* (false alarms) for class *j*. Similarly, n_{ji}, the total number of pixels labelled as class *j*, but classified as class *i* are the *False Negatives* (misses) for class *j*.

  Two extended versions of IoU are also widely in use:

○ *Mean Intersection over Union* (mIoU): mIoU is the class-averaged IoU, as in (2.3).

$$mIoU = \frac{1}{k}\sum_{j=1}^{k} \frac{n_{jj}}{n_{ij} + n_{ji} + n_{jj}}, \qquad i \neq j \tag{2.3}$$

○ *Frequency-weighted IoU* (FwIoU): This is an improved version of MIoU that weighs each class importance depending on appearance frequency by using t_j (the total number of pixels labelled as class *j*, as also defined in (2.1)). The formula of FwIoU is given in (2.4):

$$FwIoU = \frac{1}{\sum_{j=1}^{k} t_j}\sum_{j=1}^{k} t_j \frac{n_{jj}}{n_{ij} + n_{ji} + n_{jj}}, \qquad i \neq j \tag{2.4}$$

IoU and its extensions, compute the ratio of true positives (hits) to the sum of false positives (false alarms), false negatives (misses) and true positives (hits). Thereby, the IoU measure is more informative when compared to pixel accuracy simply because it takes false alarms into consideration, whereas PA does not. However, since false alarms and misses are summed up in the denominator, the significance between them is not measured by this metric, which is considered its primary drawback. In addition, IoU only measures the number of pixels correctly labelled without considering how accurate the segmentation boundaries are.

- *Precision-Recall Curve (PRC)-based metrics*: Precision (ratio of hits over a summation of hits and false alarms) and recall (ratio of hits over a summation of hits and misses) are the two axes of the PRC used to depict the trade-off between precision and recall, under a varying threshold for the task of binary classification. PRC is very similar to ROC. However, PRC is more powerful in discriminating the effects between the false positives (alarms) and false negatives (misses). That is predominantly why PRC-based metrics are commonly used for evaluating the performance of semantic segmentation. The formula for Precision (also called Specificity) and Recall (also called Sensitivity) for a given class *j*, are provided in (2.5):

$$Prec. = \frac{n_{jj}}{n_{ij} + n_{jj}}, \quad Recall = \frac{n_{jj}}{n_{ji} + n_{jj}}, i \neq j \tag{2.5}$$

There are three main PRC-based metrics:

○ F_{score}: Also known as the '*dice coefficient*', this measure is the harmonic mean of the precision and recall for a given threshold. It is a normalised measure of similarity, and ranges between 0 and 1 (Please see (2.6)).

$$F_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.6}$$

◦ *PRC-AuC*: This is similar to the ROC-AUC metric. It is simply the area under the PRC. This metric refers to information about the precision-recall trade-off for different thresholds, but not the *shape* of the PR curve.

◦ *Average Precision* (AP): This metric is a single value which summarises both the shape and the AUC of PRC. In order to calculate AP, using the PRC, for uniformly sampled recall values (e.g., 0.0, 0.1, 0.2, ..., 1.0), precision values are recorded. The average of these precision values is referred to as the average precision. This is the most commonly used single value metric for semantic segmentation. Similarly, mean average precision (mAP) is the mean of the AP values, calculated on a per-class basis.

IoU and its variants, along with AP, are the most commonly used accuracy evaluation metrics in the most popular semantic segmentation challenges [24, 25, 26, 27, 28, 29, 30].

2.4 Object Detection-based Methods

There has been a recent growing trend in computer vision, which aims at specifically resolving the problem of object detection, that is, establishing a bounding box around all objects within an image. Given that the image may or may not contain any number of objects, the architectures utilised to tackle such a problem differ to the existing fully-connected/convolutional classification or segmentation models.

The pioneering study that represents this idea is the renowned 'Regions with CNN features' (RCNN) network [32]. Standard CNNs with fully convolutional and fully connected layers lack the ability to provide varying length output, which is a major flaw for an object detection algorithm that aims to detect an unknown number of objects within an image. The simplest way to resolve this problem is to take different regions of interest from the image, and then to employ a CNN in order to detect objects within each region separately. This region selection architecture is called the 'Region Proposal Network' (RPN) and is the fundamental structure used to construct the RCNN network (see Figure **??**.a). Improved versions of RCNN, namely 'Fast-RCNN' [32] and 'Faster-RCNN' [33] were subsequently also proposed by the same research group. Because these networks allow for the separate detection of all objects within the image, the idea was easily implemented for instance segmentation, as the 'Mask-RCNN' [34].

The basic structure of RCNNs included the RPN, which is the combination of CNN layers and a fully connected structure in order to decide the object categories and bounding box positions. As discussed within the previous sections of this paper, due to their cumbersome structure, fully connected layers were largely abandoned with FCNs. RCNNs shared a similar fate when the 'You-Only-Look-Once' (YOLO) by [35] and 'Single Shot Detector' (SSD) by [36] were proposed. YOLO utilises a single convolutional network that predicts the bounding boxes and the class probabilities for these boxes. It consists of no fully connected layers, and consequently provides real-time performance. SSD proposed a similar idea, in which bounding boxes were predicted after multiple convolutional layers. Since each convolutional layer operates at a different scale, the architecture is able to detect objects of various scales. Whilst slower than YOLO, it is still considered to be faster then RCNNs. This new breed of object detection techniques was immediately applied to semantic segmentation. Similar to MaskRCNN, 'Mask-YOLO' [37] and 'YOLACT' [38] architectures were implementations of these object detectors to the problem of instance segmentation.

Locating objects within an image prior to segmenting them at the pixel-level is both intuitive and natural, due to the fact that it is effectively how the human brain supposedly accomplishes this task [39]. In addition to these "two-stage (detection+segmentation) methods, there are some recent studies that aim at utilizing the segmentation task to be incorporated into one-stage bounding-box detectors and result in a simple yet efficient instance segmentation framework [40, 41, 42, 43]. In conclusion, employing object detection-based methods for semantic segmentation is an area significantly prone to further development in the near future.

