ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Ramazan ABDULLAH

PERFORMANCE ANALYSIS OF DEEP LEARNING OBJECT DETECTION BASED IMAGE SEGMENTATION METHODS

DEPARTMENT OF COMPUTER ENGINEERING

ADANA, 2020

ABSTRACT

MASTER THESIS

PERFORMANCE ANALYSIS OF DEEP LEARNING OBJECT DETECTION BASED IMAGE SEGMENTATION METHODS

Ramazan ABDULLAH

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Prof. Dr. Selma Ayşe ÖZEL
Year: 2020, Pages 65
Jury : Prof. Dr. Selma Ayşe ÖZEL
Assc. Prof. Dr. Mustafa ORAL
Asst. Prof. Dr. Fatih KILIÇ

Image segmentation which divides the input image into multiple regions or segments is one of the most difficult problems to be solved in the area of computer vision. Without image segmentation, understanding the images by computer is not an easy process. There are two types of image segmentation task: i) semantic segmentation, in which multiple objects from the same class are considered as the same, ii) instance segmentation where multiple objects from the same class are taken as different, therefore, each object (instance) is to be classified separately. Image segmentation is used in numerous applications such as satellite image processing, medical image processing, texture recognition, face recognition systems, automated plate recognition systems, etc. In this thesis, our aim is to apply deep learning-based object detection to perform semantic segmentation and evaluate its performance. Therefore, first, we applied YOLO to find the bounding boxes of each object in the images, then we used GrabCut and DeepGrabCut methods to classify foreground and background pixels to make semantic segmentation. DeepGrabCut is a deep learning-based version of GrabCut, and we compared their performances on two image datasets having more than 35K images. The experimental analysis has shown that object detection with object selection can be used to make semantic segmentation with acceptable error rates when optimal parameters setting was done for the methods applied.

Keywords: Deep learning, prediction, image segmentation, object detection, computer vision.

ÖZ

YÜKSEK LİSANS TEZİ

DERİN ÖĞRENME NESNE ALGILAMA TABANLI GÖRÜNTÜ BÖLÜTLEME YÖNTEMLERİNİN PERFORMANS ANALİZİ

Ramazan ABDULLAH

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Prof. Dr. Selma Ayşe ÖZEL
Yıl: 2020, Pages: 65
Jüri : Prof. Dr. Selma Ayşe ÖZEL
: Doç. Dr. Mustafa ORAL
: Dr. Öğr. Üyesi Fatih KILIÇ

Girdi görüntüsünü birden çok bölgeye veya bölüme ayıran görüntü bölümleme, bilgisayarlı görü alanında çözülmesi en zor problemlerden biridir. Görüntü bölütleme olmadan, görüntüleri bilgisayarın anlaması kolay bir işlem değildir. İki tür görüntü bölütleme görevi vardır: i) aynı sınıftan birden çok nesnenin aynı olarak kabul edildiği anlamsal bölütleme, ii) aynı sınıftaki birden çok nesnenin farklı alındığı örnek bölütleme, bu nedenle her nesne (örnek) ayrı olacak şekilde sınıflandırılmaktadır. Görüntü bölütleme, uydu görüntüsü işleme, tıbbi görüntü işleme, doku tanıma, yüz tanıma sistemleri, otomatik plaka tanıma sistemleri, vb. çok sayıda uygulamada kullanılır. Bu tezde amacımız, anlamsal bölütleme yapmak ve performansını değerlendirmek için derin öğrenme tabanlı nesne tespitini uygulamaktır. Bu nedenle, önce görüntülerdeki her nesnenin sınırlayıcı kutularını bulmak için YOLO uygulandı, ardından anlamsal bölütleme yapmak için ön plan ve arka plan piksellerini sınıflandırmak için GrabCut ve DeepGrabCut yöntemleri kullanıldı. DeepGrabCut, GrabCut'un derin öğrenme tabanlı bir sürümü olup, bu yöntemlerin performansları 35K'dan fazla görüntüye sahip iki görüntü veri kümesinde karşılaştırıldı. Deneysel analizler, nesne seçimi ile nesne algılamanın, uygulanan yöntemler için optimum parametre ayarı yapıldığında kabul edilebilir hata oranları ile anlamsal bölütleme yapmak için kullanılabileceğini göstermiştir.

Anahtar Kelimeler: Derin öğrenme, tahmin, görüntü bölütleme, nesne algılama, bilgisayarla görme.

EXTENDED ABSTRACT

With the significant advances in image capturing devices and technologies, digital images take a very important role in our life. Images can store, represent and transmit the data in a very easy way. As always, an image can represent tons of words. In computer science, images are commonly used data types to do a lot of tasks such as object detection, object tracking, recognition tasks, face verification, and etc.

In computer vision, the main task is to get useful information from the processed image. The useful information extraction is very difficult and costly for computers unlike humans. In order to make the computer be able to understand the image, a fundamental low to mid-level vision task divides the image into many meaningful parts, and each parts can answer a lot of questions about the image like how many objects does the image have? and where are all these objects? These partitioning and labeling tasks are the most complex duties of computer vision which are called as image segmentation.

Image segmentation is the computer vision task that tries to partition the input image into multiple regions, so that each region contains an object. Therefore, the difference between the image segmentation and the object detection can be defined as follows: in object detection the aim is to generate a rectangle that contains the object, and give a label representing the detected object. In the segmentation task on the other hand, the aim is to define a region containing the object. However, the labeling is in the pixel level, which means that segmentation task gives a label for each pixel, so the output is not a rectangle, instead the output is a polygon that has the object shape.

Image segmentation is very useful for understanding the image by computers such as the qualitative and quantitative properties of the objects in the segmented regions of the image. This understanding can be used in other computer vision tasks such as pedestrian detection, localization of objects in satellite images, face

recognition, image compression, image editing, image retrieval, etc. There exist numerous image segmentation methods and the most widely used methods can be listed as follows:

The first method is the threshold segmentation method that uses a threshold to make the segmentation. This method is commonly used over the grayscale images. The threshold value is defined automatically according to the region pixel colors. In other words, it is a clustering process over the selected region.

The second method is the regional growth segmentation that groups the pixels that have similar properties to define the segmented region. Thus, segmentation is done by finding the seed pixels, then grouping the seed's neighborhood pixels that have similar properties to define the segmented region.

The third technique is called as the edge detection segmentation method. This method can be used for colored and gray images. The main idea of this method is to find many regions with a discontinuity detection area of the edge to generate the segmentation result.

The fourth method is the segmentation based on clustering. The main idea of this method is to use the similarity between pixel groups to generate the pixel labels (image segmentation). First the image is divided into a number of subclasses then many iterations are done to merge the similar subclasses as the main class. This grouping depends on the similarity and the distance between the subclasses.

The last and the most modern one is the segmentation based on supervised learning by using the CNN. In this method, the main idea is to generate feature maps which are used to generate a mask for the image. The generated mask is used as the labeling map for all the pixels.

In this study, we use GrabCut object selection algorithm with deep learning for image segmentation. Basically, GrabCut is applied for foreground extraction, which means to extract the selected object from the image and remove all the unrelated pixels to the selected object, thus it needs bounding box that contains the object.

In this study; we used two versions of the GrabCut object selection method to do semantic segmentation. The first one, namely GrabCut, is the traditional method that does not require any training step and classifies foreground and background pixels in the given bounding box for an object in the image. The second method, namely DeepGrabCut, is the deep learning-based version of GrabCut, and it requires a training step for doing object selection. Both GrabCut and DeepGrabCut methods require the bounding box of the object to classify its foreground and background pixels. Therefore, we used YOLO, which is a deep learning-based method, to predict the bounding box for the object in the given image. In this thesis we used YOLOv3 and Tiny-YOLOv3 versions, and we apply the below four combinations to make semantic segmentation, and compare their performances:

- YOLOv3 + GrabCut.
- YOLOv3 + DeepGrapCut.
- Tiny-YOLOv3 + GrabCut.
- Tiny-YOLOv3 + DeepGrapCut.

While the object detection part cannot generate perfect boxes, some boxes are small for the detected object, and some other boxes are bigger. Having inappropriate sized boxes causes some problems for the object segmentation. For the traditional GrabCut, the provided box (by the YOLO) is used directly. However, in the Deep learning-based version (DeepGrabCut) there is a skip method to avoid the error in the detection. Instead of using only the predicted box, the algorithm generates various boxes with different sizes using the size of the predicted box, then these multiple boxes are used to make the object selection. So the algorithm doesn't take just one box for every object. According to the results that have been obtained from the whole experiments that have been done in this thesis, this method (object detection with object selection) can be used to make the segmentation task with

acceptable error rates. However, to obtain more accurate semantic segmentation, better object detection methods are required.

GENİŞLETİLMİŞ ÖZET

Görüntü alma cihazları ve teknolojilerindeki önemli gelişmelerle birlikte dijital görüntüler hayatımızda çok önemli bir rol oynamaktadır. Görüntüler, verileri çok kolay bir şekilde saklayabilir, temsil edebilir ve iletebilir. Her zaman olduğu gibi bir görüntü tonlarca kelimeyi temsil edebilir. Bilgisayar biliminde görüntüler, nesne algılama, nesne izleme, tanıma, yüz doğrulama vb. birçok görevi yerine getirmek için yaygın olarak kullanılan bir veri türüdür.

Bilgisayarlı görüde asıl görev, önceden işlenmiş görüntüden faydalı bilgiler elde etmektir. Bu bilgilerin çıkarılması, insanlardan farklı olarak bilgisayarlar için çok zor ve maliyetlidir. Bilgisayarın görüntüyü anlayabilir hale gelebilmesi için, görüntüyü birçok anlamlı bölüme bölmek gerekir. Böylelikle bilgisayar her bir bölümde görüntünün kaç nesneye sahip olduğu, tüm bu nesnelerin nerede olduğu gibi görüntü ile ilgili birçok soruyu cevaplayabilir. Bu bölütleme ve etiketleme işlemleri görüntü bölütleme adı verilen bilgisayar görüsünün en karmaşık görevidir.

Görüntü bölütleme (segmentasyon), girdi görüntüsünü birden fazla bölgeye bölmeye çalışan bilgisayarlı görü görevidir. Her bölge bir nesne içerir. Görüntü bölütleme ile nesne algılama arasındaki fark şu şekilde özetlenebilir: Nesne algılamada amaç nesneyi içeren bir dikdörtgen oluşturmak ve dikdörtgene algılanan nesneyi temsil eden bir etiket vermektir. Ancak bölütlemedeki amaç ise, nesneyi içiren bir bölge tanımlamaktır. Fakat etiketleme piksel seviyesindedir, bu nedenle her piksel için bir etiket verir verilir, böylece çıktı bir dikdörtgen değil, nesnenin şekline sahip bir çokgendir.

Görüntünün bölümlere ayrılması, görüntünün bölümlenmiş bölgelerindeki nesnelerin niteliksel ve niceliksel özellikleri gibi bilgisayarlarla görüntüyü anlamak için çok yararlıdır. Bu anlayış, yaya tespiti, uydu görüntülerinde nesnelerin yerelleştirilmesi, yüz tanıma, görüntü sıkıştırma, görüntü düzenleme veya görüntü erişim gibi diğer bilgisayarlı görü görevlerinde kullanılabilir. Birçok görüntü

bölütleme yöntemi kullanılmaktadır. En yaygın kullanılan yöntemler aşağıdaki gibi listelenebilir:

Bu yöntemlerden ilki, eşik bölütleme yöntemidir. Bu yöntemde bölütleme yapmak için bir eşik değeri kullanılır. Genellikle gri tonlamalı görüntülerde uygulanır. Eşik değeri, bölge piksel renklerine göre otomatik olarak tanımlanır. Aslında bu yöntem, seçilen bölge üzerinde uygulanan bir kümeleme işlemidir.

İkinci yöntem, bölgesel büyüme bölütlemedir. Bu yöntemde, bölütlü bölgeyi tanımlamak için benzer özelliklere sahip pikseller gruplanır. Böylelikle bölütleme tohum pikseller bulunarak yapılır, daha sonra tohum benzer özelliklere sahip olan komşu pikselleri bölütlenmiş bölgeyi tanımlamak için gruplandırır.

Üçüncü yöntem, kenar algılama bölütleme yöntemi olarak isimlendirilir. Bu yöntem renkli ve gri görüntüler için kullanılabilir. Bu yönteminin ana fikri, bölütleme sonucunu oluşturmak için kenarın süreksizlik algılama alanına sahip birçok bölge bulmaktır.

Dördüncü yöntem, kümelenmeye dayalı bölütlemedir. Bu yöntemin ana fikri, pikselin etiketini oluşturmak için piksel grupları arasında benzerliği kullanmaktır (görüntü bölütleme). Önce görüntü bir dizi alt sınıfa bölünür, ardından benzer alt sınıfları birleştirmek için birçok yineleme yapılır. Bu yöntemde gruplama ait sınıflar arasında benzerlik ve mesafeye bağlıdır.

Sonuncu ve en modern olan yöntem ise, CNN kullanarak denetimli öğrenmeye dayalı bölütlemedir. Bu yöntemde ana fikir, görüntü için bir maske oluşturmak amacıyla kullanılan özellik haritalarını oluşturmaktır. Oluşturulan maske, tüm pikseller için etiketleme haritasıdır.

Bu çalışmada görüntü bölütleme için GrabCut nesne seçimi algoritması derin öğrenme ile birlikte kullanılmaktadır. Temel olarak GrabCut arka planın elde edilmesi için kullanılır. Diğer bir deyimle, görüntüden seçilen bir nesneyi çıkarmak ve seçilen nesne ile ilgili olmayan pikselleri ortadan kaldırmak için uygulanır. Bu nedenle GrabCut algoritması seçilecek nesneyi içeren bir kapsayıcı dikdörtgene ihtiyaç duyar.

Bu çalışmada anlamsal bölütleme yapmak için GrabCut nesne seçme yönteminin iki versiyonu kullanılmıştır. İlk yöntem GrabCut olarak isimlendirilen, herhangi bir öğrenme aşamasına ihtiyaç duymayan ve verilen bir çevreleyici dikdörtgendeki ön-plan ve arka-plan piksellerini sınıflandıran geleneksel yöntemdir. İkinci yöntem, yani Deep GrabCut, GrabCut'un derin öğrenme tabanlı sürümüdür ve nesne seçimi yapmak için bir eğitim adımı gerektirir. Hem GrabCut hem de DeepGrabCut yöntemleri, ön plan ve arka plan piksellerini sınıflandırmak için nesnenin sınırlayıcı kutusunu gerektirir. Bu nedenle, verilen görüntüdeki nesnenin sınırlayıcı kutusunu tahmin etmek için derin öğrenme tabanlı bir yöntem olan YOLO kullandık. Bu tezde YOLOv3 ve Tiny-YOLOv3 versiyonları kullanıldı ve aşağıdaki dört kombinasyon ile anlamsal bölütleme yapıldı ve performansları karşılaştırıldı:

- YOLOv3 + GrabCut.
- YOLOv3 + DeepGrapCut.
- Tiny-YOLOv3 + GrabCut.
- Tiny-YOLOv3 + DeepGrapCut.

