مدينة زويل للعلوم والتكنولوچيا
Zewail City of Science and Technology

Convolutional Neural Networks for image segmentation

Year : 2018
Supervisor : Dr. Hatem Fayed
Team Members :

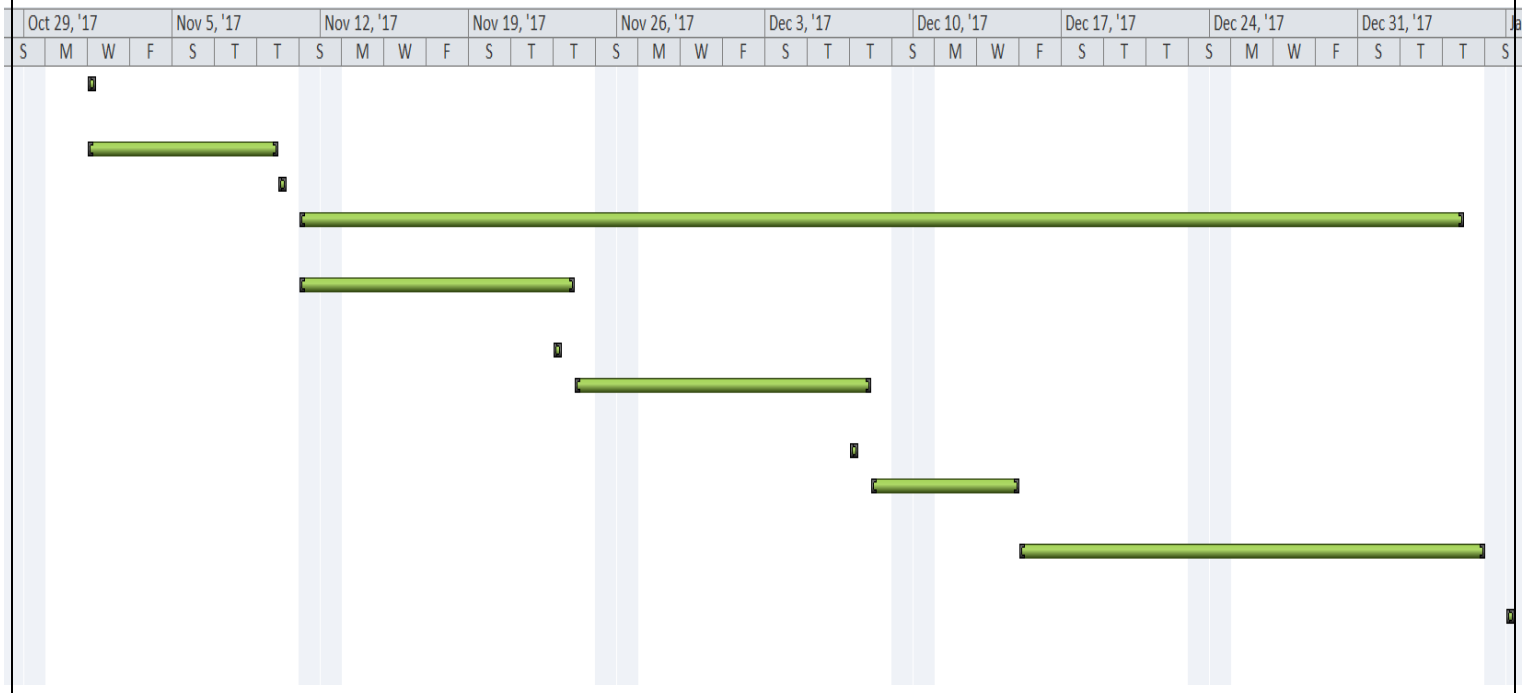
Name	ID
Amr Elgendy	**201304826**
Alhasan Abdellatif	**201305277**
Mahmoud Elsayed	**201304903**
Youssef Sherif Mansour	**201305127**