CHAPTER 3
Our contributions

We decided to draw in from the recent object-detection algorithms to improve the quality of semantic segmentation algorithms. We wanted to have a strong baseline to compare our model against and chose to use the MSCG-NET[44] based on graph neural networks. Our contributions include using a attention based model for a novel task of semantic segmentation. Further we provide ablation studies on how our results can be improved by tweaking several hyper-parameters in the model. To conclude our work, we try to use various explainability methods to understand how our model performs on each of these tasks.

3.1 Dataset

The dataset[1] used in this thesis is a subset of the Agriculture-Vision dataset. The dataset contains 21,061 aerial farmland images captured throughout 2019 across the US. Each image consists of four 512x512 color channels, which are RGB and Near Infra-red (NIR). Each image also has a boundary map and a mask. The boundary map indicates the region of the farmland, and the mask indicates valid pixels in the image. Regions outside of either the boundary map or the mask are not evaluated.

This dataset contains six types of annotations: Cloud shadow, Double plant, Planter skip, Standing Water, Waterway and Weed cluster. These types of field anomalies have great impacts on the potential yield of farmlands, therefore it is extremely important to accurately locate them. In the Agriculture-Vision dataset, these six patterns are stored separately as binary masks due to potential overlaps between patterns.

3.2 MSCG-Net

Currently, the end-to-end semantic segmentation models are mostly inspired by the idea of fully convolutional networks that generally consist of an encoder-decoder architecture. To achieve higher performance, CNN-based endto- end methods normally rely on deep and wide multi-scale CNN architectures to create a large receptive field in order to obtain strong local patterns, but also capture long range dependencies between objects of the scene. However, this approach for modeling

[1]https://github.com/SHI-Labs/Agriculture-Vision

global context relationships is highly inefficient and typically requires a large number of trainable parameters, considerable computational resources, and large labeled training datasets.

Graph Convolution Networks

Graph Convolutional Networks (GCNs) are neural networks designed to operate on and extract information from graphs and were originally proposed for the task of semi-supervised node classification. G= (A,X)denotes an undirected graph with nn-odes, where $A \in R^{nXd}$ is the adjacency matrix and $X \in R^{nXd}$ is the feature matrix. At each layer, the GCN aggregates information in one-hop neighborhood

Self Constructing Graphs

The Self-Constructing Graph (SCG) module allows the construction of undirected graphs, capturing relations across the image, directly from feature maps, instead of relying on prior knowledge graphs. It has achieved promising performance on semantic segmentation tasks in remote sensing and is efficient with respect to the number of trainable parameters,outperforming much larger models. It is inspired by variational graph auto-encoders The SCG module learns a mean matrix $\mu \in R^{nXc}$ and a standard deviation matrix $\sigma \in R^{nXc}$ of a Gaussian using two single-layer convolutional networks

MSCG-Net We propose a so-called Multi-view SCG-Net (MSCG-Net) to extend the vanilla SCG and GCN modules by considering multiple rotated views in order to obtain and fuse more robust global contextual information in airborne images in this work.We first augment the features (X) learned by a backbone CNN network to multiple views (X90 and X180) by rotating the features. The employed SCG-GCN module then outputs multiple predictions: with different rotation degrees (the index indicates the degree of rotation). The fusion layer merges all the predictions together by reversed rotations and element-wise additions

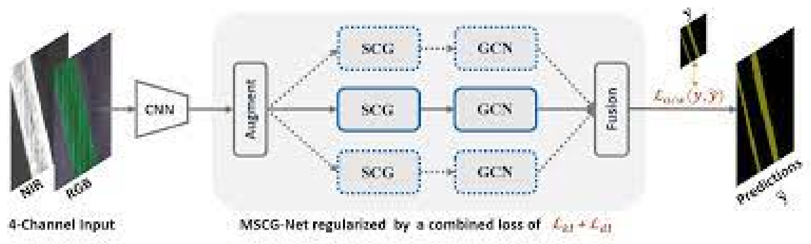


Figure 3.1: Model Architecture for MSCG-Net for semantic segmentation

3.3 DETR

Detr[45] is a new architecture that was proposed to improve existing object detection algorithms by replacing the Region of Interest(ROI)) methods with a Transformer based set prediction method that can predict the exact number of required boxes. Transformers[45] have been proposed to replace RNN based architectures with attention so as to prevent the problem of gradient exploding and also improve the overall features learnt.

Detr infers a fixed-size set of N predictions, in a single pass through the decoder, where N is set to be significantly larger than the typical number of objects in an image. One of the main difficulties of training is to score predicted objects (class, position, size) with respect to the ground truth. Our loss produces an optimal bipartite matching between predicted and ground truth objects, and then optimize object-specific (bounding box) losses.

Let us denote by y the ground truth set of objects, and $= \{_i\}_{i=1}^{N}$ the set of N predictions. Assuming N is larger than the number of objects in the image, we consider y also as a set of size N padded with $\varnothing$ (no object). To find a bipartite matching between these two sets we search for a permutation of N elements $\sigma \in \Sigma_N$ with the lowest cost:

$$\hat{\sigma} =_{\sigma \in \Sigma_N} \sum_{i}^{N} y_{i},_{\sigma(i)}, \tag{3.1}$$

where $y_{i},_{\sigma(i)}$ is a pair-wise *matching cost* between ground truth y_i and a prediction with index $\sigma(i)$. This optimal assignment is computed efficiently with the Hungarian algorithm, following prior work.

The matching cost takes into account both the class prediction and the similarity of predicted and ground truth boxes. Each element i of the ground truth set can be seen as a $y_i = (c_i, b_i)$ where c_i is the target class label (which may be $\varnothing$) and $b_i \in [0, 1]^4$ is a vector that defines ground truth box center coordinates and its height and width relative to the image size. For the prediction with index $\sigma(i)$ we define probability of class c_i as $_{\sigma(i)}(c_i)$ and the predicted box as $_{\sigma(i)}$. With these notations we define $y_{i},_{\sigma(i)}$ as $-c_i \neq \varnothing_{\sigma(i)}(c_i) + c_i \neq \varnothing b_{i},_{\sigma(i)}$.