Nesne algılama yöntemi mükemmel çevreleyici kutular oluşturamazken, algılanan nesne için oluşturulan bazı kutular küçük, bazıları ise daha büyük olmaktadır. Uygun olmayan şekilde boyutlandırılmış kutulara sahip olmak, nesne bölütleme için bazı sorunlara neden olmaktadır. Geleneksel GrabCut yöntemi için, (YOLO tarafından) elde edilen çevreleyici kutu doğrudan kullanılır. Bununla birlikte, derin öğrenme tabanlı sürümde (DeepGrabCut) ise, tespit hatasını önlemek için bir atlama yöntemi vardır. DeepGrabCut algoritması, yalnızca tahmin edilen kutuyu kullanmak yerine, tahmin edilen kutunun boyutunu kullanarak farklı boyutlarda çeşitli kutular oluşturur, ardından bu çoklu kutular nesne seçimini yapmak için kullanılır. Dolayısıyla, algoritma her nesne için yalnızca bir kutu kullanmaz. Bu tezde yapılan deneylerden elde edilen sonuçlara göre, bölütleme görevini kabul edilebilir hata oranlarıyla yapmak için nesne seçimi ile nesne tespiti

yönteminin en iyi parametre ayarları ile kullanılabileceği görülmektedir. Ancak, daha doğru anlamsal bölütleme elde etmek için daha iyi nesne algılama yöntemleri gereklidir.

ACKNOWLEDGEMENT

Primarily, I would like to express my sincere gratitude to my advisor Prof. Dr. Selma Ayşe ÖZEL, for her motivation, patience and enthusiasm. Her guidance helped me in all the time of research and writing of this thesis.

I would like to thank members of the MSc thesis jury, Assc. Prof. Dr. Mustafa ORAL and Asst. Prof. Dr. Fatih KILIÇ, for their suggestions and corrections.

Last but not the least, I would like to thank my family for their endless support and encouragement for my life and career.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF ABBREVIATIONS

CV : Computer Vision
FCN : Fully Connected Network
mIoU : mean of Intersection over Union
FPN : Featured Pyramid Network
PSPNet : Pyramid Scene Parsing Network
CNN : Convolutional Neural Networks
MLP : Multi-Layer Perceptron
FC : Fully Connected
YOLO : You Only Look Once
IOU : Intersection Over Union
CEDN : Convolutional Encoder-Decoder Network

1. INTRODUCTION

Recently, the image segmentation task became an important process in computer vision, image analysis, and image understanding systems. It means to divide the input image into multiple regions or segments (R. Szeliski, 2010). Segmentation has a wide application in different areas (D. Forsyth et al., 2002), from finding the tumor boundary and tissue volumes in the medical images to detect the pedestrian for autonomous cars, also to augmented reality, and various videos analyzing applications.

Numerous image segmentation algorithms have been developed, from the traditional methods where the image segmentation is based on clustering technique such as thresholding (N. Otsu, 1979), histogram-based bundling, region growing (R. Nock et al., 2004), k-means clustering (N. Dhanachandra et al., 2015), watersheds (L. Najman et al., 1994), to more modern algorithms like active contours (M. Kass et al., 1988), graph cuts (Y. Boykov et al., 2001), conditional and Markov random fields (N. Plath et al., 2009), and sparsity-based (J. -L et al., 2005) methods. However, in recent years and after deep learning become more popular, various network models have been released, and have better performance. In some cases, the most difficult two problems in computer vision (both the image segmentation and object detection) can be solved in the same network.

The image segmentation problem can be reduced to a labeling or classification task, but at the pixel level. It means in the image segmentation process a class label is assigned for every pixel. The segmentation task can be separated into two types. The first one considers the multiple objects of the same class as a single category (e.g., home, bird, tree), this segmentation type is called as semantic segmentation. The second type is the instance segmentation, which considers the multiple objects of the same class as separate classes (or instances). Obviously, for

computers, the instance segmentation is more difficult than the semantic segmentation.

In this thesis, our aim is to perform semantic segmentation by applying traditional and deep learning-based methods and compare their performances. In order to make image segmentation, two different methods, namely GrabCut and DeepGrabCut, are applied to separate the pixels belong to the object (i.e., foreground) from the background pixels. GrabCut is a traditional method that separates foreground and background pixels so that it does not need any training step. However, DeepGrabCut is a deep learning-based method that applies the GrabCut algorithm, and it requires a training step over a training image dataset to classify image pixels as foreground and background. Both GrabCut and DeepGrabCut methods require a bounding box for each object to detect foreground and background pixels. In order to find bounding boxes for each object in the image, we use YOLO which is a fast object detection method. Therefore, in this thesis, first, we apply object detection by YOLO, then we use GrabCut and DeepGrabCut for object selection to perform semantic segmentation of the images. Then, we compare their performances.

The rest of this thesis is organized as follows: in the next part a brief summary of the related work is given. In section 3 background information about the object selection methods and object detectors are presented. Section 3 also covers other materials and methods used in this thesis. Section 4 shows the experimental results and discussions. Finally, section 5 concludes our study.

2. RELATED WORK

In this part, the image segmentation studies that use deep learning and their results on some segmentation challenge are presented.

Fully Connected Network is an end-to-end network for image segmentation developed by (J. Long et al., 2015). The main idea of FCN started by modifying the common neural networks like Alexa, VGG16, and GoogleNet which have a fixed input size for the images. The authors changed the fully connected layers into convolutional layers which make the network take non-fixed size input. The network generates a feature map as a first step, and then make upsampling the features map to generate the output with the pixel-wise classification (segmented output). (J. Long et al., 2015) trained the network using a pixel-wise loss function. Also, in the network, they added skip connections to concatenate the high-level feature map with the dense and specific feature maps. In the 2012 PASCAL VOC segmentation challenge, FCN has achieved a 62.2% mIoU score after training the models on the 2012 ImageNet dataset, and the model achieved 78.8% mIoU in object detection for the same challenge.

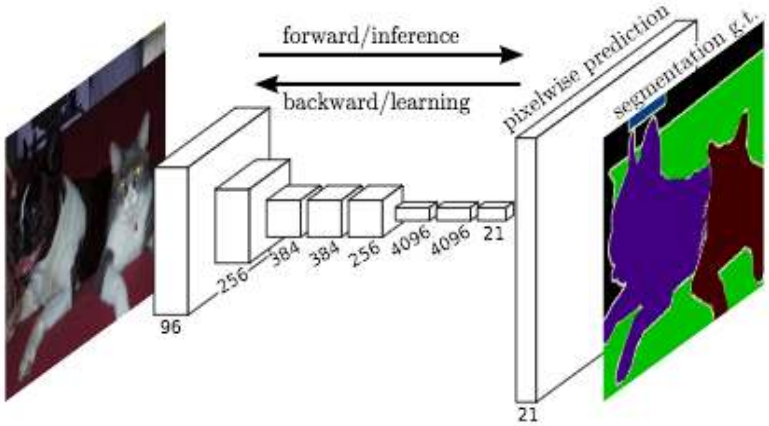


Figure 2.1. FCN Architecture (J. Long et al., 2015).

(W. Liu et al., 2015) noted that FCN has a weak point that causes losing the global context in the deep layers. Thus, it makes the representation of the feature map not so accurate. To solve this problem, they published ParseNet which is like FCN an end-to-end architecture. But unlike FCN, ParseNet doesn't take regions as input (keep the global image context), it predicts labels for whole image pixels at the same time. (W. Liu et al., 2015) used a model to take features maps as input, this model processes input using two ways. At first, the global pooling layer is applied to the input, after that the output of the pooling layer is normalized by using the L2 Euclidean Norm, and then an unpooling process is done (this unpooling layer makes the output size the same as the input size). The second way normalizes the initial feature maps (here there is no pooling) by applying the L2 Euclidean Normalizing, after that the result of the first and the second ways are concatenated. The normalization process makes the concatenation result stay in a scaled state. As two feature maps have been concatenated, the performance gets better. As a result, the authors achieved a 40.4% mIoU score on the PASCAL-Context challenge, and for the segmentation part of the same challenge the architecture obtained a 69.8% mIoU score. Figure 2.2 shows the architecture of ParseNet.

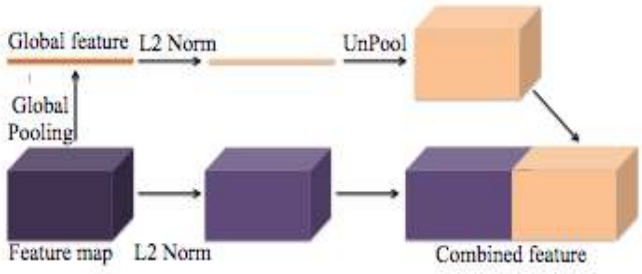


Figure 2.2. ParseNet architecture. (W. Liu et al., 2015).

In 2015 a new end-to-end model was published by (H. Noh et al., 2015) called Convolutional and Deconvolutional Network, where they used two linked

deep neural networks. The first one has a VGG architecture. This part is used for generating the features in vector shape. Then, this features vector is fed to the second part which is a deconvolutional network. The second part converts the features vector into feature maps by using unpooling and deconvolutional layers. Finally, all feature maps are concatenated to generate the segmented output image.

This network on the 2012 PASCAL VOC segmentation challenge had a 72.5% mIoU. Figure 2.3 shows the Convolutional and Deconvolutional Network architecture.

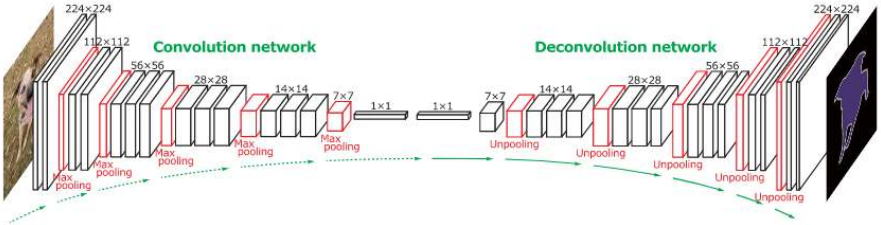


Figure 2.3. Convolutional and Deconvolutional Networks architecture (H. Noh et al. 2015).

The FCN of (J. Long et al., 2015) again has been modified by (O. Ronneberger et al., 2015). They built a network composed of two parts. The first one is to get the feature maps and the second one is to give a class of every pixel. The first part (downsampling part) has an FCN-like architecture with 3*3 convolutions, the second part (upsampling part) uses a deconvolutional layer which makes the feature maps size changes by increasing the height and width, also decreasing the number of feature maps or the channel value. The authors called the network as U-Net. While there are connections from the downsampling part to the upsampling part in different levels which gives the network its shape like a U letter. But the aim of these connections is to crop the features maps to avoid losing the image context. After all, a 1*1 convolutional layer is used to create the segmented result. The U-net is a commonly used architecture and have been based on the U-Net the recent

architectures such as FPN and PSPNet etc. because U-net can be trained with small datasets. According to (J. Long et al., 2015) experiments, 30 images were found as enough to train the U-net network. Figure 2.4 shows the U-net network architecture.

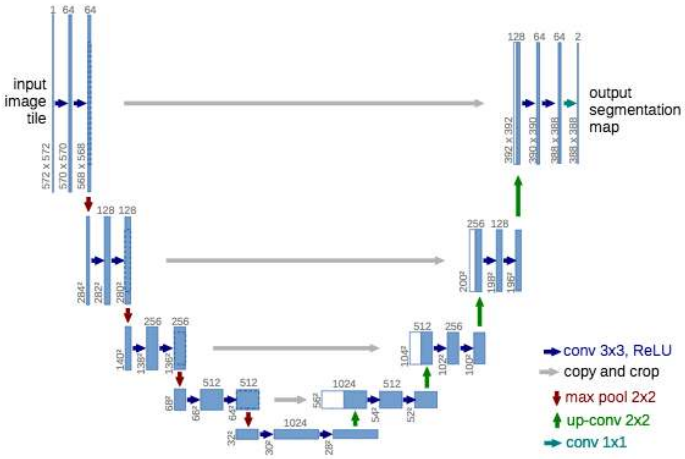


Figure 2.4 The U-net architecture (O. Ronneberger et al., 2015).

In 2016 T.-Y. Lin et al published their paper of FPN (Feature Pyramid Network) which can be used for both segmentation and object detection. This method is more accurate, because it feeds the network with less information. The FPN (T.-Y. Lin et al., 2016) is composed of two pathways: bottom-up and top-down; also has side connections between the two pathways to merge the low-resolution and the high-resolution features. In the bottom-up pathway, the input has a fixed size, the downsampling is done by using the pooling layers. The authors named the feature maps that have the same size as a stage, just the stage outputs are used in the pyramid level. The top-down pathway makes the upsampling process by unpooling the feature maps that are generated by the downsampling part (bottom-up pathway), which is transmitted using the side connections (lateral connections) that merge the feature maps of the downsampling part by using a 1*1 convolution with the feature maps of the upsampling part (top-down pathway). After merging the feature maps,

they are fed to a 3*3 convolution to generate the stage's output. Each stage in the top-down pathway gives a prediction to detect an object. And for the segmentation tasks, there are two MLP networks. Each MLP generates a mask. This technique is used in many deep learning architectures such as R-CNN (R. Girshick et al., 2014), Fast R-CNN (R. Girshick et al., 2015), Faster R-CNN (S. Ren et al., 2016). The FPN is based on DeepMask (P. 0. Pinheiro et al., 2015) and SharpMask (P. 0. Pinheiro et al., 2016) frameworks achieved a 48.1% Average Recall (AR) score on the 2016 COCO segmentation challenge. Figure 2.5 shows the model.

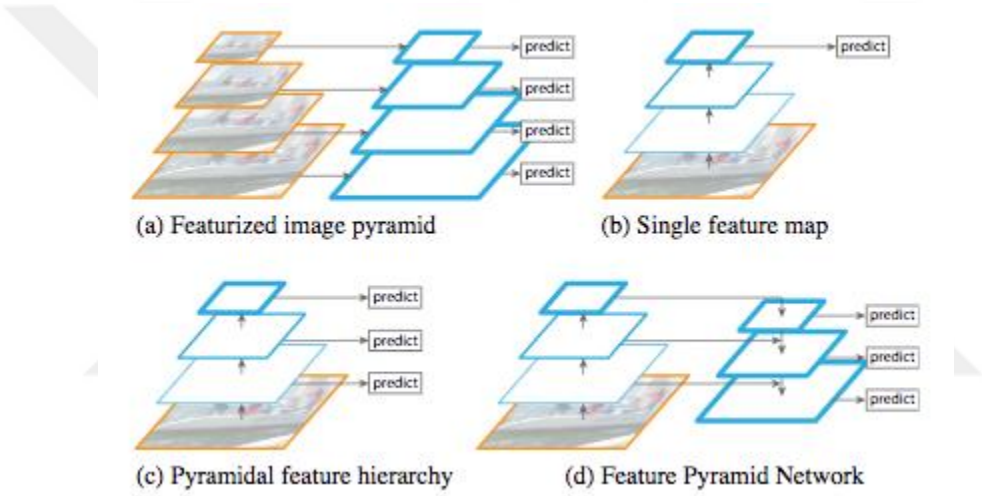


Figure 2.5. Comparison of architectures. (a): The image is scaled with several sizes and each one is processed with convolutions to provide predictions which is computationally expansive. (b): The image has a single scale processed by a CNN with convolution pooling layers. Each step of the CNN is used to provide a prediction. (d) Architecture of the FPN with the bottom-up part of the left and the top-down part on the right. (T.-Y. Lin et al., 2016).