Table of Contents

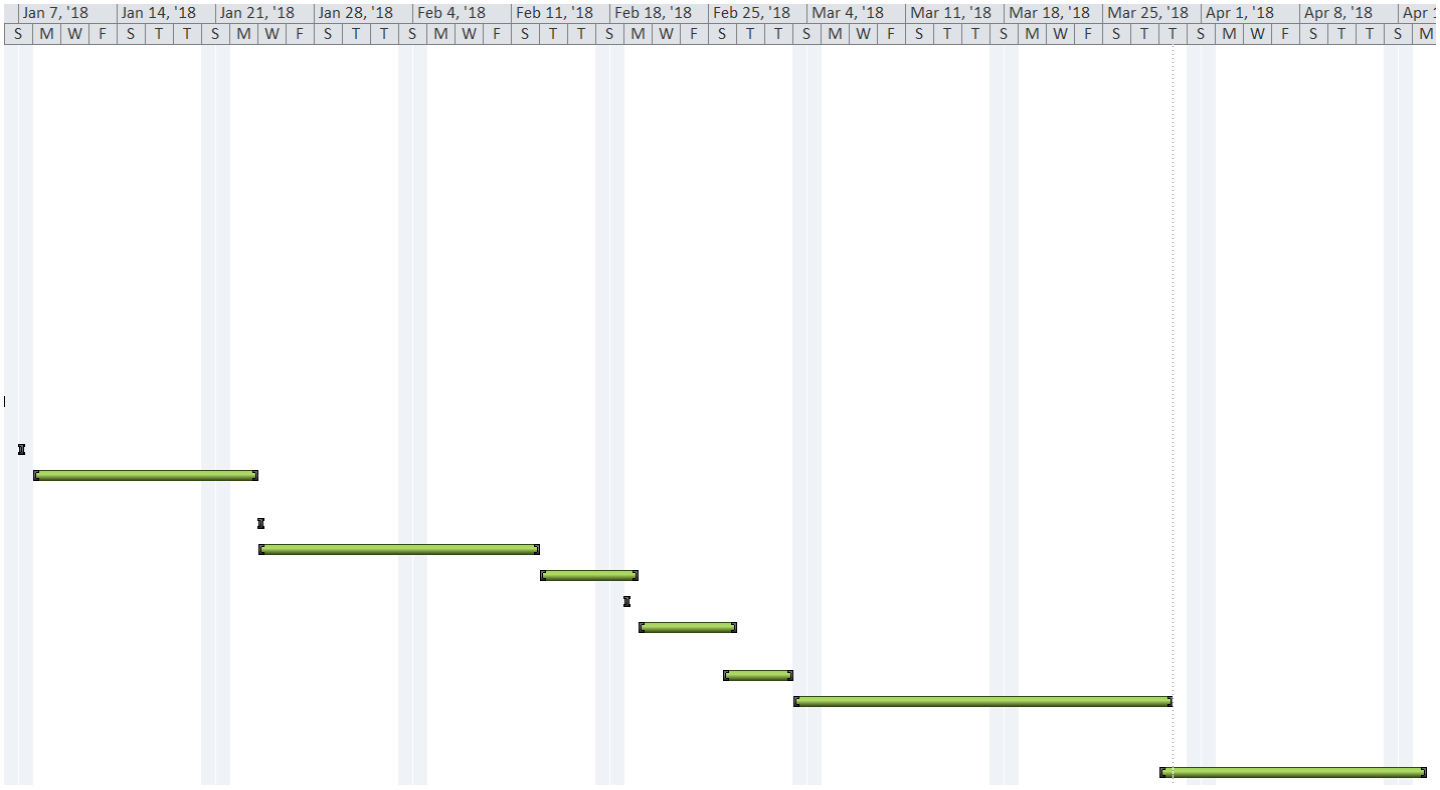
Abstract

In our modern age of technology, Computer vision techniques and pattern recognition, as a relatively fresh field of study, get refined by the second and new techniques surface more often than our abilities to keep up, so the transition from traditional Object detection algorithms like YOLO (**Y**ou **O**nly **L**ook **O**nce) to Semantic and Instance segmentation algorithms, that deal with classification at the pixel level, was inevitable. In this project, we attempt to explore some of those different techniques such as Mask RCNN and U-NET, compare