This procedure of finding matching plays the same role as the heuristic assignment rules used to match proposal or anchors to ground truth objects in modern detectors. The main difference is that we need to find one-to-one matching for direct set prediction without duplicates.

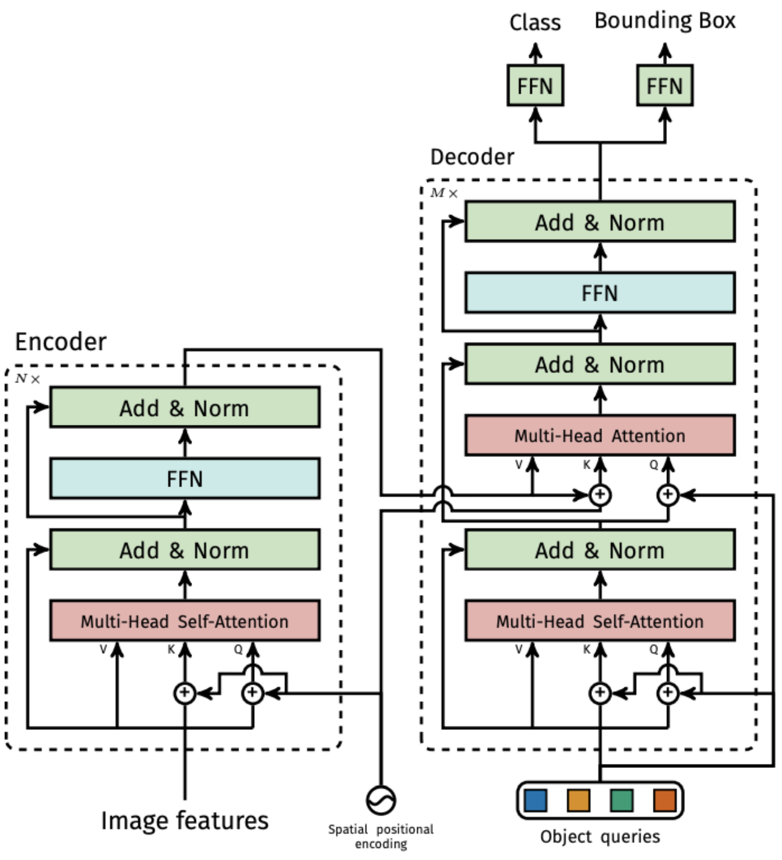


Figure 3.2: Transformer architecture used in DETR

The second step is to compute the loss function, the *Hungarian loss* for all pairs matched in the previous step. We define the loss similarly to the losses of common object detectors, a linear combination of a negative log-likelihood for class prediction and a box loss defined later:

$$y, = \sum_{i=1}^{N} \left[-\log_{\hat{\sigma}(i)}(c_i) + c_i \neq \varnothing b_{i,\hat{\sigma}}\,(i) \right] \,, \tag{3.2}$$

where $\hat{\sigma}$ is the optimal assignment computed in the first step (3.1). In practice, we down-weight the log-probability term when $c_i = \varnothing$ by a factor 10 to account for class imbalance. This is analogous to how Faster R-CNN training procedure balances positive/negative proposals by subsampling. Notice that the matching cost between an object and $\varnothing$ doesn't depend on the prediction, which means that in that case the cost is a constant. In the matching cost we use probabilities $_{\hat{\sigma}(i)}(c_i)$ instead of log-probabilities. This makes the class prediction term commensurable to $\cdot, \cdot$ (described below), and we observed better empirical performances.

3.4 DETR for semantic segmentation

Extending DETR for semantic segmentation is a fairly complicated task. The model needs to be used such that we can recreate the segmented images from the embedding vector provided by the DETR architecture. In the figure below, we show how DETR architecture can be combined with upsampling layers to create a image segmentation model. We replace the decoder module used in DETR with a decoder similar to the one used in U-Net such that the image segmentation works.

3.4.1 Input model for DETR

We tried two different methods for providing the images as inputs to the DETR model:

1. **ResNet Backbone**: The resnet model is used to extract the features from the images which is given as input to the transformer model. ResNet follows VGG's full 33 convolutional layer design. The residual block has two 33 convolutional layers with the same number of output channels. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. Then, we skip these two convolution operations and add the input directly before the final ReLU activation function. This kind of design requires that the output of the two convolutional layers has to be of the same shape as the input, so that they can be added together. If we want to change the number of channels, we need to introduce an additional 11 convolutional layer to transform the input into the desired shape for the addition operation.

2. **Patch Embeddings**: In this type, we reshape the image $x \in \mathbb{R}^{H\times W\times C}$ into a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N\times(P^2\cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection

Based on our analysis of the two input methods, we found out that using CNN backbone based DETR produced the best results for this particular task.

3.4.2 Loss Functions

Further to improve the accuracy of our models, we used different loss functions for semantic segmentation. The distribution of the classes is highly imbalanced in the dataset (e.g. most pixels in the images belongs to the background class and only few belong to classes such as planter skip and standing water). To address this problem, most existing methods make use of weighted loss functions with pre-computed class weights based on the pixel frequency of the entire training data to scale the loss for each class-pixel according to the fixed weight before computing gradients. In this work, we introduce a novel class weighting method based on iterative batch-wise class rectification, instead of pre-computing the fixed weights over the whole dataset.

The proposed adaptive class weighting method is derived from median frequency balancing weights. We first compute the pixel-frequency of class j over all the past training steps as follows

$$f_j^t = \frac{\hat{f}_j^t + (t-1) * f_j^{t-1}}{t} \ . \tag{3.3}$$

where, $t \in \{1, 2, ..., \infty\}$ is the current training iteration number, $\hat{f}_j^t$ denotes the pixel-frequency of class j at the current t-th training step that can be computed as $\frac{\mathrm{SUM}(y_j)}{\sum_{j \in C} \mathrm{SUM}(y_j)}$, and $f_j^0 = 0$.