While extracting the feature maps is the main problem in the image segmentation, (H. Zhao et al., 2016) published their paper with a feature extractor using an expanded version of the ResNet (K. He et al., 2015), and they called this new architecture as Pyramid Scene Parsing Network (PSPNet) where the change of extractor in the network leads to generate a better global context representation. The

feature maps are obtained in a pyramid pooling module with four different pooling layers of different levels. Each level has a convolutional layer (with 1*1 filters size) to decrease the dimensions. By using this method, the pyramid analyses the sub-regions of images. Then, the initial features are generated by upsampling and concatenating the four levels' output. Finally, a convolutional layer is used to generate the pixel-level classification. On the 2012 PASCAL VOC segmentation challenge, the PSPNet has scored an 85% mIoU. Figure 2.6 shows the PSPNet architecture.

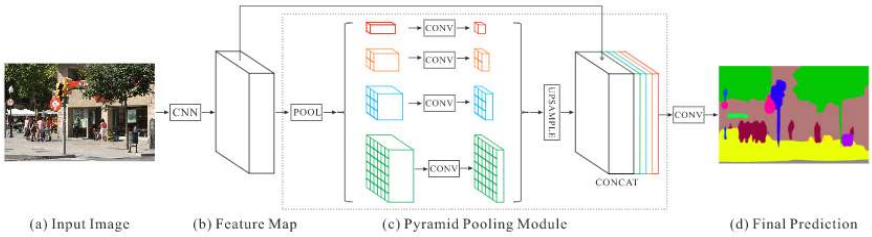


Figure 2.6. PSPNet architecture (H. Zhao et al., 2016).

While the Faster R-CNN (S. Ren et al., 2015) is an object detection architecture using the RPN (Region Proposal Network) to obtain the bounding boxes candidates, (K. He et al., 2017) built the Mask R-CNN network which is nothing else than Faster R-CNN but give three outputs: the first one is the box coordinates, the second one shows the class of the detected box, and the third one is an FCN network so that its input is the detected box and its output is a binary mask. So, in this model, there is a multi-task loss function that comes from the error over the detection part and the segmentation part. The best score on the 2017 COCO segmentation challenge for the Mask R-CNN is 41.8% AP score. Figure 2.7 shows the architecture of the Mask R-CNN.

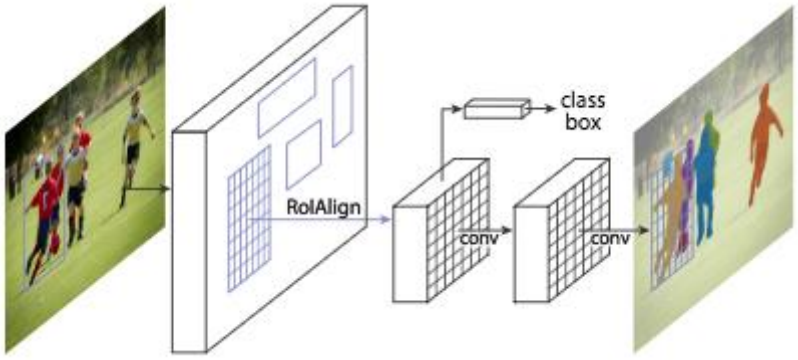


Figure 2.7. Architecture of Mask R-CNN (K. He et al., 2017).

According to (L.-C. Chen et al., 2017) the consecutive max-pooling and all of the striding processes decrease the feature maps. So, they proposed the DeepLab network that uses the new Atrous convolution, which were inspired by the convolution of (H. Zhao et al., 2016). Atrous convolution captures multiple scales of the objects without increasing the number of weights. It consists of filters targeting the spares pixels with the fixed rate. Therefore, when the rate is assigned to 2, the filter targets one pixel instead of two, and the Atrous returns to normal convolution when the rate is equal to 1. Figure 2.8 shows the difference between the normal convolution and the Atrous convolution.

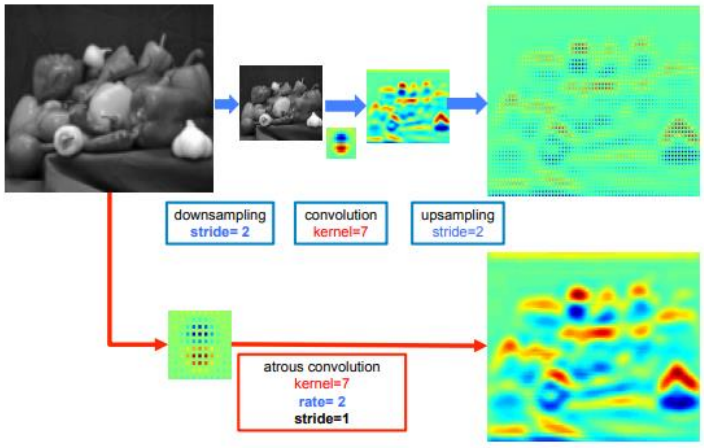


Figure 2.8. Normal convolution and the Atrous convolution output visualization (L.-C. Chen et al., 2017).

Applying the several Atrous convolution with different rates for the same input and combining them will give the network a pyramid architecture, which is called the Atrous Spatial Pyramid Pooling (ASPP). After generating the feature maps, a bilinear interpolation process is applied over the concatenated feature maps, and thus a result with the input size is generated After that, the output is processed with a fully connected CRF (Conditional Random Field) (Krähenbühl and V. Koltun, 2012) which takes the edges between the features and long-term dependencies to generate the segmented image.

On the 2012 PASCAL VOC challenge, the DeepLab scored a 79.7% mIoU. And on the PASCAL-Context challenge the DeepLab scored a 45.7% mIoU. Figure 2.9 shows the ASPP.

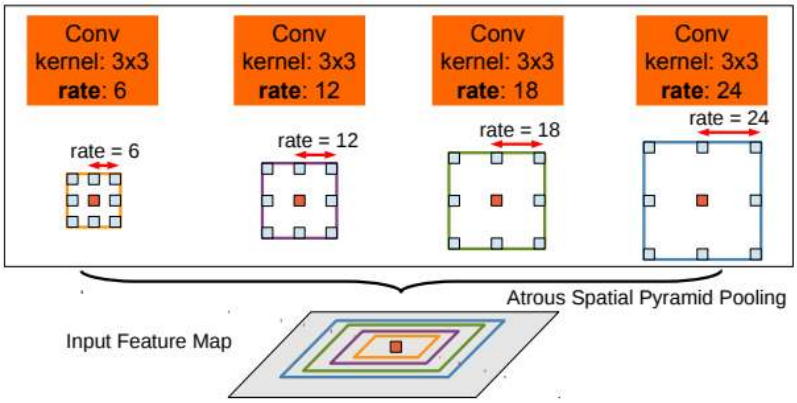


Figure 2.9. Atrous Spatial Pyramid Pooling (ASPP). (L.-C. Chen et al., 2017).

After DeepLab 1 and 2, a lot of research were done to improve it, and the study performed by (L.-C. Chen et al., 2017) published the DeepLab3 which is another study to improve its performance. DeepLab3 combines two modules such that both uses the Atrous convolutions, but one of them is parallel, and the other one is a cascaded module. The authors used an architecture similar to ResNet, but with Atrous convolution. On the 2012 PASCAL VOC challenge, the highest score obtained by DeepLabv3 is 86.9% mIoU.

(L.-C. Chen et al., 2018) again developed the DeepLab3 architecture by using an encoder-decoder structure. But they used the DeepLap3 as an encoder part, and they called the new architecture as DeepLab3+. The authors made two improvements: The first one is Atrous separable convolution which is composed of a depth wise convolution (spatial convolution for each channel of the input); and the second one is pointwise convolution (1x1 convolution with the depth wise convolution as input). The outputs of the ASPP are processed by a 1x1 convolution and upsampled by a factor of 4. The outputs of the encoder CNN are also processed by another 1x1 convolution and concatenated to the previous ones. The feature maps are fed to two 3x3 convolutional layers and the outputs are upsampled by a factor of

4 to create the final segmented image. The DeepLab3+ scored an 89.0% mIoU on the 2012 PASCAL VOC challenge.

In this study, in order to segment the input images, we used an architecture which has two parts: the detection part, and the selection part. The main idea is inspired by Mask-RCNN, which is to use object-detection in image segmentation. In Mask-RCNN the detection task is done in two steps: first by finding the object, then classifying it. In this thesis we used the YOLO algorithm as an object detector, which makes the detection in one step, thus it makes the task faster. Also, Mask-RCNN uses an FCN architecture for segmenting the boundary boxes, but we used two selection algorithms in two separated approaches. The first one is the GrabCut which doesn't need any training and gives a good performance in case of non-overlapped objects. For the second approach, we used the DeepGrabCut which is similar to Convolutional and Deconvolutional Networks, but the main difference is that there is a random generating and selection mechanism before processing the boxes to avoid the error of the object detection network. Therefore, in this thesis, we experimentally analyze the performance of both GrabCut and DeepGrabCut methods on the image segmentation problem and compare them with the existing methods that were used for this purpose.

3. MATERIALS AND METHODS

In this section, the datasets, the deep learning architectures, the object detection algorithm YOLO, and the object selection methods that are used in this thesis are presented. After that, we explain how the methods are used in this thesis.

3.1. Datasets

In this study three datasets, namely The PASCAL VOC dataset (2012), Common Objects in COntext (COCO), and The Semantic Boundaries Dataset (SBD), are used to measure the performance of the used methods. [1]The details of the datasets are explained in the below paragraphs.

The PASCAL VOC dataset (2012)[1] is widely used for object detection and segmentation tasks. The dataset contains 11k images for training and validation, also contains 10k images for testing. For evaluating the results of the image segmentation, the mIoU metric (mean Intersection over union) is used. The IoU is the result of dividing the area of overlap and the area of union between the ground truth and the predicted areas. Thus, the mIoU is the mean of the IoU of all segmented objects over all the images of the test dataset.

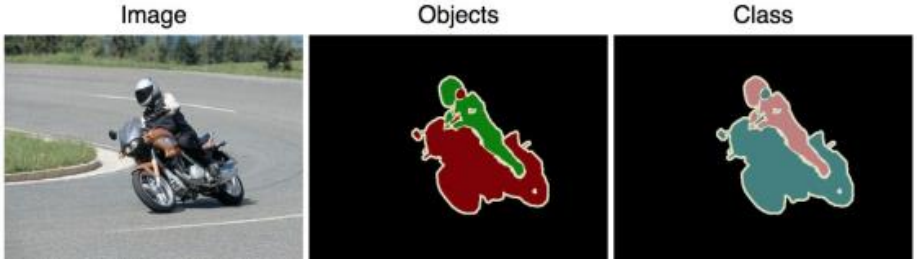


Figure 3.1. An example of the PASCAL VOC image dataset. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html)

[1] The PASCAL VOC dataset is available at http://host.robots.ox.ac.uk/pascal/VOC/

The COCO dataset[2] is one of the commonly used datasets for object detection and image segmentation studies. This dataset contains 200k images separated into four sub-datasets, training, validation, and two testing datasets one for the developers and researchers called test-dev and another one for the challenge called test-challenge. But the annotations of the two test sub-datasets are not provided. The COCO dataset contains 80 classes. For performance evaluation the IoU is used in both the AP (Average Precision) and the AR (Average Recall) computation. Figure 3.2. shows an example from the COCO dataset images.

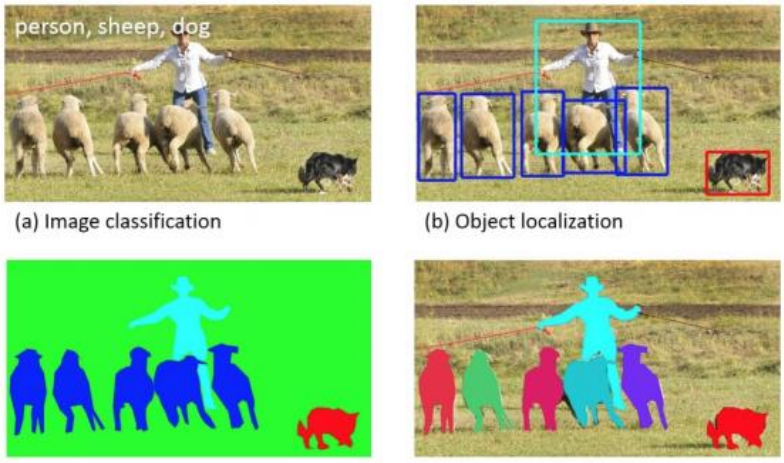


Figure 3.2. An example from the COCO dataset images. (http://cocodataset.org/)

The Semantic Boundaries Dataset (SBD) [3] is created to make the semantic contour, which is unlike the semantic segmentation (where the prediction is for the pixels inside the object), the prediction is for the pixels of the object boundary, which is obviously harder than the semantic segmentation. This dataset provides semantic segmentation and the boundaries annotation for 11355 images separated into 20 classes of objects. Figure 3.3 shows an example of SBD dataset images.

[2] The COCO dataset is available at https://cocodataset.org/
[3] The Semantic Boundaries dataset is available at http://home.bharathh.info/pubs/codes/SBD/download.html

Figure 3.3. An example from the SBD dataset images. (http://home.bharathh.info/pubs/codes/SBD/download.html)

3.2. Convolutional Neural Network

The Convolutional Neural Networks (CNNs) are the most commonly used architecture in deep learning which was proposed by LeCun et al. (1989). In the following years, it has been used for different Computer Vision tasks including image segmentation. The main task of CNNs is to extract the features from the images then the output of the network depends on the task that the network is designed for. CNNs architecture contains mainly three layers: The input layer, the hidden layers, and of course the output layer.

The input layer, in general cases, has the following shape [*w, h, c*], where the *w, h, c* are the width, height, and depth of the input image, respectively. The output layer consists of several layers of Fully Connected layers (FC). These layers' duty is to convert the extracted features into a specific shape that fits the aim of the used architecture. The hidden layer consists of different types of layers. Those layers are convolutions followed by pooling layers in general case. Figure 3.4 shows a simple CNN architecture for image classification.

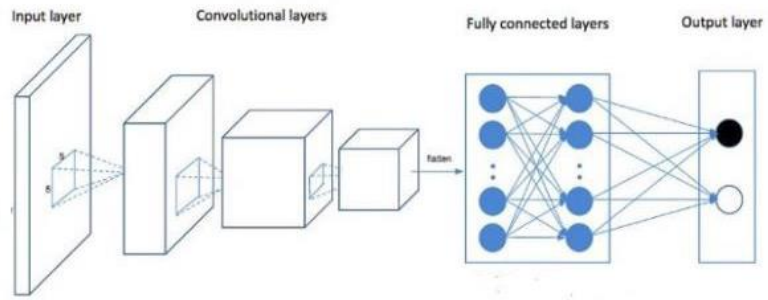


Figure 3.4. Basic CNN structure for image classification (Kim, 2014)

3.2.1. Convolution Layer

The Convolution layer is the mostly used layer in CNNs architecture. In the computer vision field, the main aim of convolutions is to detect edges, colors, corners from the input image (features extracting). Also, the deeper layers inside the network are used for extracting more complex features like shapes, digits, face parts, and so on (Source: https://cs231n.github.io/convolutional-networks/).