their performance and accuracy by applying them exclusively on the Nucleus dataset provided by the Science bowl competition 2018, which deal with detecting Nuclei instances in experimental microscopic images. This dataset is collected from different experiments which makes it extremely varied. So, the key in this project is the development of a model that avoids overfitting and performs well on general datasets. For Mask RCNN we try different variants and try combining new ones.

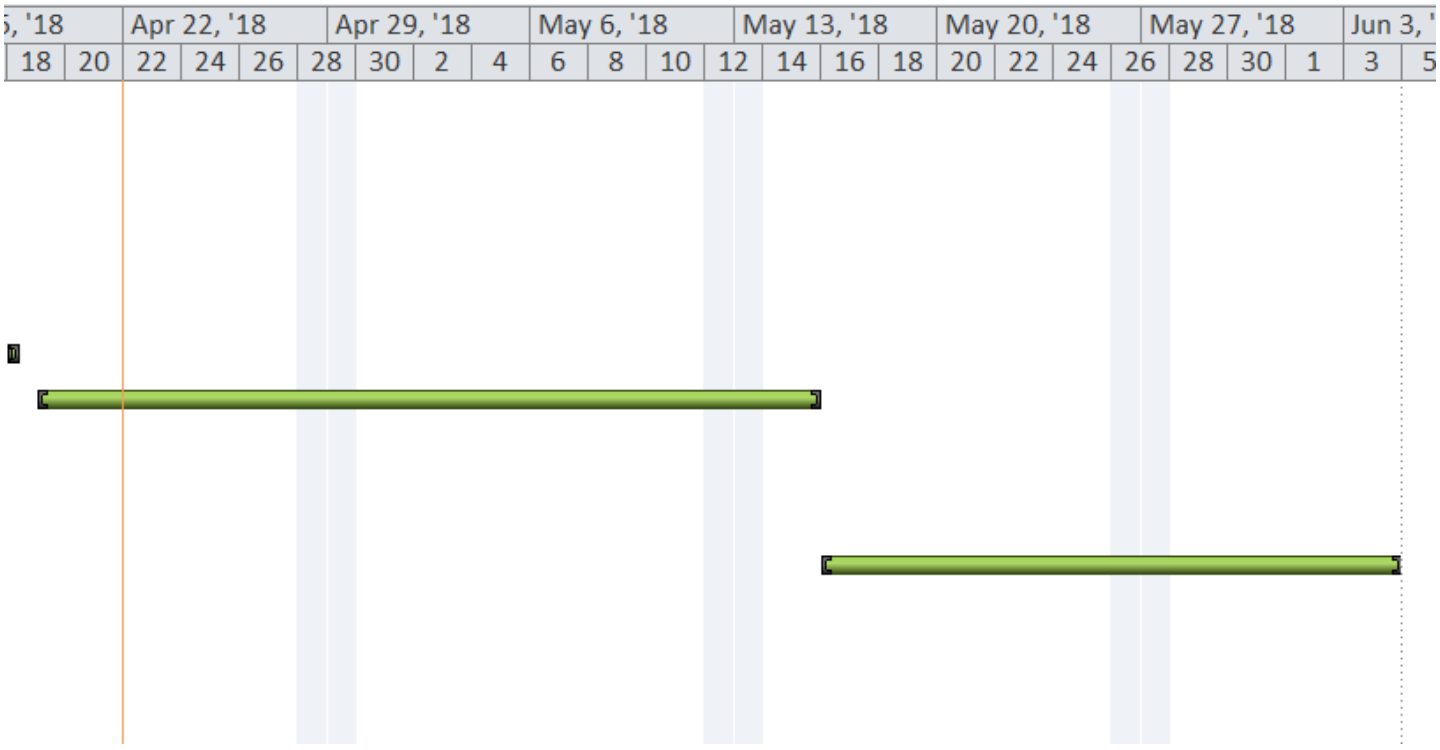
Gantt chart



Covering tasks for the first two months of the project timeline starting November