The iterative median frequency class weights can thus be computed as

$$w_j^t = \frac{\text{median}(\{f_j^t | j \in C\})}{f_j^t + \epsilon} \ . \tag{3.4}$$

here, C denotes the number of labels (7 in this paper), and $\epsilon = 10^{-5}$.

Then we normalize the iterative weights with adaptive broadcasting to pixel-wise level such that

$$\tilde{w}_{ij} = \frac{w_j^t}{\sum_{j \in C}(w_j^t)} * (1 + y_{ij} + \tilde{y}_{ij}) \ , \tag{3.5}$$

where $\tilde{y}_{ij} \in (0, 1)$ and $y_{ij} \in \{0, 1\}$ denote the ij-th prediction and the ground-truth of class j separately in the current training samples.

In addition, instead of using traditional cross-entropy function which focuses on positive samples, we introduce a positive and negative class balanced function (PNC) which is defined as

$$\boldsymbol{p} = \boldsymbol{e} - \log\left(\frac{1 - \boldsymbol{e}}{1 + \boldsymbol{e}}\right) \ , \tag{3.6}$$

where $\boldsymbol{e} = (\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$.

Building on the dice coefficient with our adaptive class weighting PNC function, we develop an adaptive multi-class weighting (ACW) loss function for multi-class

Model	mIOU(%)
DETR - ResNet50	54.8
DETR - ResNet101	56.1
DETR - ResNeXT50	**57.1**
DETR - Patch Embeddings	54.2
MSCG-Net - ResNet50	55.0
MSCG-Net - Ensemble	66.2

Table 3.1: Results obtained for our model compared to the existing state-of-the-art

segmentation tasks

$$\mathcal{L}_{acw} = \frac{1}{|Y|} \sum_{i \in Y} \sum_{j \in C} \tilde{w}_{ij} * p_{ij} - \log \left(\text{MEAN}\{d_j | j \in C\} \right) \ , \tag{3.7}$$

where Y contains all the labeled pixels and d_j is the dice coefficient given as

$$d_j = \frac{2 \sum_{i \in Y} y_{ij} \tilde{y}_{ij}}{\sum_{i \in Y} y_{ij} + \sum_{i \in Y} \tilde{y}_{ij}} \ . \tag{3.8}$$

The overall cost function of our model, with a combination of two regularization terms $\mathcal{L}_{kl}$ and $\mathcal{L}_{dl}$ as defined in the equations, is therefore defined as

$$\mathcal{L} \leftarrow \mathcal{L}_{acw} + \mathcal{L}_{kl} + \mathcal{L}_{dl} \ . \tag{3.9}$$

3.4.3 Hyperparameters

We used grid search to find the learning rate for the model. The results of the grid search showed us that a learning rate of 0.001 with the Adam Optimizer gave the best results. We trained our models for over 50 epochs using free GPU on google colab. Further, for the decoder we needed to choose the right number of layers and found out of that six layers of convolution followed by upsampling layers gave the best results.

3.4.4 Results

The table below shows the results obtained for various models. As we can see, MSCG-Net attains a score of 55.0% for an individual model and a score of 66.2% for a ensemble-based model. Our models with a ResNeXT backbone is able to achieve a score of 57.2% which is 2% above the existing state-of-the-art at the time of writing the thesis.