These features are extracted by using a dot product between two matrices: the first one is called kernels or filters, which are the learnable parameters of the network; and the other matrix is a part of the image (by using the stride the multiplication covers all the pixels). For the RGB images (width*height*3) the kernels have smaller width and height than the input image, but the depth parameter or the channel parameter is equal to the third parameter of the image. The output depth is equal to the number of kernels.

While the kernels don't have the same image width and height, a stride operation is needed to apply over all the kernels for all the image pixels. Stride refers to how many units the filters move over the input data. Figure 3.5. shows an example of applying the stride.

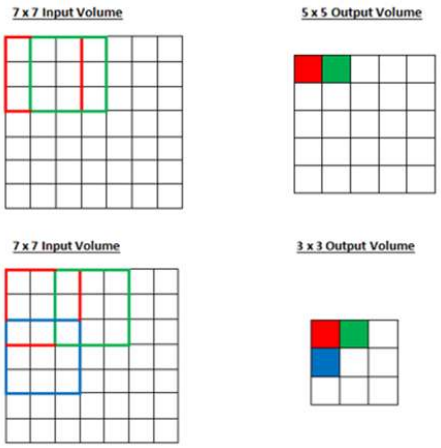


Figure 3.5. Application of a 3x3 filter (kernel) to an input data size of 7x7 for Stride=1 (top), and Stride=2 (bottom). (https://adeshpande3.github.io/ABeginner%27s-Guide-To-Understanding-Convolutional-NeuralNetworks-Part-2)

In Figure 3.5, a 3x3 filter (kernel) is applied to an input data size of 7x7 for stride = 1 at the top, and stride = 2 at the bottom. Filter moves 1 unit in *x* and *y*-directions when stride = 1 and moves 2 units in *x* and *y*-directions when stride = 2. For some cases, to prevent the size of input data from shrinking, a padding with zeros can be done. Stride, padding, kernels size, and kernels numbers are called as hyperparameters of the CNN and changing them affect the features extraction process that have a role on the learning ability of the network.

The size of the convolution output is explained by the equation 3.1 (Source: https://cs231n.github.io/convolutional-networks/), where *output* is size of output feature map, *input* is size of the input, *filter* is size of the filter, *padding* is size of padding value, and *stride* is the stride value.

$$output = \frac{input - filter + 2 * padding}{stride} + 1 \qquad (3.1)$$

3.2.2. Activation Function

Like the normal neural networks, after every convolution, there is an activation function. The most popular activation functions in CNNs are ReLU, sigmoid, hyperbolic tangent, and the Softmax, as they are explained as follows (http://cs231n.github.io/neuralnetworks-1):

The Rectified Linear Units (ReLU) function is shown in Figure 3.6, and the formula of it is computed as in equation 3.2, where *x* is the input.

$$f(x) = \begin{cases} 0 & if \quad x < 0 \\ x & otherwise \end{cases} \tag{3.2}$$

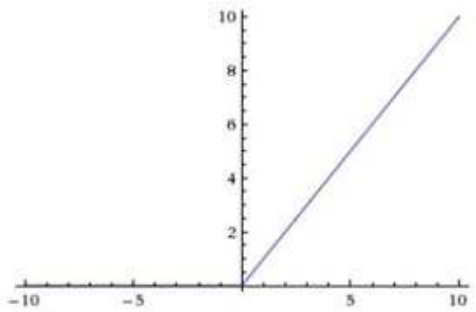


Figure 3.6. ReLU activation function. (http://cs231n.github.io/neuralnetworks-1)

The Sigmoid function is given in equation 3.3, where *x* is the input, and Figure 3.7 shows the sigmoid function graphically.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

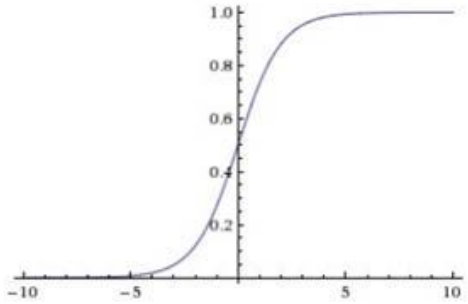


Figure 3.7. The Sigmoid function. (http://cs231n.github.io/neuralnetworks-1)

The Hyperbolic Tangent function is given in equation 3.4, where *x* is the input, and Figure 3.8. shows the Hyperbolic Tangent function.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1 \qquad (3.4)$$

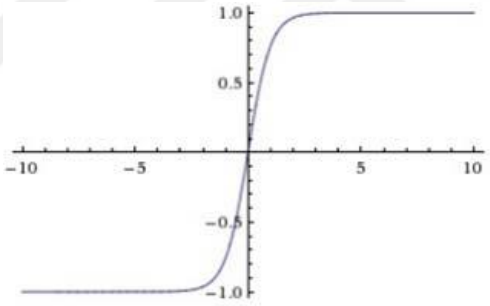


Figure 3.8. The Hyperbolic Tangent function. (http://cs231n.github.io/neuralnetworks-1)

The last activation function, which is also very popular, is the **Softmax** function, which is computed as in equation 3.5 (https://deepai.org/machine-learning-glossary-and-terms/softmax-layer), where *x* is the input. Figure 3.9 shows the Softmax function graphically.

$$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad for\ i = 1, \ldots\ldots, k\ and\ x = (x_1, \ldots\ldots, x_k)\ \in R^k \qquad (3.5)$$

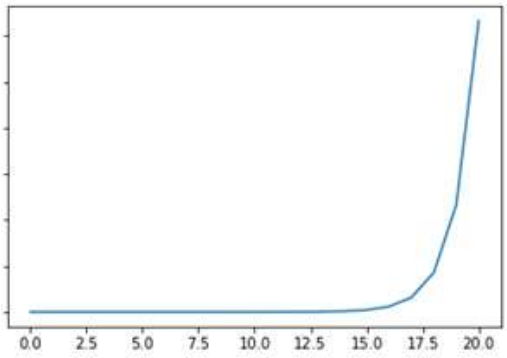


Figure 3.9. The Softmax Function. (https://deepai.org/machine-learning-glossary-and-terms/softmax-layer)

3.2.3. Pooling Layer

The pooling layer is a very important layer because of its ability to reduce the dimensions (just width and height without changing the depth value) of the input. This reduction makes the calculations easier and less complex; also makes the feature extraction more accurate. Pooling layers also contain sliding windows, but unlike the convolutional layer it doesn't have weights to learn, or even to be multiplied with the input values. There are two types of pooling layers:

1.Max-pool: passes the maximum value in the applied window into the output.

2.Average-pool: passes the average value in the applied window into the output.

The output size after applying the pooling operation is explained in equations 3.6 and 3.7 where *f* the filter size. The two dimensions *H* and *W* decreases after the pooling operation but for the channel *c* value no change occurs. Figure 3.10 shows a Max Pooling Layer (Source: https://cs231n.github.io/convolutional-networks).

$$H = \left\lfloor \frac{H - f}{stride} \right\rfloor + 1 \qquad (3.6)$$

$$W = \left\lfloor \frac{W - f}{stride} \right\rfloor + 1 \tag{3.7}$$

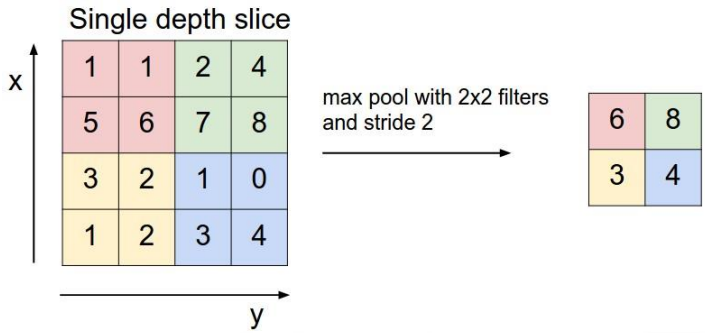


Figure 3.10. The Max Pooling Layer. (https://cs231n.github.io/convolutional-networks/)

3.2.4. Output Layer

This layer consists of FC layers of the multi-layer perceptrons. The input of this layer is the output of the last pooling layer. Probability of classes is computed by weighting these inputs through various activation functions.

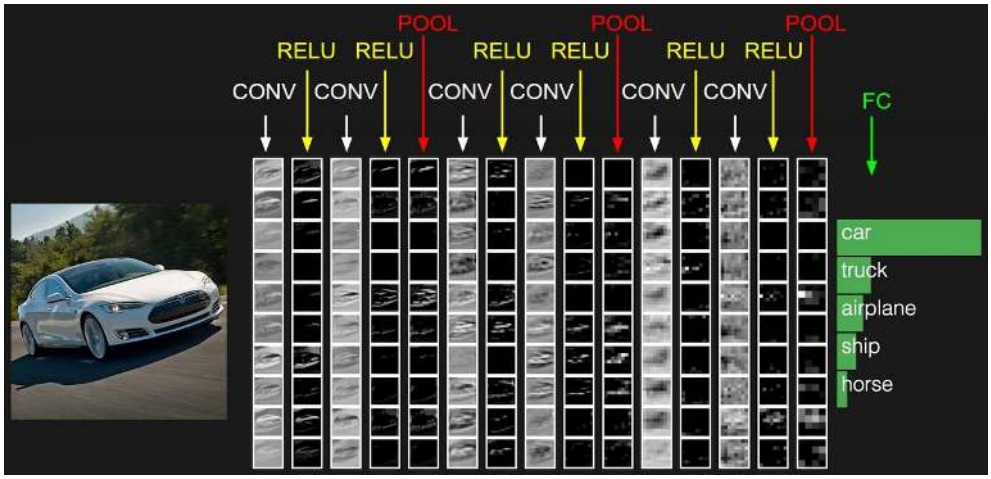


Figure 3.11. An Example of Image classifier. (https://cs231n.github.io/convolutional-networks).

The use of CNN for image classification can be explained through Figure 3.11 (Zhang and Wallace, 2017) as follows: in the input layer, there is an image in RGB. The input size is determined by the image size, or the image can be resized before it is given as input to the network. A sequence of convolution, ReLU activation function, and pooling layers are applied on the input image. Finally, in the output layer (Fully Connected layer); predictions for each class label are obtained by applying the Softmax activation function.

3.2.5. ResNet

At the past the deep learning architectures were started with a few layers (e.g., AlexNet), but this shallow architecture was not so accurate for the complex tasks. Thus, this result led to the use of deeper network architectures. Using the deeper networks provides extracting more complex features that cannot be extracted in the shallow networks. However, using a deeper network doesn't always give the best performance. Because in the training operation, the network faces the problem of losing gradients. Also, the network takes a long time to train, because the gradient signal gets the zero value faster than the shallow networks. So, (He et al., 2015) published their architecture of the residual network (called ResNet) which depends on skip connection between layers, thus it gives the ability to use deeper architecture without affecting the training process. Figure 3.12 shows the learning ability of the CNN model with respect to the number of.

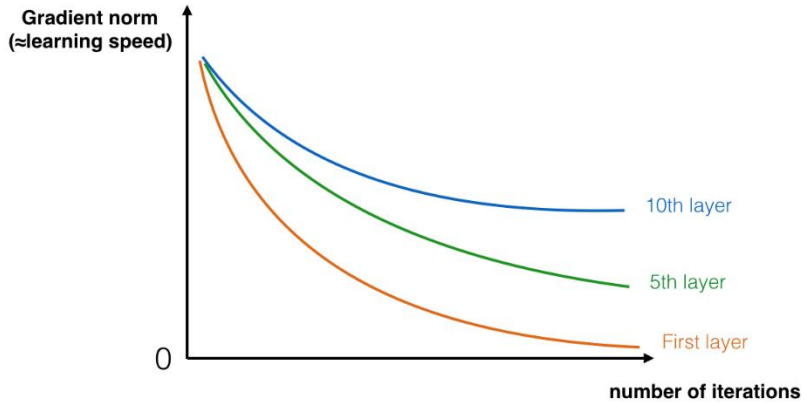


Figure 3.12. Learning speed of CNN with respect to the number of iterations. (https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/9ddRV/lab).

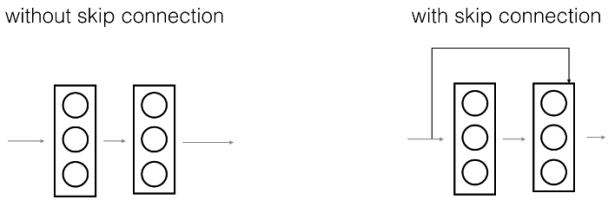


Figure 3.16. The normal connection (on the left) and the connection with shortcut (on the right). (Source: https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/9ddRV/lab).

To build the ResNet, (He et al., 2015) added a "shortcut" or a "skip connection", which makes the model skip layers. Skipping layers means a shallower network. Figure 3.16 shows the normal connection between two layers (on the left) which is called as the main path, and the skip connection (on the right) by using a sequence of these blocks a ResNet can be built.

(He et al. 2015) said that using the skip connection makes the training faster, and the performance higher. In addition, (He et al. 2015) proposed two types of skip connection blocks depending on the input and the output size (same or different), and they called them as “identity block” and “convolutional block”.

The identity block is the core of the ResNet idea. It is used when the input of the first layer and the output of the last layer have the same size. It is done by

taking the input of the first layer and adding it to the output of the last one. To get the output of the block, the ReLU function is applied over the addition. Figure 3.17 shows the structure of the identity block.

In addition, (He et al., 2015) presented that the identity block can skip 3 layers, not just one, but the addition is still before the activation function of the last layer.

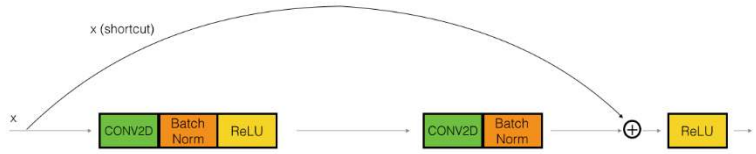


Figure 3.17. Identity block over two layers. (https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/9ddRV/lab).

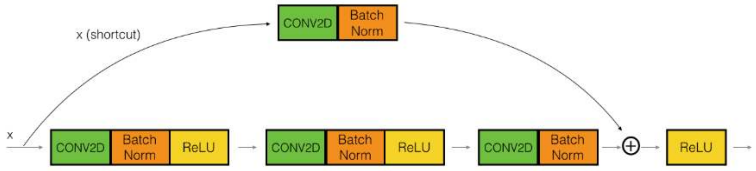


Figure 3.18. Convolutional block over three layers. (https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/9ddRV/lab).