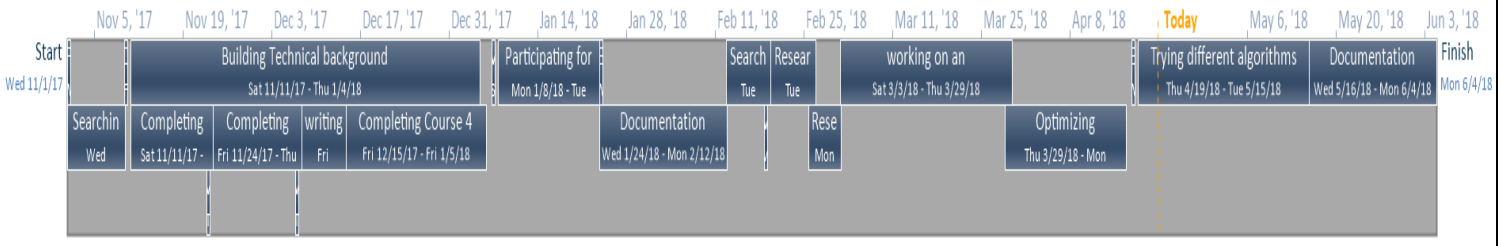


Covering tasks of the project timeline up until late April



Tasks from late April till the end of the proposed timeline

Task Name	Duration	Start	Finish
First Meeting and deciding the field of research	2 hrs	Wed 11/1/17	Wed 11/1/17
Searching for ideas	7 days	Wed 11/1/17	Thu 11/9/17
Confirmation Meeting	2 hrs	Fri 11/10/17	Fri 11/10/17
Building Technical background	2 mons	Sat 11/11/17	Thu 1/4/18
Completing Course 1 (deep learning spec.)	2 wks	Sat 11/11/17	Thu 11/23/17
Meeting	2 hrs	Thu 11/23/17	Thu 11/23/17
Completing Course 2 (deep learning spec.)	2 wks	Fri 11/24/17	Thu 12/7/17
Meeting	2 hrs	Thu 12/7/17	Thu 12/7/17
writing our own implementation of NN	1 wk	Fri 12/8/17	Thu 12/14/17
Completing Course 4 (deep learning spec.)	16 days	Fri 12/15/17	Fri 1/5/18
Meeting	2 hrs	Sun 1/7/18	Sun 1/7/18
Participating for Iceberg Competition	12 days	Mon 1/8/18	Tue 1/23/18
Evaluation Meeting	2 hrs	Wed 1/24/18	Wed 1/24/18
Documentation	14 days	Wed 1/24/18	Mon 2/12/18
Searching for ideas	1 wk	Tue 2/13/18	Mon 2/19/18
Meeting	2 hrs	Mon 2/19/18	Mon 2/19/18
Researching Science bowl Competition	1 wk	Tue 2/20/18	Mon 2/26/18
Researching Mask RCNN	1 wk	Mon 2/26/18	Fri 3/2/18
working on an implementation for Mask RCNN	1 mon	Sat 3/3/18	Thu 3/29/18
Optimizing algorithm and Preparing submissions for the competition	13 days	Thu 3/29/18	Mon 4/16/18
Evaluation Meeting	2 hrs	Wed 4/18/18	Wed 4/18/18
Trying different algorithms and comparing performances for a case study	19 days	Thu 4/19/18	Tue 5/15/18
Documentation	14 days	Wed 5/16/18	Mon 6/4/18



Full Timeline

Acknowledgments

First of all, we are grateful to Almighty GOD for making us complete this project and for all his blessings.