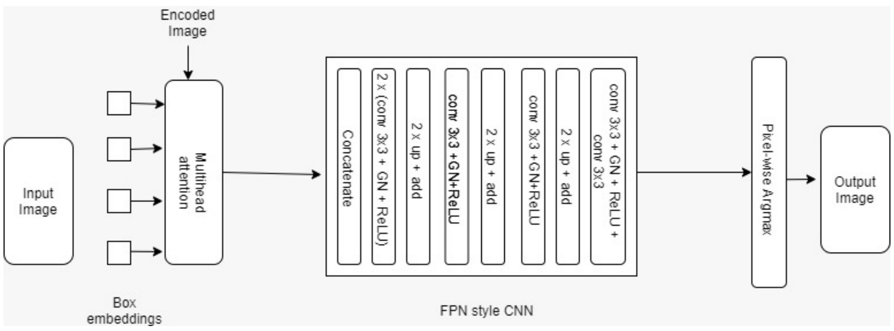


Figure 3.3: DETR model for segmentation

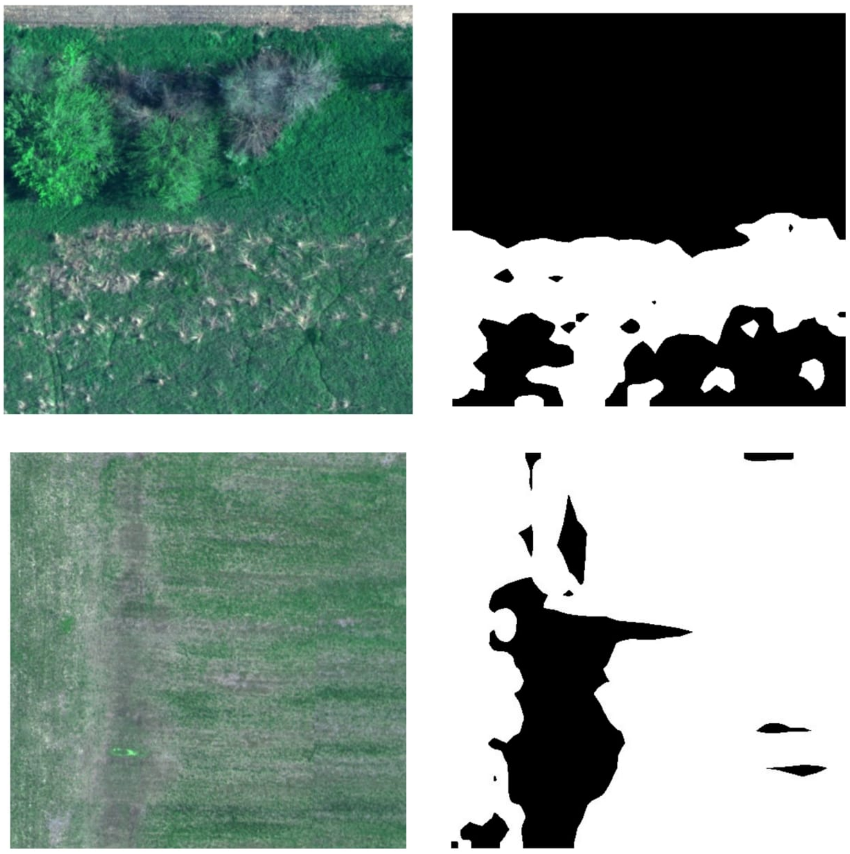


Figure 3.4: The satellite image and the masked output for weed detection

3.5 Future Work

Our future work involves replacing the CNN modules with vision transformer and further improving the attention modules used in the DETR architecture.