The convolutional block is the second block type. This type is used in case of the input and the output (input and output of the block) don't have the same dimensions. The convolutional block is similar to the identity block. However, there is a convolution layer on the skip connection. This convolution changes the size to be fitted with the size of the output of the last layer, also, they are added before feeding to the activation function. The convolution in the skip connection doesn’t have an activation function. And in the same case in the identity block, the

convolution block can skip over 3 hidden layers. Figure 3.18 shows the Convolutional block over three layers.

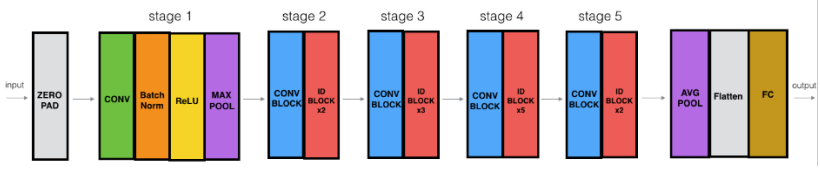


Figure 3.19. ResNet-50 architecture (https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/9ddRV/lab).

Figure 3.19 shows the ResNet-50 architecture which contains five stages with average pooling and fully connected layer as an output layer. The ResNet-101 is used in this study because of its ability to learn faster than the other deep networks, also it can get the complex features from the image which is very important for the segmentation algorithms.

3.3. Object Detection

Object detection is a well-known task in computer vision. Basically, object detection is the process of classification and defining the location of the objects in a frame or image. The result of object detection is a box that contains the object with the class label that is given to that object such as person, car, etc.

The task of the image classification is to define the class of the input image and give the image just one class label; but in object detection the duty is more complex; and the classification is done not for the whole image. Object detection is a localization task, which means finding every object in the image and classifying that object. So, the result of the object detection is a boundary box containing an object and also a class label assigned for the area of the image defined by the boundary box. The object detection algorithms can be separated into two types:

The first type of algorithms contains two steps to make the localization of the objects. The first step is to create the potential boxes on the input, then these

boxes need to have class labels, so a CNN classifier is applied over the boxes. Now some objects have been detected by more than one box, so an eliminating process is run to detect and eliminate the duplicates. This process is slow and not good for the training operation, because the training must have been done over the classes separately. There are numerous algorithms that depends on this technique such as (RCNN) and there are improved versions of RCNN like Fast-RCNN, and Faster-RCNN.

The second type image detection algorithms use regression. In this type of methods, instead of making the boxes detection and then classification in separate steps; both of the box detection and the class label generation for the detected box are done together in the same run. YOLO algorithm is one of the most famous algorithms which uses this technique.

3.3.1. YOLO

YOLO is one of the most commonly used object detectors. (Redmon et al., 2016) proposed the first YOLO architecture. YOLO is one of the fastest detectors because it sends the whole image to CNN to predict multiple boxes, each box has a score for every class, which makes the learning ability of YOLO better than the other algorithms. While YOLO is from the second group of detectors, it converts the detection task to regression task.

The output of YOLO algorithm is in the shape of $(S*S) * B * (5+C)$ (Redmon et al. 2016). This output is generated by splitting the input image into separated $(S*S)$ grid cells. Each cell has its own prediction values. If one of these cells contains a midpoint of an object, this cell is responsible to represent that object. This representation contains a number B of boxes, and each box has its confidence score. The confidence score represents the existence or absence of any object inside the boundary box. Equation 3.8 (Redmon et al. 2016) shows the formula of confidence where the $Pr(Object)$ represents the probability of object existence. Figure 3.20

shows the IOU calculation method, where IoU is equal to the division of area of overlap by the area of union.

$$Confidence = Pr(Object) * IoU \quad (3.8)$$

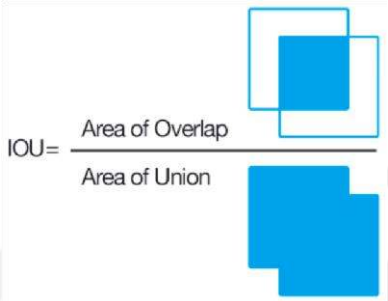


Figure 3.20. The IoU calculation method. (https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

When the cell doesn't contain any object, the confidence score is zero. So, YOLO gives for each detected box the five variables that are *x, y, w, h* and the score of the confidence. Where the first two variables (*x, y*) show the coordinates of the midpoint of the boundary boxes, the second two variables (*w, h*) show the width and height. Also, for every box, there is a predicted class which is represented by the *c* variable called the class score. To calculate the class score, YOLO multiplies the class probability with the box confidence score as given in equation 3.9 (Redmon et al. 2016).

$$\Pr(Class_i|Object) * \Pr(Object) * IOU_{predicted}^{ground\ truth} = \Pr(Class_i) * IOU_{predicted}^{ground\ truth} \quad (3.9)$$

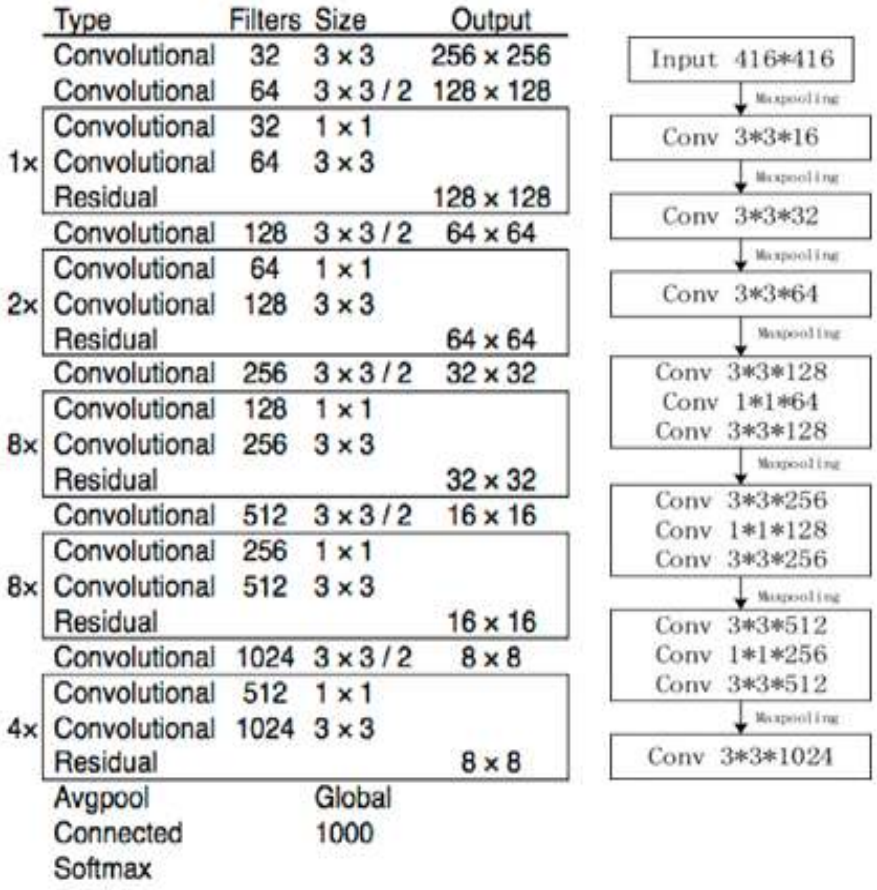
YOLOv3 architecture consists of 24 convolutional layers with LReLU (leaky ReLU) activation function ended with two FC (fully connected layers) which give the output of size *(S*S) * B * (5+C).* YOLO has been trained by using a specific

loss function which is the summation for the squared error (Redmon et al., 2016). The YOLO loss function is shown in Equation 3.10 (Redmon et al. 2016).

$$
\begin{aligned}
loss \quad = \; & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x_\iota})^2 + (y_i - \hat{y_\iota})^2] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w_\iota}}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h_\iota}}\right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(C_i - \hat{C_\iota}\right)^2 \right] + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left[\left(C_i - \hat{C_\iota}\right)^2 \right] \\
& + \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \widehat{p_\iota(c)})^2 \qquad (3.10)
\end{aligned}
$$

Where, x_i, y_i, w_i, h_i belong to the boundary boxes; c_i is used for the confidence score; $p_i(c)$ is the classification loss. The λ is used to control the effect of the difference of loss on the total loss value. (Redmon et al) set the $\lambda_{coord} = 5$ *and* $\lambda_{noobj} = 0.5$ the 1^{obj}_i represents the object that appears in the cell *i* the 1^{obj}_{ij} represents which predicted boundary box (*j*) is responsible for defining the detected object in the cell (*i*) (Redmon et al. 2016).

YOLO looks only once to the image to localize the objects. Also, it doesn't contain complex operations. All of these reasons make YOLO a very fast detector. Also, the version 3 of YOLO has the ability to detect small objects. Because of the mentioned specifications of YOLOv3 and Tiny-YOLOv3, they are used in this study. Where Tiny-YOLOv3 is a small version of the YOLOv3 and used for mobile devices and some small pcs. The architecture that is used in this study is shown in Figure 3.21 where (a) is the architecture of full YOLOv3 and (b) is for Tiny-YOLOv3.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

(a) (b)

Figure 3.21. (a)YOLOv3 architecture, (b) Tiny-YOLOv3 architecture. (https://pylessons.medium.com/yolo-v3-theory-explained-33100f6d193).

3.4. Object Selection

In this thesis, object selection methods are used for semantic image segmentation, and these methods are categorized under the two main headings: The traditional object selection method, and the deep learning-based method.

3.4.1. Traditional Object Selection Method: GrabCut

GrabCut (Rother, C. et al., 2004) is one of the traditional interactive foreground (object) extraction algorithms, which depend on separating the foreground from the background in the given box. This algorithm converts the input image into a graph, which is built by using a minimum cost reduction function, to generate the segmented version of the image. The vertices in the graph are the pixels, and there are two additional vertices. One of these extra vertices represents the background called sink, and the second vertex represents the foreground called the source. The edge weights are calculated from the color similarity and the regions in the image. Both of the added vertices are linked to all pixel vertices, and each edge weight is the probability of the vertices (pixel) to match the color distribution of the background or foreground. So, GrabCut defines two labels: one for the foreground, and one for the background. After that, it tries to classify all the pixels into two labels. To segment the graph, a Min-Cut/Max-Flow method is used; also, a Gaussian Mixture Models (GMMs) generates the region info. The steps behind the GrabCut are as follows:

The algorithm takes an image, and a rectangle (R) contains the part of the image to be segmented as input. The algorithm starts by defining three rectangles: i) *RB* represents the initial values of the background, ii) *RF* represents the foreground, and iii) *RU* represents the unknown pixels. The algorithm considers that all pixels out of the defined rectangle R belong to the background (*RB*). Also, the pixels in the R area are considered as unknown pixels (*RU*) where the algorithm aims to label the *RU* pixels as *RB* or *RF*. The result is a mask with four labels (*0,1,2,3*). 0 is for the sure background,1 is for the sure foreground, 2 is for the probable background, and 3 is for probable foreground. After that the final result is just the 1 and 3 labeled pixels as foreground, and the remaining pixels are counted as background. Figure 3.22 shows the GrabCut algorithm steps.

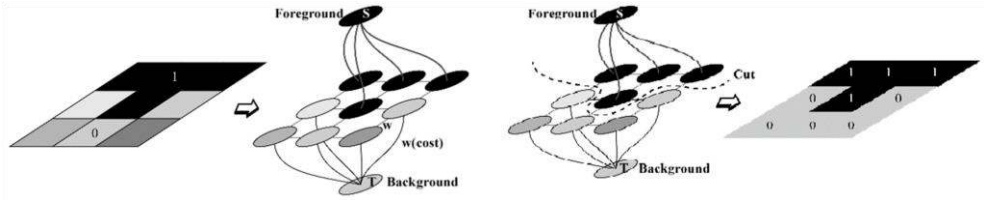


Figure 3.22. The Steps of GrabCut Algorithm. (a) the input image, (b) build the graph with two extra vertices (foreground and background), (c) cut the graph, (d) convert the graph to mask with (0) label as background and (1) as foreground. (Source: http://www.cs.ru.ac.za/research/g02m1682/)

3.4.2. Deep Learning based Object Selection: DeepGrabCut

The traditional GrabCut method considers that the boundary boxes perfectly contain the object. But in our case, the boundary boxes are generated by a deep learning model, so the boxes are not perfect; some boxes may be too large, or too small, or shifted, which may affect the result of the traditional GrabCut. However, in the deep version of the GrabCut, the situation is different. The boxes are processed before using them. The DeepGrabCut (Xu, N. et al., 2017) uses a convolutional encoder-decoder network (CEDN) to give the segmented result. Figure 3.23 shows the steps of DeepGrabCut algorithm, which start by taking the input image and converting it to a distance map. Then the generated distance map is propagated to the encoder part of the network to generate the feature map. After that, the feature map is processed by the encoder part to generate the output mask.

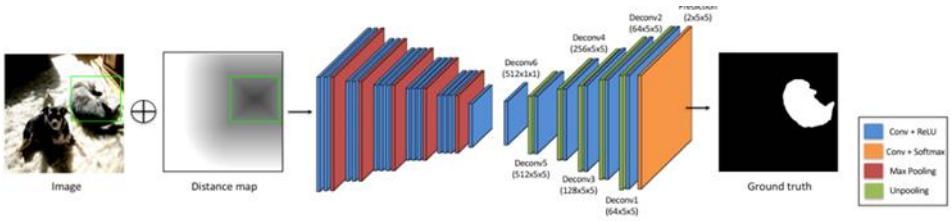


Figure 3.23. The steps of the Deep GrabCut Algorithm (Xu, N. et al., 2017).

The powerful point of DeepGrabCut is that the algorithm generates many boxes for each object. After that, the sampling process is done over these boxes.

$B^i = [x^i_{min}, x^i_{max}, y^i_{min}, y^i_{max}]$ where x, y represent the coordinate of the box B^i. The algorithm randomly samples N boxes. Equation 3.11 (Xu, N. et al., 2017) shows the formulas for the sampling process.

$$x^i_{min/max} = x^0_{min/max} + v * g^i_j * (x^0_{max} - x^0_{min})$$
$$y^i_{min/max} = y^0_{min/max} + v * g^i_j * (y^0_{max} - y^0_{min}) \quad (3.11)$$

where $g^i_j \sim N(0,1)$, $j \in \{1,2,3,4\}$ are standard Gaussian random variables, and v is a hyperparameter that controls the degree of rectangle variation (Xu, N. et al., 2017).

With this sampling strategy, a small training dataset can be augmented while keeping the model free from overfitting. The algorithm then transforms each of the sampled rectangles into a distance map. In particular, given a rectangle B in an image I, let us define the pixels on the edge of B as the set $S_e = \{p_i \mid p_i \text{ is on the edge of } B\}$ where p_i represents the location of pixel i. Similarly, the pixels inside B are defined as a set S_i and the pixels outside B as a set S_0. Then the algorithm creates a 2-D distance map D which has the same width and height as the image I. Equation 3.12 (Xu, N. et al., 2017) shows how to compute D at location p_i.

$$D(p_i) = \begin{Bmatrix} 128 - min_{\forall p_j \in S_e} |P_i - P_j| & if\ P_i \in S_i, \\ 128, & if\ P_i \in S_e, \\ 128 + min_{\forall p_j \in S_e} |P_i - P_j| & if\ P_i \in S_o, \end{Bmatrix} \quad (3.12)$$

In equation 3.12 $||$ refers to the Euclidean distance. (Xu, N. et al., 2017) used signed distance transformation to better capture the context information of rectangle inputs while the unsigned transformation gives inferior results. For the efficiency of

data storage, the values of *D* are in range between 0 to 255. Finally, the training pair is created by concatenating the RGB channels of *I* and the distance map *D*.