We would like to express our deepest gratitude to Professor Hatem Fayed, Director of applied mathematics department at Zewailcity, for supervising us and providing us with every support we needed throughout the project.

We would also like to thank Professor Andrew NG for his amazing deep learning specialization on Coursera that taught us a lot. As well as Professor Waleed Abdulla for his amazing opensource Matterport group that provided the base framework for our project

Finally, we are grateful to all our families, colleagues and friends who helped and supported us to complete this work.

List of Tables

List of Figures

I. Introduction

Image Segmentation, one of the many computer vision focuses, is the process of classifying and partitioning an image into a group of segments that cover regions of interest in that image to obtain a meaningful, easy-to-analysis output. As opposed to the traditional object detection, Segmentation works on a pixel level where each pixel is labeled to belong to a class. Segmentation is classified according to this labeling process into two types, either the pixel can be classified to belong to a certain class, which is known as semantic segmentation, or the pixel can be further classified to belong to instances of classes, which is known as Instance segmentation. This idea can be illustrated in figures 1.1a to c. This project will focus mainly on Instance segmentation.

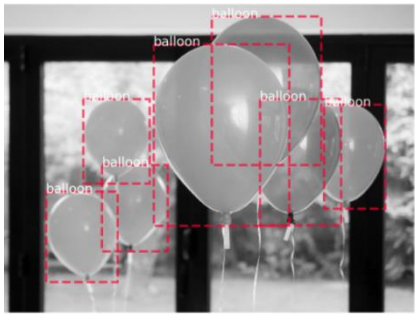


Figure 1.1a Traditional Object detection

Figure 1.1b Semantic segmentation

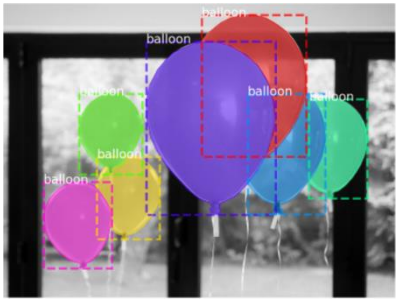


Figure 1.1c Instance segmentation

In the world of practicality, proposing solutions for a problem always involves a tradeoff, the one encountered in this project is one of speed versus accuracy. As much as accuracy is important, speed is also crucial in real-time applications such as navigation in robots or autonomous vehicles. Thus, the main goal of this project is to find a balance between speed and accuracy with more emphasis on the speed of detection. This problem is approached by trying different models and fine-tune their parameters for optimum results. 2 main models are selected, the modern highly-accurate yet considerably slow **Mask RCNN** and The Densely stacked yet fast **U-net**. Both models are to be fine-tuned and compared performance-wise by training both on the Nucleus dataset, a comparably small challenging dataset with high diversity introduced in 2018 Data Science Bowl hosted by Kaggle [10].

This report goes into the details of the previously described process. Chapter 2 reviews prior work relevant to problem stated here and the different models used in detail. In chapter 3, an analysis of the used dataset is discussed along with problems faced, proposed solutions and the screening process of those solutions. Lastly, Conclusions reached are presented in Chapter 4.

II. Literature Review

We begin by introducing a modern basic concept in deep learning which is FCN. It began to appear with the rise of the segmentation tasks in the computer vision community, and now, it is almost found in every image segmentation deep model architecture.

i. Fully Convolutional Networks for Semantic Segmentation

In the paper of Fully Convolutional Networks for Semantic Segmentation [11], they showed how a convolutional network, trained end-to-end and pixels-to-pixels, provides improvement in the accuracy of semantic segmentation compared to classical techniques that label each pixel with the class of its enclosing object or region. In addition, their created model does not use any post- or pre-processing complications such as super-pixels or proposals used by other models. Earlier models have used patch classification for semantic segmentation where each pixel is separately classified using a patch of images around it; that is because the networks usually have fully connected layers that require fixed sized images.