APPENDIX A
CODE ATTACHMENTS

A.1 DETR model used for semantic segmentation

```
import torch
import torch.nn.functional as F
from torch import nn

from util import box_ops
from util.misc import (NestedTensor, nested_tensor_from_tensor_list,
                       accuracy, get_world_size, interpolate,
                       is_dist_avail_and_initialized)

from .backbone import build_backbone
from .matcher import build_matcher
from .segmentation import (DETRsegm, PostProcessPanoptic,
    PostProcessSegm,
                           dice_loss, sigmoid_focal_loss)
from .transformer import build_transformer


class DETR(nn.Module):
    """ This is the DETR module that performs object detection """
    def __init__(self, backbone, transformer, num_classes,
            num_queries, aux_loss=False):
        """ Initializes the model.
        Parameters:
            backbone: torch module of the backbone to be used. See
                backbone.py
            transformer: torch module of the transformer architecture
                . See transformer.py
            num_classes: number of object classes
            num_queries: number of object queries, ie detection slot.
                 This is the maximal number of objects
                         DETR can detect in a single image. For COCO,
                             we recommend 100 queries.
            aux_loss: True if auxiliary decoding losses (loss at each
                 decoder layer) are to be used.
        """
        super().__init__()
        self.num_queries = num_queries
        self.transformer = transformer
        hidden_dim = transformer.d_model
        self.class_embed = nn.Linear(hidden_dim, num_classes + 1)
        self.bbox_embed = MLP(hidden_dim, hidden_dim, 4, 3)
        self.query_embed = nn.Embedding(num_queries, hidden_dim)
        self.input_proj = nn.Conv2d(backbone.num_channels, hidden_dim
            , kernel_size=1)
        self.backbone = backbone
```

```python
        self.aux_loss = aux_loss

    def forward(self, samples: NestedTensor):
        """ The forward expects a NestedTensor, which consists of:
               - samples.tensor: batched images, of shape [batch_size x 3 x H x W]
               - samples.mask: a binary mask of shape [batch_size x H x W], containing 1 on padded pixels

            It returns a dict with the following elements:
               - "pred_logits": the classification logits (including no-object) for all queries.
                                Shape= [batch_size x num_queries x (num_classes + 1)]
               - "pred_boxes": The normalized boxes coordinates for all queries, represented as
                               (center_x, center_y, height, width). These values are normalized in [0, 1],
                               relative to the size of each individual image (disregarding possible padding).
                               See PostProcess for information on how to retrieve the unnormalized bounding box.
               - "aux_outputs": Optional, only returned when auxilary losses are activated. It is a list of
                                dictionnaries containing the two above keys for each decoder layer.
        """
        if isinstance(samples, (list, torch.Tensor)):
            samples = nested_tensor_from_tensor_list(samples)
        features, pos = self.backbone(samples)

        src, mask = features[-1].decompose()
        assert mask is not None
        hs = self.transformer(self.input_proj(src), mask, self.query_embed.weight, pos[-1])[0]

        outputs_class = self.class_embed(hs)
        outputs_coord = self.bbox_embed(hs).sigmoid()
        out = {'pred_logits': outputs_class[-1], 'pred_boxes': outputs_coord[-1]}
        if self.aux_loss:
            out['aux_outputs'] = self._set_aux_loss(outputs_class, outputs_coord)
        return out

    @torch.jit.unused
    def _set_aux_loss(self, outputs_class, outputs_coord):
        # this is a workaround to make torchscript happy, as torchscript
        # doesn't support dictionary with non-homogeneous values, such
        # as a dict having both a Tensor and a list.
        return [{'pred_logits': a, 'pred_boxes': b}
```

```
                for a, b in zip(outputs_class[:-1], outputs_coord[:-1])]


class SetCriterion(nn.Module):
    """ This class computes the loss for DETR.
    The process happens in two steps:
        1) we compute hungarian assignment between ground truth boxes and the outputs of the model
        2) we supervise each pair of matched ground-truth / prediction (supervise class and box)
    """
    def __init__(self, num_classes, matcher, weight_dict, eos_coef, losses):
        """ Create the criterion.
        Parameters:
            num_classes: number of object categories, omitting the special no-object category
            matcher: module able to compute a matching between targets and proposals
            weight_dict: dict containing as key the names of the losses and as values their relative weight.
            eos_coef: relative classification weight applied to the no-object category
            losses: list of all the losses to be applied. See get_loss for list of available losses.
        """
        super().__init__()
        self.num_classes = num_classes
        self.matcher = matcher
        self.weight_dict = weight_dict
        self.eos_coef = eos_coef
        self.losses = losses
        empty_weight = torch.ones(self.num_classes + 1)
        empty_weight[-1] = self.eos_coef
        self.register_buffer('empty_weight', empty_weight)

    def loss_labels(self, outputs, targets, indices, num_boxes, log=True):
        """Classification loss (NLL)
        targets dicts must contain the key "labels" containing a tensor of dim [nb_target_boxes]
        """
        assert 'pred_logits' in outputs
        src_logits = outputs['pred_logits']

        idx = self._get_src_permutation_idx(indices)
        target_classes_o = torch.cat([t["labels"][J] for t, (_, J) in zip(targets, indices)])
        target_classes = torch.full(src_logits.shape[:2], self.num_classes,
                                    dtype=torch.int64, device=src_logits.device)
        target_classes[idx] = target_classes_o

        loss_ce = F.cross_entropy(src_logits.transpose(1, 2),
            target_classes, self.empty_weight)
        losses = {'loss_ce': loss_ce}

        if log:
            # TODO this should probably be a separate loss, not
                hacked in this one here
            losses['class_error'] = 100 - accuracy(src_logits[idx],
                target_classes_o)[0]
        return losses

    @torch.no_grad()
    def loss_cardinality(self, outputs, targets, indices, num_boxes):
        """ Compute the cardinality error, ie the absolute error in
            the number of predicted non-empty boxes
        This is not really a loss, it is intended for logging
            purposes only. It doesn't propagate gradients
        """
        pred_logits = outputs['pred_logits']
        device = pred_logits.device
        tgt_lengths = torch.as_tensor([len(v["labels"]) for v in
            targets], device=device)
        # Count the number of predictions that are NOT "no-object" (
            which is the last class)
        card_pred = (pred_logits.argmax(-1) != pred_logits.shape[-1]
            - 1).sum(1)
        card_err = F.l1_loss(card_pred.float(), tgt_lengths.float())
        losses = {'cardinality_error': card_err}
        return losses

    def loss_boxes(self, outputs, targets, indices, num_boxes):
        """Compute the losses related to the bounding boxes, the L1
            regression loss and the GIoU loss
            targets dicts must contain the key "boxes" containing a
                tensor of dim [nb_target_boxes, 4]
            The target boxes are expected in format (center_x,
                center_y, w, h), normalized by the image size.
        """
        assert 'pred_boxes' in outputs
        idx = self._get_src_permutation_idx(indices)
        src_boxes = outputs['pred_boxes'][idx]
        target_boxes = torch.cat([t['boxes'][i] for t, (_, i) in zip(
            targets, indices)], dim=0)

        loss_bbox = F.l1_loss(src_boxes, target_boxes, reduction='
            none')

        losses = {}
        losses['loss_bbox'] = loss_bbox.sum() / num_boxes

        loss_giou = 1 - torch.diag(box_ops.generalized_box_iou(
            box_ops.box_cxcywh_to_xyxy(src_boxes),
            box_ops.box_cxcywh_to_xyxy(target_boxes)))
        losses['loss_giou'] = loss_giou.sum() / num_boxes
        return losses

    def loss_masks(self, outputs, targets, indices, num_boxes):
        """Compute the losses related to the masks: the focal loss
            and the dice loss.
            targets dicts must contain the key "masks" containing a