The model's duty is to take the concatenated pairs (image, rectangles) as input, then generate a mask as output. The model has two parts: The first part is the encoder which has the VGG-16 architecture, but (Xu, N. et al., 2017) just used the first 14 layers of the network. The encoder part's duty is to extract the features of the images and convert them into feature maps. On the opposite side, the decoder part consists of convolutional and unpooling layers, these layers generate the output in mask shape which carries the pixels' classes. Also, the authors used the dropout to avoid overfitting in the decoder part.

(Xu, N. et al., 2017) train the model over the PASCAL VOC 2012 dataset and COCO dataset. In each iteration, 4 bounding boxes are randomly sampled for each instance and make them a mini batch. Also, *v* (sampling parameter) is equal to 15e-2, and the Adam optimization function is used to train the model.

3.3. Proposed System by Using YOLO and GrabCut

In this study an object detection-based segmentation method is tested by using YOLO object detection followed by traditional GrabCut as the first approach, and the DeepGrabCut as the second approach. Both proposed approaches have the following two steps:

1. *The object detection:* To do this step two of YOLO versions were trained and tested (YOLOv3 and Tiny-YOLOv3) but with some modifications: The YOLOv3 in its pure case has 1000 classes and trained on ImageNet dataset. But in this study, we used two datasets for testing, those datasets have different number of classes, where the PASCAL VOC dataset has 20 classes, and the COCO 2014 contains 80 classes. So, we modified the number of classes in YOLO versions.

2. *Object Selection:* Two methods that are: the traditional GrabCut and the DeepGrabCut were applied and tested. For the GrabCut, we used the GrabCut with

a rectangle (object's box). For the DeepGrabCut a lot of modifications and training are needed before using it. DeepGrabCut in its pure case uses the VGG 16 architecture as an encoder, but in this study, we used the ResNet-101 which is one of the best feature's extractors. Also, we trained the model in different ways. In the original model, (Redmon, J. and Farhadi, 2018) used PASCAL VOC and COCO datasets separately for training and testing. But in this study besides the PASCAL VOC and the COCO datasets, a boundary semantic dataset (SBD) is used for training. In other words, the model in this study is firstly trained with the PASCAL VOC and the SBD datasets at the same time and stored for testing over the PASCAL VOC dataset. Then again trained over the COCO and the SBD datasets at the same time. This training helps the network to segment the overlapped objects correctly without merging the two objects in one object.

So, the workflow of the proposed method consists of object detection, object selection, and concatenation of the results to form the output in the mask shape. Figure 2.24 and Figure 2.25 show the used two approaches.

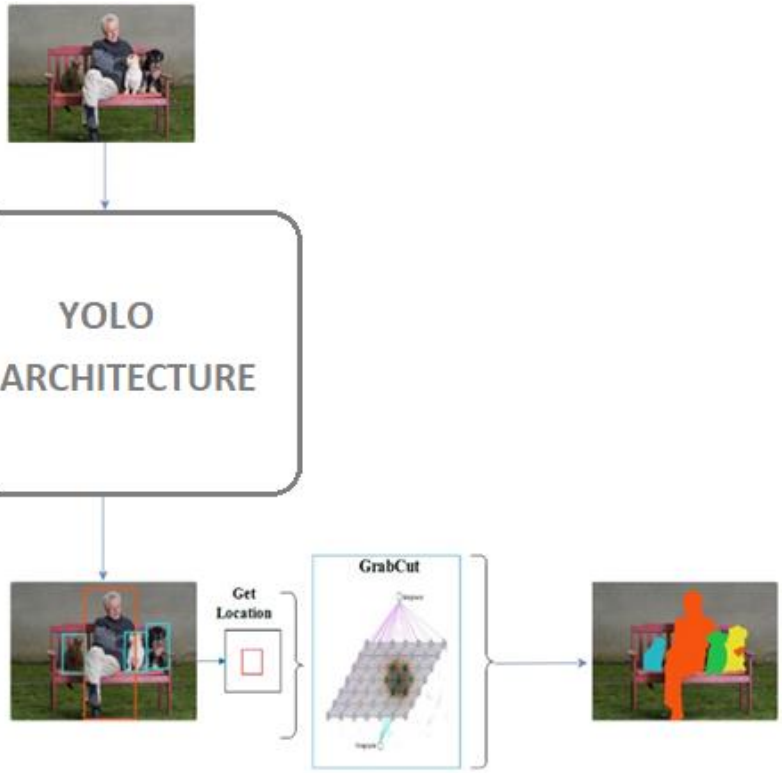


Figure 3.24. The first approach using the traditional GrabCut method.

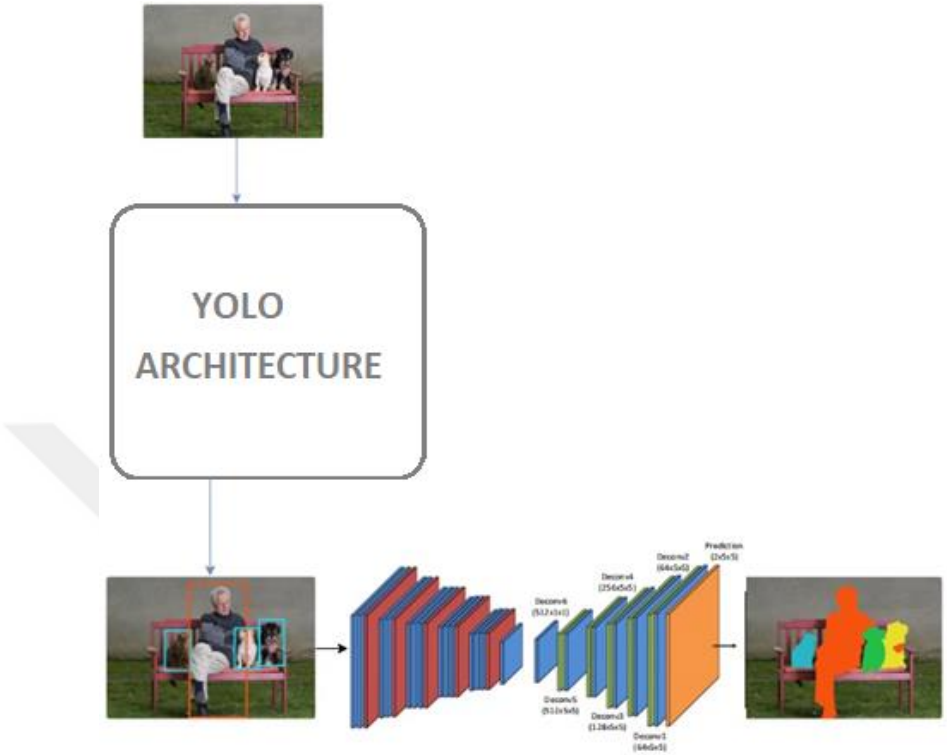


Figure 3.25 The second approach using the DeepGrabCut method.

3.4. Training and Tuning

3.4.1. Preparing the datasets:

For training the detection models, the provided images' annotations are converted into a text file (train.txt, val.txt, test.txt) each line in these text files has the following form: image_path, $object_0$_rectangle_info, $object_1$_rectangle_info $object_i$_rectangle_info. This process is done for both of the PASCAL VOC and COCO datasets separately. Figure 3.26 shows an example of the annotation of the PASCAL VOC 2012 dataset.

```
VOCdevkit/VOC2012/JPEGImages/2008_000002.jpg 34,11,448,293,19
VOCdevkit/VOC2012/JPEGImages/2008_000003.jpg 46,11,500,333,18 62,190,83,243,14
VOCdevkit/VOC2012/JPEGImages/2008_000007.jpg 1,230,428,293,3
VOCdevkit/VOC2012/JPEGImages/2008_000009.jpg 217,161,294,221,9 465,167,500,218,9
VOCdevkit/VOC2012/JPEGImages/2008_000016.jpg 91,15,392,353,19
VOCdevkit/VOC2012/JPEGImages/2008_000021.jpg 14,148,475,288,0
VOCdevkit/VOC2012/JPEGImages/2008_000026.jpg 122,7,372,375,14 211,147,325,255,11
VOCdevkit/VOC2012/JPEGImages/2008_000027.jpg 9,32,500,375,6
VOCdevkit/VOC2012/JPEGImages/2008_000032.jpg 6,118,489,274,5 336,173,364,224,14
VOCdevkit/VOC2012/JPEGImages/2008_000034.jpg 6,234,45,362,4 1,156,103,336,14 36,111,198,416,14 91,42,338,500,14
```

Figure 3.26 Train.txt file of PASCAL VOC 2012 dataset.