In contrast, Fully Convolutional Networks provide dense prediction without the use of fully connected layers allowing for working with images of any size with much faster and efficient inference and learning. Defining the loss function to be a sum over the spatial dimensions of the final layer then the stochastic gradient descent, optimization technique they used, computed on the whole image will be the same as that taking over all of the final layer receptive fields. Traditional classification nets, AlexNet for example, take fixed sized input and produce outputs with non-spatial properties.

The idea of converting these classification nets to what they called *fully convolutional networks* lies behind the idea of viewing the fully connected layers in the networks as convolutions with kernels that cover their entire input region.

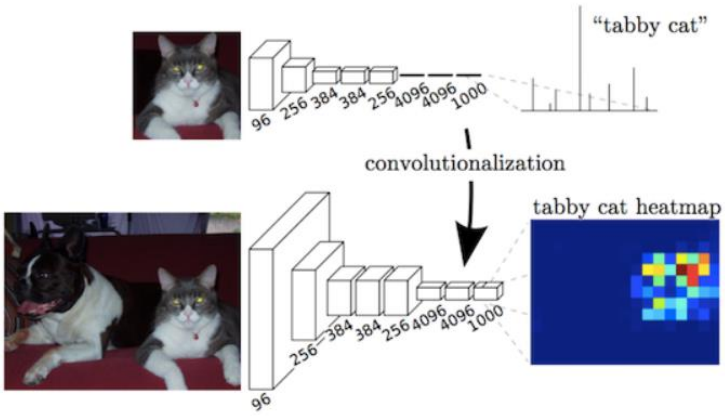


Figure 2.1 Converting classification nets to FCNs that produce feature maps

This is equivalent to evaluating the original classification network on overlapping input patches but is much more efficient because computation is shared over the overlapping regions of patches. Thus, the spatial output maps make them the natural choice for dense problems like semantic segmentation. Both learning and inference are performed whole-image at a time by dense feedforward computation and backpropagation.

The computation of both of them is greatly reduced in time by taking advantage of the aggressive optimization done in convolution. After producing the convolutional fully connected layers, the feature maps obtained need to be up-sampled in order to do a pixel wise prediction. To connect the coarse outputs back to pixels, they used deconvolution layers that is a backward convolution that up samples the outputs to the pixels. Noting that the deconvolution layers need not to be fixed it can be learned. They found that such a deconvolution step is fast and effective for dense prediction. It is noted that they trained with a per-pixel logistic loss function and validated with the standard metric of mean pixel intersection over union, with the mean taken over all classes, including background.

Combining the steps explained above, the architecture of segmentation method consists of, (A) convolutionalizing classification nets, as described earlier by converting fully connected layers to convolutions, in order to produce feature maps with spatial outputs. This is followed by (B) a deconvolution layer to up sample the coarse outputs to pixel outputs. Since up sampling produces coarse segmentation maps it causes loss of information during pooling and can limit the detail in the up sampled output. For this reason, they added (C) skips between layers to fuse coarse appearance information. These skips work on combining final prediction layers with lower layers that have finer strides thus improving segmentation details. (figure 2.2)