                tensor of dim [nb_target_boxes, h, w]
        """
        assert "pred_masks" in outputs

        src_idx = self._get_src_permutation_idx(indices)
        tgt_idx = self._get_tgt_permutation_idx(indices)
        src_masks = outputs["pred_masks"]
        src_masks = src_masks[src_idx]
        masks = [t["masks"] for t in targets]
        # TODO use valid to mask invalid areas due to padding in loss
        target_masks, valid = nested_tensor_from_tensor_list(masks).
            decompose()
        target_masks = target_masks.to(src_masks)
        target_masks = target_masks[tgt_idx]

        # upsample predictions to the target size
        src_masks = interpolate(src_masks[:, None], size=target_masks
            .shape[-2:],
                                mode="bilinear", align_corners=False)
        src_masks = src_masks[:, 0].flatten(1)

        target_masks = target_masks.flatten(1)
        target_masks = target_masks.view(src_masks.shape)
        losses = {
            "loss_mask": sigmoid_focal_loss(src_masks, target_masks,
                num_boxes),
            "loss_dice": dice_loss(src_masks, target_masks, num_boxes
                ),
        }
        return losses

    def _get_src_permutation_idx(self, indices):
        # permute predictions following indices
        batch_idx = torch.cat([torch.full_like(src, i) for i, (src, _
            ) in enumerate(indices)])
        src_idx = torch.cat([src for (src, _) in indices])
        return batch_idx, src_idx

    def _get_tgt_permutation_idx(self, indices):
        # permute targets following indices
        batch_idx = torch.cat([torch.full_like(tgt, i) for i, (_, tgt
            ) in enumerate(indices)])
        tgt_idx = torch.cat([tgt for (_, tgt) in indices])
        return batch_idx, tgt_idx

    def get_loss(self, loss, outputs, targets, indices, num_boxes, **
        kwargs):
        loss_map = {
            'labels': self.loss_labels,
            'cardinality': self.loss_cardinality,
            'boxes': self.loss_boxes,
            'masks': self.loss_masks
```

```python
        }
        assert loss in loss_map, f'do you really want to compute {loss} loss?'
        return loss_map[loss](outputs, targets, indices, num_boxes, **kwargs)

    def forward(self, outputs, targets):
        """ This performs the loss computation.
        Parameters:
             outputs: dict of tensors, see the output specification of the model for the format
             targets: list of dicts, such that len(targets) == batch_size.
                      The expected keys in each dict depends on the losses applied, see each loss' doc
        """
        outputs_without_aux = {k: v for k, v in outputs.items() if k != 'aux_outputs'}

        # Retrieve the matching between the outputs of the last layer and the targets
        indices = self.matcher(outputs_without_aux, targets)

        # Compute the average number of target boxes accross all nodes, for normalization purposes
        num_boxes = sum(len(t["labels"]) for t in targets)
        num_boxes = torch.as_tensor([num_boxes], dtype=torch.float, device=next(iter(outputs.values())).device)
        if is_dist_avail_and_initialized():
            torch.distributed.all_reduce(num_boxes)
        num_boxes = torch.clamp(num_boxes / get_world_size(), min=1).item()

        # Compute all the requested losses
        losses = {}
        for loss in self.losses:
            losses.update(self.get_loss(loss, outputs, targets, indices, num_boxes))

        # In case of auxiliary losses, we repeat this process with the output of each intermediate layer.
        if 'aux_outputs' in outputs:
            for i, aux_outputs in enumerate(outputs['aux_outputs']):
                indices = self.matcher(aux_outputs, targets)
                for loss in self.losses:
                    if loss == 'masks':
                        # Intermediate masks losses are too costly to compute, we ignore them.
                        continue
                    kwargs = {}
                    if loss == 'labels':
                        # Logging is enabled only for the last layer
                        kwargs = {'log': False}
                    l_dict = self.get_loss(loss, aux_outputs, targets, indices, num_boxes, **kwargs)
                    l_dict = {k + f'_{i}': v for k, v in l_dict.items()}
                    losses.update(l_dict)

        return losses


class PostProcess(nn.Module):
    """ This module converts the model's output into the format expected by the coco api"""
    @torch.no_grad()
    def forward(self, outputs, target_sizes):
        """ Perform the computation
        Parameters:
            outputs: raw outputs of the model
            target_sizes: tensor of dimension [batch_size x 2] containing the size of each images of the batch
                          For evaluation, this must be the original image size (before any data augmentation)
                          For visualization, this should be the image size after data augment, but before padding
        """
        out_logits, out_bbox = outputs['pred_logits'], outputs['pred_boxes']

        assert len(out_logits) == len(target_sizes)
        assert target_sizes.shape[1] == 2

        prob = F.softmax(out_logits, -1)
        scores, labels = prob[..., :-1].max(-1)

        # convert to [x0, y0, x1, y1] format
        boxes = box_ops.box_cxcywh_to_xyxy(out_bbox)
        # and from relative [0, 1] to absolute [0, height] coordinates
        img_h, img_w = target_sizes.unbind(1)
        scale_fct = torch.stack([img_w, img_h, img_w, img_h], dim=1)
        boxes = boxes * scale_fct[:, None, :]

        results = [{'scores': s, 'labels': l, 'boxes': b} for s, l, b in zip(scores, labels, boxes)]

        return results


class MLP(nn.Module):
    """ Very simple multi-layer perceptron (also called FFN)"""

    def __init__(self, input_dim, hidden_dim, output_dim, num_layers):
        super().__init__()
        self.num_layers = num_layers
        h = [hidden_dim] * (num_layers - 1)
        self.layers = nn.ModuleList(nn.Linear(n, k) for n, k in zip([
            input_dim] + h, h + [output_dim]))

    def forward(self, x):
        for i, layer in enumerate(self.layers):
            x = F.relu(layer(x)) if i < self.num_layers - 1 else
                layer(x)
        return x


def build(args):
    # the `num_classes` naming here is somewhat misleading.
    # it indeed corresponds to `max_obj_id + 1`, where max_obj_id
    # is the maximum id for a class in your dataset. For example,
    # COCO has a max_obj_id of 90, so we pass `num_classes` to be 91.
    # As another example, for a dataset that has a single class with
        id 1,
    # you should pass `num_classes` to be 2 (max_obj_id + 1).
    # For more details on this, check the following discussion
    # https://github.com/facebookresearch/detr/issues/108#
        issuecomment-650269223
    num_classes = 20 if args.dataset_file != 'coco' else 91
    if args.dataset_file == "coco_panoptic":
        # for panoptic, we just add a num_classes that is large
            enough to hold
        # max_obj_id + 1, but the exact value doesn't really matter
        num_classes = 250
    device = torch.device(args.device)

    backbone = build_backbone(args)

    transformer = build_transformer(args)

    model = DETR(
        backbone,
        transformer,
        num_classes=num_classes,
        num_queries=args.num_queries,
        aux_loss=args.aux_loss,
    )
    if args.masks:
        model = DETRsegm(model, freeze_detr=(args.frozen_weights is
            not None))
    matcher = build_matcher(args)
    weight_dict = {'loss_ce': 1, 'loss_bbox': args.bbox_loss_coef}
    weight_dict['loss_giou'] = args.giou_loss_coef
    if args.masks:
        weight_dict["loss_mask"] = args.mask_loss_coef
        weight_dict["loss_dice"] = args.dice_loss_coef
    # TODO this is a hack
    if args.aux_loss:
        aux_weight_dict = {}
        for i in range(args.dec_layers - 1):
            aux_weight_dict.update({k + f'_{i}': v for k, v in
                weight_dict.items()})
        weight_dict.update(aux_weight_dict)

    losses = ['labels', 'boxes', 'cardinality']
    if args.masks:
        losses += ["masks"]
    criterion = SetCriterion(num_classes, matcher=matcher,
        weight_dict=weight_dict,
                             eos_coef=args.eos_coef, losses=losses)
    criterion.to(device)
    postprocessors = {'bbox': PostProcess()}
    if args.masks:
        postprocessors['segm'] = PostProcessSegm()
        if args.dataset_file == "coco_panoptic":
            is_thing_map = {i: i <= 90 for i in range(201)}
            postprocessors["panoptic"] = PostProcessPanoptic(
                is_thing_map, threshold=0.85)

    return model, criterion, postprocessors
```