For the PASCAL VOC, the ground truth table for every image is stored in an XML file separately, and it takes the image name. So, by taking each image name and comparing the name with the XML file names we find the related XML file, then we save the image info in one line into the text file. Figure 3.27 shows a VOC XML file.

```
<annotation>
        <folder>VOC2012</folder>
        <filename>2007_000033.jpg</filename>
        <source>
                <database>The VOC2007 Database</database>
                <annotation>PASCAL VOC2007</annotation>
                <image>flickr</image>
        </source>
        <size>
                <width>500</width>
                <height>366</height>
                <depth>3</depth>
        </size>
        <segmented>1</segmented>
        <object>
                <name>aeroplane</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>9</xmin>
                        <ymin>107</ymin>
                        <xmax>499</xmax>
                        <ymax>263</ymax>
                </bndbox>
        </object>
        <object>
                <name>aeroplane</name>
                <pose>Left</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>421</xmin>
                        <ymin>200</ymin>
                        <xmax>482</xmax>
                        <ymax>226</ymax>
                </bndbox>
        </object>
        <object>
                <name>aeroplane</name>
                <pose>Left</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>325</xmin>
                        <ymin>188</ymin>
                        <xmax>411</xmax>
                        <ymax>223</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 3.27. A VOC xml file, image name, segmentation information and the image's objects info.

For the COCO dataset the images annotations are stored in a json file, also one file is used for every group of datasets (a file for validation, a file for training and a file for testing). Again, the json file is converted into a form that is used in YOLO architectures by taking the name of the image_path followed with the objects' rectangles info.

In the PASCAL VOC dataset, the ground truth table for segmentation is provided by the colored images (RGB) stored in the segmentation class folder. Each image in the truth table has the name of the original image name. Also, to color the results, the PASCAL VOC dataset provides a color. Text file which contains color of each class. So, the output of the object selection algorithm will be an image with the 20 colors, and the black color (0,0,0) represents the background. But the truth table for the COCO dataset is not that easy to get as PASCAL VOC, the COCO dataset saves everything in json files and provides a python script which has a function (COCO API) to generate the ground truth table in image shape. The function needs to select which classes are going to be used. After that, the images that contain the objective classes are listed in an array which is processed to get every image mask. We stored these masks with the name of the original image. Also, the images are in gray scale and the background takes the 0 value. The SBD dataset is used for just training the DeepGrabCut network beside the VOC PASCAL and COCO Datasets. The usage of this boundary segmentation dataset helps the network to segment the overlapped objects correctly. The SBD is a subset of the PASCAL VOC dataset, so it doesn't contain images, also the images' color distribution function is the same (same images). So, it just has the annotation for the boundary segmentation task.

3.4.2. Training

In this study, there are two trainable parts, the detection part, and the object selection part. For the detection part, we have two networks, the YOLOv3 and the Tiny-YOLOv3. However, in the object selection part, we have just one network, the DeepGrabCut (traditional GrabCut doesn't need training).

The detection part is trained as follows:

First, the YOLOv3 network is built from scratch, which is a Darknet-53 network, after that the YOLO loss function is coded. The initial weights have been taken from the official YOLO website, and converted to Keras weights, because the

original weights are fitted by the Darknet platform. Then, the used datasets are prepared by generating the annotations to get the (train.txt, val.txt). After that, the training task is started as follows:

YOLOv3 over the PASCAL VOC 2012 training, the PASCAL VOC 2012 contains 5717 images for training. We used this training set for both the training and validation by taking 90% of the images for training and 10% for validation by setting val-split equals 0.1. Table 3.1 shows the used hyperparameters (Redmon et al., 2016) in the training process. YOLOv3 was trained through 100 epochs and the network weights were saved in every 10 epochs.

Table 3.1 The hyperparameters for training YOLOv3 over PASCAL VOC 2012

Parameter	Batch size	val-split	momentum	decay	learning rate
Value	64	0.1	0.9	5e-4	3e-4

YOLOv3 over the MS COCO 2014 training, the MS COCO 2014 contains 23644 images as training, we used this training set for both the training and validation by taking 90% of the images for training and 10% for validation. Table 3.2 shows the used hyperparameters (Redmon et al., 2016) in the training operation. Yolov3 was trained through 50 epochs and the network weights were saved in every 10 epochs.

Table 3.2.The hyperparameters for training YOLOv3 over MS COCO 2014.

Parameter	Batch size	val-split	momentum	decay	learning rate
Value	64	0.1	0.9	5e-4	3e-4

Tiny-YOLOv3 over the PASCAL VOC 2012 training, the PASCAL VOC 2012 contains 5717 images for training. We used this training set for both the training and validation by taking 90% of the images for training and 10% for validation. Table 3.3 shows the used hyperparameters in the training process. Tiny-

YOLOv3 was trained through 50 epochs and the network weights were saved every 10 epochs.

Table 3.3. The hyperparameters for training Tiny-YOLOv3 over PASCAL VOC 2012

Parameter	Batch size	val-split	momentum	decay	learning rate
Value	128	0.1	0.9	5e-4	1e-4

Tiny-YOLOv3 over the COCO 2014 dataset training, the MS COCO 2014 contains 23644 images as training, we used this training set for both the training and validation by taking 90% of the images for training and 10% for validation. Table 3.4 shows the used hyperparameters in the training process. Tiny-YOLOv3 was trained through 5 epochs because the used dataset has a huge number of samples and the network weights were saved in every 1 epoch. Test results showed that the weights saved at 1 epoch were the most successful at detecting the location of PASCAL VOC 2012 dataset images.

Table 3.4. The hyper parameters for training Tiny-YOLOv3 over COCO 2014

Parameter	Batch size	val-split	momentum	decay	learning rate
Value	128	0.1	0.9	5e-4	3e-4

For the object selection part (segmentation part) the ResNet-101 network is built from scratch, then a classification architecture using ResNet weights are used as initial weights.

DeepGrabCut over the PASCAL VOC 2012 with SBD datasets training, Table 3.5 shows the used hyperparameters (Xu, N. et al., 2017) in the training process. DeepGrabCut was trained through 200 epochs and the network weights were saved in every 10 epochs. In this training, the batch size is very small this leads

to low learning ability (because of the limitation of the used GPU). Also, the mix between the two datasets (PASCAL VOC and SBD) make the training more accurate while the PASCAL VOC dataset deals with the semantic segmentation and the SBD dataset deals with boundary segmentation. Also, for COCO dataset, the parameters are used for training the DeepGrabCut network.

Table 3.5. The hyperparameters for training DeepGrabCut over PASCAL VOC 2012

Parameter	Batch size	val-split	momentum	decay	learning rate
Value	2	0.1	0.9	5e-4	1e-4

The whole implementation and computations were performed on a PC with Intel Core i7-7700HQ, 16 GB RAM, NVIDIA GTX 1060 GPU and Ubuntu 18.04 operating system. Python language, OpenCv image processing framework, Tensorflow-gpu, Keras-gpu, Pytorch frameworks were used in the development of the system. Figure 3.28 shows the PC's GPU specifications.

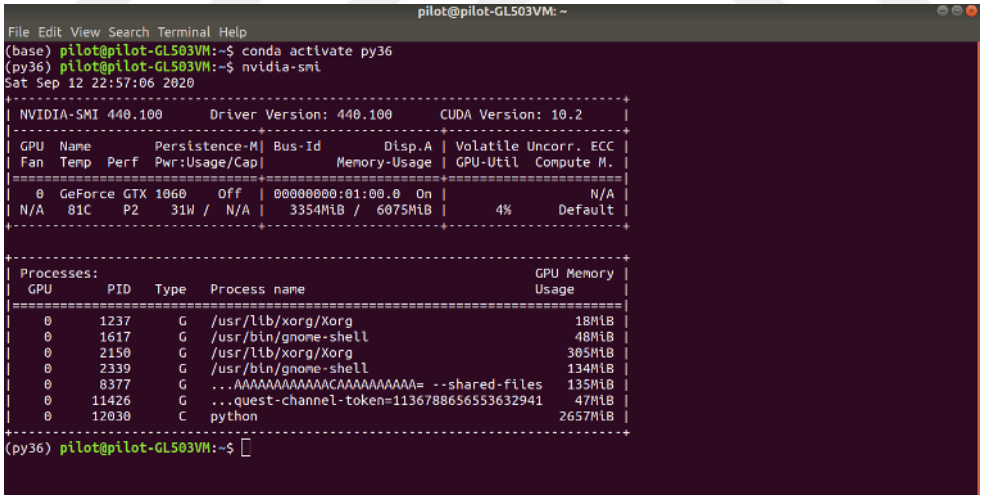


Figure 3.28. GPU specifications of the used PC: GPU: GeForce GTX 1060 Nvidia card, with driver version 440.10 and CUDA 10.2.

4. RESULTS AND DISCUSSIONS

The segmentation task is a classification task at pixel-level, so the classification metrics can be used to measure the performance of the used methods. Figure 4.1 shows the confusion matrix table that is used in performance measurement.

		Predicted label	
		Label 0	Label 1
Actual label	Label 0	TP (true positive)	FP (false positive)
	Label 1	FN (false negative)	TN (true negative)

Figure 4.1. Confusion matrix. (Source: https://medium.com/@yanfengliux/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef)

In Figure 4.1, TP represents the true positive, which means number of samples that are correctly predicted as the positive class. FP represents the false positive, that is equal to the number negative samples predicted as the positive class. FN is the false negative, which denotes the number positive samples that are predicted as the negative class. TN represents the true negative, this means number of samples correctly predicted as the negative class.

We used three traditional metrics to evaluate the segmentation part performance: (i) *Precision* which measures how accurate is the predictions (Equation 4.1. (a)), (ii) *Recall* which measures how good is the prediction of all the positives (Equation 4.1. (b)) and (iii) mIoU (mean Intersection over Union) is the mean IoU of the image which is calculated by taking the IoU of each class and averaging them (Equation 4.1. (c)) (Source: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173).

The PASCAL VOC dataset uses the mIoU metric for evaluating the performance, but for the COCO dataset, the used metric is the mean Average Precision (mAP) which computes the average precision value for recall value from 0 to 1. So, to calculate mAP, the area under the graph of precision to recall is calculated, but to make it possible, we smoothed out the zigzag pattern first as shown in Figure 4.2.

$$pre = \frac{TP}{TP+FP} \quad (a)$$
$$rec = \frac{TP}{TP+FN} \quad (b) \qquad (4.1)$$
$$IoU = \frac{AREAOFOVERLAP}{AREAOFUNION} \quad (c)$$
$$mAP = \int_0^1 p(r)dr \quad (d)$$
$$mAP = \frac{1}{n} \sum_{r\in\{0.0,\ \ldots\ldots,1.0\}} AP_r \quad (e)$$

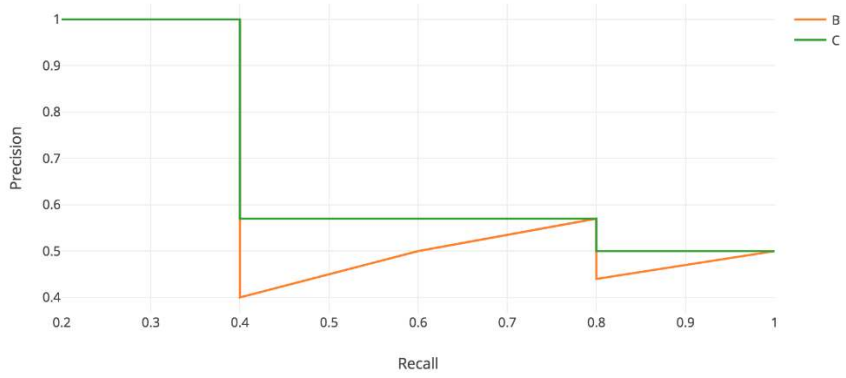


Figure 4.2. An example of Precision to Recall graph. B is the original one, and C is smoothed graph. (https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173)

Figure 4.2 shows an example of an AP graph after smoothing. The green line is the smoothed version of the original function which is drawn by the orange line in

the figure. Equation 4.1. (d) and (e) show the formula of mAP for the original, and the smoothed cases, respectively.

4.1. Experimental Results of the Object Detection Part

The approaches introduced in this study were evaluated in three stages. In the first stage, the object detection performance of the re-trained YOLOv3 and Tiny-YOLOv3 over both datasets PASCAL VOC 2012 and the COCO 2014 were evaluated. In the second stage the performance of the object selection methods GrabCut and the retrained DeepGrabCut were evaluated by taking the bounding boxes that are available for the annotation of the used dataset. The last stage is the evaluation of the performance of putting the object detection and the object selection parts together to make the image segmentation. In all the three stages we used the mean of Intersection over Union (mIoU) metric and the mAP metric for performance evaluation.

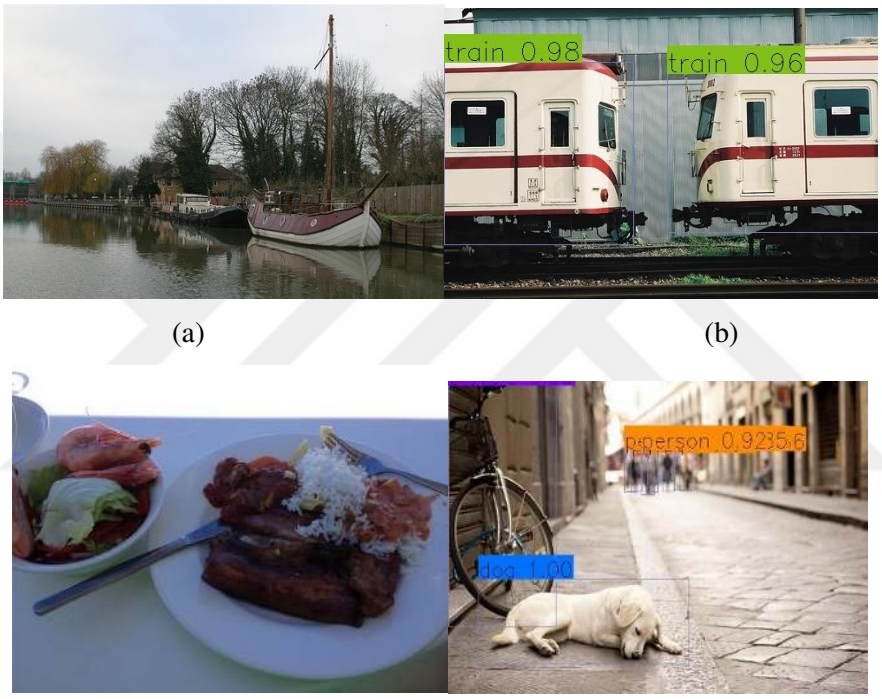
4.1.1. Performance of YOLOv3

For the PASCAL VOC dataset, the re-trained YOLOv3 achieved 50.4 mAP. The model was unable to make any predictions on only 279 images out of 2913 (9.5% of the images cannot be detected). Also, the model achieved 80% mIoU. But for the COCO 2014, the re-trained YOLOv3 performance was 55.3 mAP and it was unable to detect only 1623 images out of 32466 (4.8 % of the images cannot be detected). The mIoU rate of the model was 84%. Figure 4.3 shows examples from both datasets for detected and undetected images. Table 4.1 presents the detection performance of the model on the two datasets. Also, YOLOv3 can process 8.5 frames per second on the used machine. According to these results we can say that YOLOv3 has slightly better performance for COCO 2014 dataset.

Table 4.1. The result of the YOLOv3 detection architecture.

Datasets	mAP	mIoU	Total Undetectable	FPS*
VOC 2012	50.4	80%	279 images out of 2.913	8.5
COCO 2014	55.3	84%	1623 images out of 32466	8.5

* FPS is the number of frames per second that can be processed on the used machine.



(a) (b)

(c) (d)

Figure 4.3. Results of object detection by YOLOv3. (b, d) are successful detections on the PASCAL VOC 2012, and the MS COCO 2014, respectively. (a, c) show unsuccessful detections on PASCAL VOC 2012, and the COCO 2014.

4.1.2. Performance of Tiny-YOLOv3

We repeated the previous experiment by using the Tiny-YOLOv3 architecture. For the PASCAL VOC dataset, the re-trained Tiny-YOLOv3 achieved 30.7 mAP. The model was unable to make any predictions on 955 images out of

2913 (32.8% of the images cannot be detected). Also, the model achieved 50% mIoU. But for the COCO 2014, the re-trained Tiny-YOLOv3 performance was 33.1 mAP, and it was unable to detect 10,532 images out of 32,466 (32.4 % of the images cannot be detected). The mIoU rate of the model was 60% for the COCO 2014 dataset. Figure 4.4 shows examples from both datasets for detected and undetected images. Table 4.2 represents the detection performance of the model on the two datasets. Also, Tiny-YOLOv3 can process 23 frames per second on the used machine.

According to Table 4.2, Tiny-YOLOv3 also has slightly better performance on the COCO 2014 dataset as that of YOLOv3. When the performance of Tiny-YOLOv3 and YOLOv3 are compared (see Table 4.1, and Table 4.2), YOLOv3 has much better mAP and mIoU. However, the number of frames processed per second is much higher in Tiny-YOLOv3, as we have expected.

Table 4.2. The results of the Tiny-YOLOv3.

Datasets	mAP	mIoU	Total Undetectable	FPS*
VOC 2012	30.7	50%	955 images out of 2.913	23
COCO 2014	33.1	60%	10532 images out of 32466	23

* FPS is the number of frames per second that can be processed on the used machine.

(a) (b)

(c) (d)

Figure 4.4. Results of object detection by Tiny-YOLOv3. (b, d) are successful detections on the PASCAL VOC 2012 and the COCO 2014 respectively. (a, c) show unsuccessful detections on PASCAL VOC 2012 and the MS COCO 2014.

4.2. Experimental results of the object selection part

To evaluate the performance of the object selection part, we used the two metrics: mean Average Precision (mAP) and mIoU that were used in the previous experiments. The mIoU is used to compare the results with the studies using PASCAL VOC dataset, and the AP metric is used to compare the results with the studies using the COCO dataset. The two object selection methods (traditional

GrabCut and DeepGrapCut) are applied over the bounding boxes that were obtained from the dataset's annotations.

4.2.1. Results of traditional GrabCut

For the traditional GrabCut, the experimental results are shown in Table 4.3, where the result having * symbol was taken from (Xu, N. et al., 2017), but the other results were calculated in this thesis study. According to Table 4.3, the results on COCO 2014 dataset is slightly better.

Table 4.3. The results of GrabCut.

Dataset	mAP	mIoU
VOC 2012	37.20*	66.24
COCO 2014	39.66	65.10

4.2.2. Results of DeepGrabCut

For the DeepGrabCut, the experimental results are shown in Table 4.4, where the result having * symbol was taken from (Xu, N. et al., 2017), but the other results were calculated in this thesis study. According to Table 4.4, we can see that the results on COCO dataset is also slightly better. When the results on Table 4.3 and Table 4.4 are compared, we can easily see that DeepGrabCut is much more successful than GrapCut.

Table 4.4. The results of DeepGrabCut.

Dataset	mAP	mIoU
VOC 2012	67.3*	85.40
COCO 2014	68.23	86.54

4.3. Experimental results of the object selection and object detection parts together

The performance of using the object detection followed with object selection is evaluated by using the two metrics: mean Average Precision (mAP) and mIoU as in the previous experiments given in this section. The mIoU is used to compare the results with the studies using PASCAL VOC dataset, and the mAP metric is used to compare the results with the studies using the COCO dataset. The two metrics are calculated for the two cases: i) the case of taking all the images, and ii) the case of taking just the images with detected objects.

4.3.1. Results of YOLO and traditional GrabCut