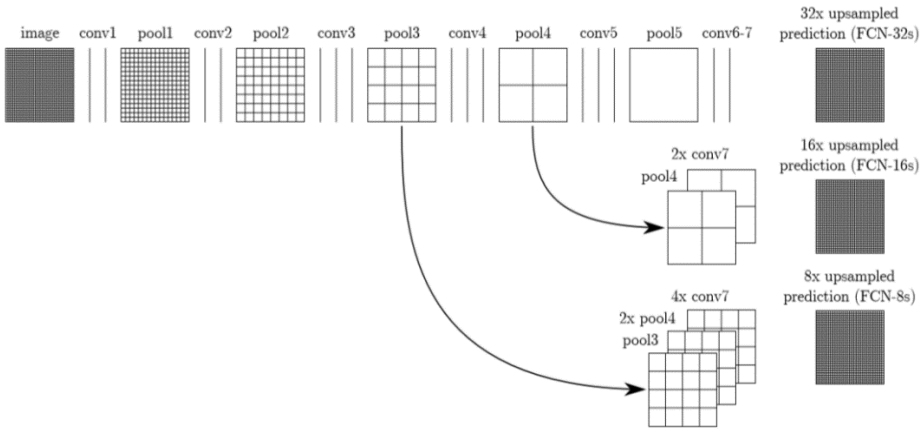


Figure 2.2 the segmentation architecture of FCN

As shown in figure 2.3, as they fuse information from lower layers to higher ones, the performance improves.

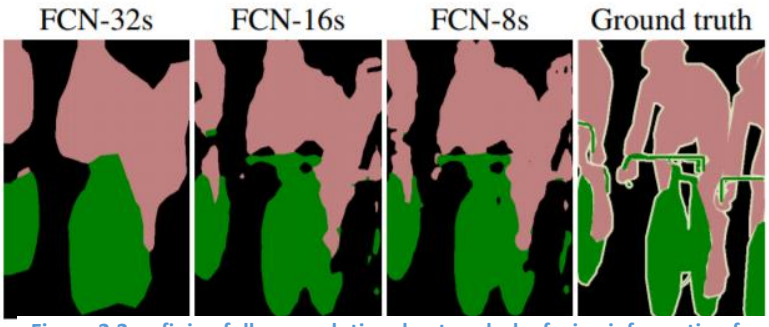


Figure 2.3, refining fully convolutional networks by fusing information from layers with different strides improves spatial detail.

An architecture that depends heavily on the FCN concept is the U-NET which was one of the first segmentation deep networks. Next, we discuss the U-NET deeply to use it as a main classifier.

ii. U-NET

Many image segmentation methods require the use of large training datasets in order to output a good accuracy. In most cases, these large training sets are beyond reach and cannot be provided at the time of training. U-Net [8], a convolutional neural network built in 2015, uses the available samples more efficiently relying on the idea of data augmentation. Usual convolutional neural networks have a contracting structure; meaning that as the input image is followed by many convolutional layers and successive fully connected layers, the size of the output is reduced with increasing number of feature channels. The idea of U-net is to supplement this contracting structure by successive up sampling operators which increase the resolution of the output. In addition, feature map copied from the contracting path are concatenated with the up sampled output as shown below in figure 2.4.