REFERENCES

[1] Hayit Greenspan, Bram van Ginneken, and Ronald M. Summers. “Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique”. In: *CVPR* 35 (3 2016), pp. 1153 –1159.

[2] Avi Ben-Cohen, Idit Diamant, Eyal Klang, Michal Amitai, and Hayit Greenspan. “Fully convolutional network for liver segmentation and lesions detection”. In: *Deep learning and data labeling for medical applications*. Springer, 2016, pp. 77–85.

[3] Andrew Gibiansky, Sercan Arik, Gregory Diamos, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. “Deep voice 2: Multi-speaker neural text-to-speech”. In: *Advances in neural information processing systems*. 2017, pp. 2962–2970.

[4] Jose Sotelo, Soroush Mehri, Kundan Kumar, João Felipe Santos, Kyle Kastner, Aaron C. Courville, and Yoshua Bengio. “Char2Wav: End-to-End Speech Synthesis”. In: *ICLR*. 2017.

[5] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. “Generative Adversarial Text to Image Synthesis”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, 2016, 1060–1069.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018.

[7] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[9] Lihi Shiloh, Avishay Eyal, and Raja Giryes. “Efficient Processing of Distributed Acoustic Sensing Data Using a Deep Learning Approach”. In: *J. Lightwave Technol.* 37.18 (2019), pp. 4755–4762.

[10] H. Haim, S. Elmalem, R. Giryes, A. M. Bronstein, and E. Marom. "Depth Estimation From a Single Image Using Deep Learned Phase Coded Mask". In: *IEEE Transactions on Computational Imaging* 4.3 (2018), pp. 298–310.

[11] E. Schwartz, R. Giryes, and A. M. Bronstein. "DeepISP: Toward Learning an End-to-End Image Processing Pipeline". In: *IEEE Transactions on Image Processing* 28.2 (2019), pp. 912–923. ISSN: 1941-0042. DOI: 10.1109/TIP.2018.2872858.

[12] W. Yang, X. Zhang, Y. Tian, W. Wang, J. Xue, and Q. Liao. "Deep Learning for Single Image Super-Resolution: A Brief Review". In: *IEEE Transactions on Multimedia* 21.12 (2019), pp. 3106–3121. ISSN: 1941-0077. DOI: 10.1109/TMM.2019.2919431.

[13] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[14] Chang Gao, Derun Gu, Fangjun Zhang, and Yizhou Yu. "Reconet: Real-time coherent video style transfer network". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 637–653.

[15] Zhe Chen, Jing Zhang, and Dacheng Tao. "Progressive LiDAR adaptation for road detection". In: *IEEE/CAA Journal of Automatica Sinica* 6.3 (2019), pp. 693–702.

[16] Wei-Chiu Ma, Shenlong Wang, Rui Hu, Yuwen Xiong, and Raquel Urtasun. "Deep Rigid Instance Scene Flow". In: *CVPR*. 2019.

[17] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering." In: *CVPR* (2015), pp. 815–823.

[18] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wei Liu. "CosFace: Large Margin Cosine Loss for Deep Face Recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018.

[19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[20] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim. "A survey of deep learning-based network anomaly detection". In: *Cluster Computing* 22.1 (2019), pp. 949–961. ISSN: 1573-7543. DOI: 10.1007/s10586-017-1117-8.

[21] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. “Text Understanding with the Attention Sum Reader Network”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 908–918. DOI: 10.18653/v1/P16-1086.

[22] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. “Image style transfer using convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2414–2423.

[23] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual losses for real-time style transfer and super-resolution”. In: *European conference on computer vision*. Springer. 2016, pp. 694–711.

[24] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. “The Pascal Visual Object Classes (VOC) Challenge”. In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691.

[25] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. “The Role of Context for Object Detection and Semantic Segmentation in the Wild”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[26] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. “Detect What You Can: Detecting and Representing Objects using Holistic Models and Body Parts”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[27] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. Yuille. “Joint Object and Part Segmentation Using Deep Learned Potentials”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1573–1581.

[28] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. “Semantic contours from inverse detectors”. In: *2011 International Conference on Computer Vision*. 2011, pp. 991–998.

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[30] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. “The cityscapes dataset for semantic urban scene understanding”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[31] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *CoRR* abs/1511.00561 (2015).

[32] Trevor Darrell Jitendra Malik Ross B. Girshick Jeff Donahue. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013).

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. 2015, pp. 91–99.

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. “Mask R-CNN”. In: *IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2980–2988.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788.

[36] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. “SSD: Single Shot MultiBox Detector”. In: *Computer Vision - ECCV 2016 - 14th European Conference*. 2016, pp. 21–37.

[37] Jianing Sun. “Mask-YOLO: Efficient Instance-level Segmentation Network based on YOLO-V2”. In: *GitHub Repository* (2019).

[38] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. “YOLACT: Real-time Instance Segmentation”. In: *CoRR* abs/1904.02689 (2019).

[39] Ruth Rosenholtz. “Capabilities and Limitations of Peripheral Vision”. In: *Annual Review of Vision Science* 2.1 (2016), pp. 437–457. DOI: 10.1146/annurev-vision-082114-035733.

[40] W. Xu, H. Wang, F. Qi, and C. Lu. “Explicit Shape Encoding for Real-Time Instance Segmentation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 5167–5176. DOI: 10.1109/ICCV.2019.00527.

[41] R. Zhang, Z. Tian, C. Shen, M. You, and Y. Yan. “Mask Encoding for Single Shot Instance Segmentation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 10223–10232. DOI: 10.1109/CVPR42600.2020.01024.

[42] Y. Lee and J. Park. “CenterMask: Real-Time Anchor-Free Instance Segmentation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 13903–13912. DOI: `10.1109/CVPR42600.2020.01392`.

[43] E. Xie, P. Sun, X. Song, W. Wang, X. Liu, D. Liang, C. Shen, and P. Luo. “PolarMask: Single Shot Instance Segmentation With Polar Representation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 12190–12199. DOI: `10.1109/CVPR42600.2020.01221`.

[44] Qinghui Liu, Michael Kampffmeyer, Robert Jenssen, and Arnt-Børre Salberg. *Multi-view Self-Constructing Graph Convolutional Networks with Adaptive Class Weighting Loss for Semantic Segmentation*. 2020. arXiv: `2004.10327 [cs.CV]`.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017).