For YOLO and the traditional GrabCut approach, the experimental results of taking just the images with the detected objects are shown in Table 4.5, and we can see that the difference between the two detection architectures YOLOv3 and Tiny-YOLOv3 is not huge. For the PASCAL VOC dataset, the YOLOv3 couldn't detect any object for 279 out of 2913 images, and the Tiny-YOLOv3 couldn't detect any object for 955 out of 2913 images. Also, the results on the COCO dataset for both the Tiny-YOLOv3 and full YOLOv3 are slightly better than that of VOC dataset, because the YOLO algorithms trained on COCO was more accurate. As in the VOC dataset, difference between YOLOv3 and Tiny-YOLOv3 is also very small for COCO dataset.

Table 4.5. The result of YOLO and GrabCut together without the images that are not detected.

Segmentation Approach	Datasets	mAP	mIoU
YOLOv3+GrabCut	VOC 2012	30.39	47.40
Tiny-YOLOv3+GrabCut	VOC 2012	30.81	47.81
YOLOv3+GrabCut	COCO 2014	32.77	50.13
Tiny-YOLOv3+GrabCut	COCO 2014	31.23	48.20

But when all the images (both the images with no detection and the detected images) were taken into account to calculate the mAP and mIoU, the accuracy decreased as shown in Table 4.6. For Tiny-YOLOv3, mAP reduced from 30.81 to 20.2 for VOC 2012 dataset. However, the change in the results of YOLOv3 for both cases were not much high (about 5%), especially for the COCO dataset. Table 4.6 shows the results of the two object detection architectures for all the tested images.

Table 4.6. The results of YOLO and GrabCut together over all the tested images.

Segmentation Approach	Datasets	mAP	mIoU
YOLOv3+GrabCut	VOC 2012	24.56	40.04
Tiny-YOLOv3+GrabCut	VOC 2012	20.2	30.29
YOLOv3+GrabCut	COCO 2014	27.03	45.43
Tiny-YOLOv3+GrabCut	COCO 2014	23.60	32.3

4.3.2. Results of YOLO and DeepGrabCut

For the YOLO and DeepGrabCut approach, the results of taking just the detected images are shown in Table 4.7, and we can see that the difference between the two detection architectures YOLOv3 and Tiny-YOLOv3 is not large. Also, the approach with Tiny-YOLOv3 is slightly better than the YOLOv3 for VOC 2012 dataset. When Table 4.7 and Table 4.5 are compared, there is no huge difference between GrabCut and DeepGrabCut, both methods have similar performances with the two YOLO architecture.

Table 4.8 shows the results of the two detection architectures for all the tested images. But when the images with no detection are taken into evaluation, the accuracy decreased as it is expected. The mAP of the Tiny-YOLOv3 reduced to 22.51 for VOC 2012. This huge difference comes from the Tiny-YOLOv3's low ability for finding bounding boxes. The YOLOv3 with DeepGrabCut has slightly

higher performance on VOC 2012 dataset than that of COCO 2014 dataset with respect to mIoU measure.

Table 4.7. The results of YOLO and DeepGrabCut together without the images that are not detected.

Segmentation Approach	Datasets	mAP	mIoU
YOLOv3+DGC	VOC 2012	30.78	47.73
Tiny-YOLOv3+DGC	VOC 2012	30.83	48.27
YOLOv3+DGC	COCO 2014	31.97	46.46
Tiny-YOLOv3+DGC	COCO 2014	30.20	47.59

Table 4.8. The results of YOLO and DeepGrabCut together for all the tested images.

Segmentation Approach	Datasets	mAP	mIoU
YOLOv3+DeepGrabCut	VOC 2012	25.49	42.88
Tiny-YOLOv3+DeepGrabCut	VOC 2012	22.51	32.22
YOLOv3+DeepGrabCut	COCO 2014	29.97	40.53
Tiny-YOLOv3+DeepGrabCut	COCO 2014	23.78	33.53

4.4. Comparison of the Used Methods

The bounding boxes for the object selection methods are found by YOLOv3 and Tiny-YOLOv3. From the results, it is observed that YOLOv3 is better than Tiny-YOLOv3. Also, the number of images with no detected object is less for YOLOv3 than that of Tiny-YOLOv3. But, Tiny-YOLOv3 is about 300% faster than YOLOv3. This low accuracy of Tiny-YOLOv3 and the speed of processing come from the fact that Tiny-YOLOv3 uses a shallow net compared to YOLOv3. But as the detection

accuracy is very important for the next stage (object selection) the use of YOLOv3 gives better results whatever the next stage uses (either traditional GrabCut or DeepGrabCut).

For the methods that are used in the object selection stage, the results show that the DeepGrabCut method is obviously better. Also, the DeepGrapCut has the ability to avoid merging two objects as one object. But traditional GrabCut doesn't need training and this makes the segmentation part general for any detection algorithm over any dataset.

4.5. Comparison of Results with Previous Studies Using the Same Datasets

In this study, PASCAL VOC and COCO datasets are used for testing the object detection based semantic segmentation, and our results are compared with the other studies that use the same dataset. Also, some studies used just one of the two datasets, so we applied the other methods and calculated the needed metrics to make the comparison more meaningful.

The study of (J. Long et al., 2015) used the PASCAL VOC 2012 dataset to evaluate the FCN architecture which achieved 62.2% mIoU score after training the models on the 2012 ImageNet dataset. In the same year, W. Liu et al. used the same dataset for training and evaluation of the ParseNet which reached to 69.8% mIoU. In 2016, FPN architecture (T.-Y. Lin et al., 2015) achieved 48.1% Average Recall (AR) score on the 2016 COCO segmentation challenge. Also, (H. Zhao et al., 2015) in the same year scored 85% mIoU on the 2012 PASCAL VOC challenge with their network PSPNet. The DeepLab3+ scored 89.0% mIoU on the 2012 PASCAL VOC challenge. The M-RCNN, which is an object detection-based segmentation, scored 37.1% AP on the COCO dataset.

Table 4.9 shows the results of the previous studies as well as results that we have obtained in this thesis to make a complete comparison. So, according to the tested methods in this study and the previous studies, the best image segmentation until writing this thesis is the DeepLab3+ which is tested on the PASCAL VOC

dataset and gives an 89.0% mIoU. The lowest performance among the previous studies belongs to M-RCNN method which has 53.2% mIoU. The highest mIoU for our tested methods is 48.27% mIoU which belongs to Tiny-YOLOv3+DeepGrabCut over PASCAL VOC dataset. This result shows that using YOLO versions and GrabCut or DeepGrabCut together has not satisfactory performance.

For the COCO dataset where the AP metrics are used to compare the results, among the compared previous studies, the best result is equal to 63.13 AP value which belongs to DeepLab3+. Among the compared previous studies on the COCO dataset, the worst result that is equal to 37.10 AP is computed by M-RCNN method. Our methods have highest 32.77 AP with YOLOv3+GrabCut and this value is less than the object detection-based image segmentation algorithm (M-RCNN).

Therefore, we evaluated the performance of the GrabCut and DeepGrabCut without using YOLO, so that the bounding boxes were taken from the annotation of the datasets. We observed that our results are better, because the boxes are perfect. In that case, the DeepGrabCut, which is a deep learning based method, were trained over these bounding boxes. So, DeepGrabCut has very good performance as shown in Table 4.9. Using YOLO as object detection part (bounding boxes provider) has a significant amount of error, therefore it affects the semantic segmentation in negative way. According to Table 4.9, the DeepGrabCut has 83.40% mIoU on Pascal VOC 2012 dataset, which it is so good comparing to the other architectures. Also, the traditional GrabCut has 66.24% mIoU on Pascal VOC 2012 which is lower than that of the DeepGrabCut, but it is still good when compared to the other methods. However, as the DeepGrabCut and traditional GrabCut need the bounding boxes, they are still not the perfect solution to this problem.

Table 4.9. The results of studies using the PASCAL VOC 2012 and COCO 2014 datasets.

Model	PASCAL VOC (mIoU)	COCO (AP)
FCN	62.2%	41.58**
ParseNet	69.8%	46.28**
Conv & Deconv	72.5%	48.60**
PSPNet	85.4%	62.88**
DeepLab	79.7%	53.23**
DeepLabv3	86.9%	59.20**
DeepLabv3+	89.0%	63.13**
M-RCNN	53.20%**	37.10
YOLOv3+GC	47.40%**	32.77**
YOLOv3+DGC	47.73%**	31.97**
Tiny-YOLOv3+GC	47.81%**	31.23**
Tiny-YOLOv3+DGC	48.27%**	30.20**
YOLOv3+GC *	40.04%**	27.03**
YOLOv3+DGC *	42.88%**	29.97**
Tiny-YOLOv3+GC *	30.29%**	23.60**
Tiny-YOLOv3+DGC *	32.22%**	23.78**
GrabCut(only)	66.24%**	39.66**
DeepGrabCut(only)	83.40%**	68.23**

* The images with no object detected are taken in the evaluation to compute mIoU and AP values.

** This result has been calculated in this study, but the others have been taken from the original papers of the methods.

5. CONCLUSIONS

In this study, deep learning-based object selection methods, deep learning-based object detection, and traditional object selection methods are used, and a comparison is made with traditional object selection and deep learning-based object selection methods. While doing this comparison two object detectors are used: one is a deep network, and the other is a shallow network. The detection and the selection models are trained and tested over two well-known datasets.

When we evaluated our results for object detection, it can be concluded that:

1. Tiny architecture is less accurate but faster. With this architecture the images with no object detected are about 32.8 % of the tested images.
2. The full architecture gives more accurate object detection, but it is slower with respect to the tiny architecture. With the full architecture, the images with no object detected are about 9.5 % of the tested images.

When we evaluated our results for the object selection part, it can be concluded that:

1. Deep learning-based architecture needs to be trained over the used dataset, but the performance (the speed and the mIoU) is better than the traditional one.
2. The traditional architecture is slower, but it doesn't need training so it can work over any dataset.

According to the experimental results, we have a cumulative error comes from the YOLO architecture and GrabCut or DeepGrabCut. YOLO architecture has 80% accuracy; therefore, it reduces accuracy of GrabCut and DeepGrabCut. This error makes the overall result not the best. When the perfect bounding boxes are used

in GrabCut and DeepGrabCut their accuracy become much higher. Therefore, the quality of the used bounding boxes affects a lot the general result, so using a better object detection method will make the segmentation result better. But according to the results in this study, it is found that deep learning-based object selection has higher segmentation successes with the use of YOLOv3 compared to the traditional segmentation method. Therefore, DeepGrabCut with YOLOv3 can be seen as a choice for image segmentation tasks. But using the traditional GrabCut will give a general object segmentation, so it can be an alternative segmentation method for deep learning-based segmentation approaches. In addition, the tested methods can be used in different segmentation problems. But the existing image segmentation algorithms still give better performance while our methods can be used for image segmentation and object detection at the same time. And with every improvement of the object detection algorithms, our method will get a benefit from that improvement. So, for the next step of this study, we will focus on improving the object selection part by changing the network architecture.

REFERENCES

Chen, L., Hermans, A., Papandreou, G., Schroff, F., Wang, P., & Adam, H. (2018). MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4013-4022. doi:10.1109/cvpr.2018.00422

Chen, L., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. Computer Vision – ECCV 2018 Lecture Notes in Computer Science, 833-851. doi:10.1007/978-3-030-01234-2_49

Chen, X., Girshick, R., He, K., & Dollár, P. (2019, August 27). TensorMask: A Foundation for Dense Object Segmentation. Retrieved December 22, 2020, from https://arxiv.org/abs/1903.12174

Dai, J., He, K., & Sun, J. (2016). Instance-Aware Semantic Segmentation via Multi-task Network Cascades. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3150-3158. doi:10.1109/cvpr.2016.343

Dhanachandra, N., Manglem, K., & Chanu, Y. J. (2015). Image Segmentation Using K -means Clustering Algorithm and Subtractive Clustering Algorithm. Procedia Computer Science, 54, 764-771. doi:10.1016/j.procs.2015.06.090

Forsyth, D., Ponce, J., Mukherjee, S., & Bhattacharjee, A. K. (2015). Computer vision: A modern approach. Uttar Pradesh, India: Pearson India Education Services Pvt.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 580-587. doi:10.1109/cvpr.2014.81

H. Shin et al., "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1285-1298, May 2016, doi: 10.1109/TMI.2016.2528162.Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11), 1222-1239. doi:10.1109/34.969114

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778. doi:10.1109/cvpr.2016.90

Huang, G., Liu, Z., Maaten, L. V., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4700-4708. doi:10.1109/cvpr.2017.243

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. International Journal of Computer Vision, 1(4), 321-331. doi:10.1007/bf00133570

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90. doi:10.1145/3065386

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi:10.1109/5.726791

Lin, T., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2117-2125. doi:10.1109/cvpr.2017.106

Liu, W., Rabinovich, A., & Berg, A. (2015, November 19). ParseNet: Looking Wider to See Better. Retrieved June 2, 2020, from https://arxiv.org/abs/1506.04579

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3431-3440. doi:10.1109/cvpr.2015.7298965

Minaee, S., & Wang, Y. (2019). An ADMM Approach to Masked Signal Decomposition Using Subspace Representation. IEEE Transactions on Image Processing, 28(7), 3192-3204. doi:10.1109/tip.2019.2894966

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017, April 17). Retrieved November 15, 2019, from https://deepai.org/publication/mobilenets-efficient-convolutional-neural-networks-for-mobile-vision-applications

Najman, L., & Schmitt, M. (1994). Watershed of a continuous function. Signal Processing, 38(1), 99-112. doi:10.1016/0165-1684(94)90059-0

Nock, R., & Nielsen, F. (2004). Statistical region merging. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(11), 1452-1458. doi:10.1109/tpami.2004.110

Noh, H., Hong, S., & Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. 2015 IEEE International Conference on Computer Vision (ICCV), 1520-1528. doi:10.1109/iccv.2015.178

Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1), 62-66. doi:10.1109/tsmc.1979.4310076

Plath, N., Toussaint, M., & Nakajima, S. (2009). Multi-class image segmentation using conditional random fields and global classification. Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09, 817-824. doi:10.1145/1553374.1553479

Redmon, J. (n.d.). Retrieved September 2, 2020, from https://pjreddie.com/darknet/

Redmon, J., & Farhadi, A. (2016, December 25). YOLO9000: Better, Faster, Stronger. Retrieved June 23, 2019, from https://arxiv.org/abs/1612.08242

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 27-30. doi:10.1109/cvpr.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149. doi:10.1109/tpami.2016.2577031

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149. doi:10.1109/tpami.2016.2577031

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 234-241. doi:10.1007/978-3-319-24574-4_28

Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut". ACM Transactions on Graphics, 23(3), 309-314. doi:10.1145/1015706.1015720

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211-252. doi:10.1007/s11263-015-0816-y

Simonyan, K., & Zisserman, A. (2015, April 10). Very Deep Convolutional Networks for Large-Scale Image Recognition. Retrieved December 22, 2020, from https://arxiv.org/abs/1409.1556

Starck, J., Elad, M., & Donoho, D. (2005). Image decomposition via the combination of sparse representations and a variational approach. IEEE Transactions on Image Processing, 14(10), 1570-1582. doi:10.1109/tip.2005.852206

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. 2015 IEEE Conference on

Computer Vision and Pattern Recognition (CVPR), 1-9. doi:10.1109/cvpr.2015.7298594

Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer.

Wang, D., & Aimoerfu. (2018). The Experimental Implementation of GrabCut for Hardcode Subtitle Extraction. 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS), 6-8. doi:10.1109/icis.2018.8466484

YOLOv3: An Incremental Improvement - arXiv. (n.d.). Retrieved December 22, 2020, from https://arxiv.org/pdf/1804.02767.pdf

Xu, N., Price, B., Cohen, S., Yang, J., & Huang, T. (2017). Deep GrabCut for Object Selection. Procedings of the British Machine Vision Conference 2017. doi:10.5244/c.31.182

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017). Pyramid Scene Parsing Network. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2881-2890. doi:10.1109/cvpr.2017.660

Zivkovic, Z. (2004). Improved adaptive Gaussian mixture model for background subtraction. Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., 26-26. doi:10.1109/icpr.2004.1333992

BIOGRAPHY

Ramazan Abdullah was born on June 4th, 1992 in Idlib. He has completed his elementary education at Ketyan Primary School, secondary and high education at Al-Bassel High School for Outstanding Students. He has completed his university education at the Department of Computer Engineering of Çukurova University in 2017.