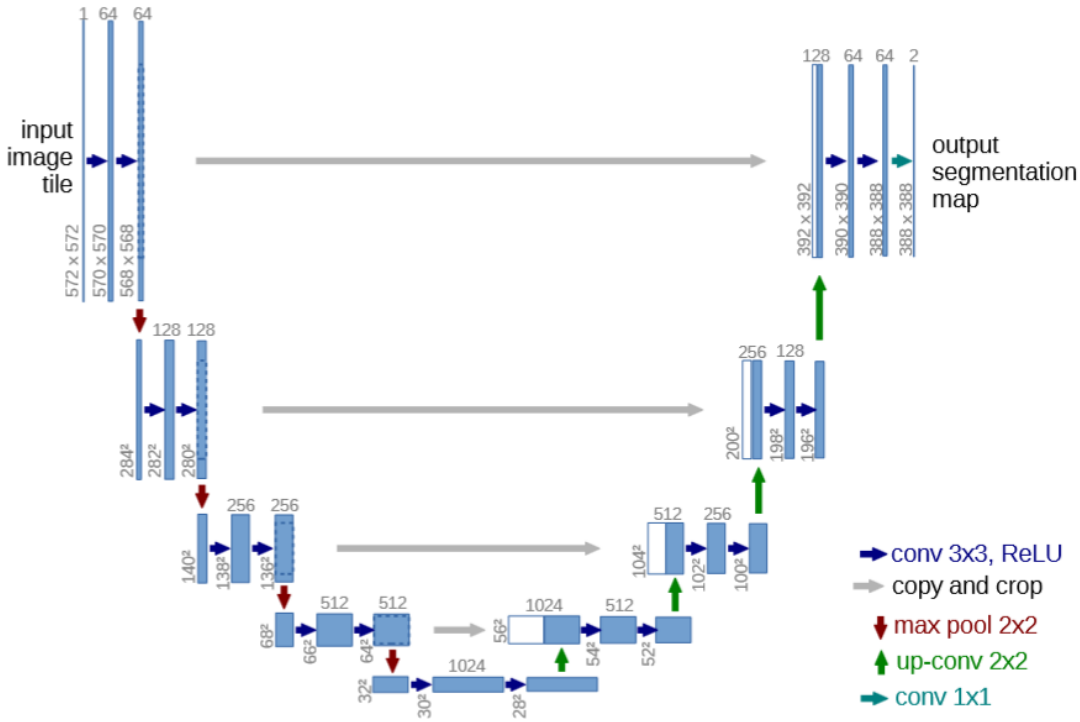


Figure 2.4. U-net architecture used in the reference paper.

This combination between features from contracting and expansive path allows good localization and the use of context. As a result, both the contracting and expansive paths are very similar yielding a U-shaped architecture. One important aspect of the U-net is that it does not have any fully connected layers meaning we can perform segmentation on arbitrary-size input and output images. The authors of U-net tested the algorithm against the ongoing competition, segmentation of neuronal structures in EM stacks, where it out performed other competitors. The contracting path looks like typical convolutional neural network architecture; however, it lacks any fully connected layers; each layer consists of two 3x3 convolutions followed by a 2x2 max pooling for down sampling and hence doubling the number of feature channels. In contrast, each layer in the expansive path consists of up sampling unit that halves the feature channels, concatenation with a cropped feature map from the contracting path and two 3x3 convolutions.

In their training, they used the input images and the corresponding maps to train the network with the stochastic gradient descent optimizer. They used the cross-entropy function as a loss function given by:

$$E = \sum_{\mathbf{x} \in \Omega} w(\mathbf{x}) \log(p_{\ell(\mathbf{x})}(\mathbf{x}))$$

Eq1

Where $w(x)$ denotes a weight function that gives some pixels more importance and p_l is a soft max function for every pixel x.

They also noted that for such network with many convolutional layers, a good initialization of the weights is very essential in training. They draw the weights from a Gaussian distribution with a standard deviation of $\sqrt{2/N}$ where N is the number of nodes in one neuron. As the sample provided was small, they needed data augmentation mainly shift and rotation as well as deformation and grey value variations.

In the experiment, they tested the u-net model on many datasets and in competitions. One of them is the segmentation of neuronal structures in electron microscopic recordings where they achieved, without any pre- or post-processing, a warping error of 0.0003529 which is significantly better than the conventional methods such as sliding window network with a warping error of 0.000420. The result of segmentation of neuronal structures is summarized in Table 2.1.

Rank	Method	Warping Error
1	Human values	0.000005
2	u-net	0. 000353
3	DIVE-SC	0.000355
4	IDSIA [1]	0.000420
5	DIVE	0.000430

Table 2.1. Results of U-net model used in segmentation of neuronal structures competition

After reviewing the structure and aspects of the U-net algorithm, we found that it would be very suitable approach to perform a semantic segmentation on the Science Bowl competition.

Now, we introduce Mask RCNN (Regions with Convolutional Neural Network) which is one of the most modern architectures that introduce a modular neural network that can be optimized per application.

iii. Mask RCNN, Fast RCNN and Faster RCNN

Image segmentation is one of the most challenging topics in computer vision due to mainly two reasons. The first is the scarcity of data because it is extremely hard and expensive to label huge datasets

and mask the objects properly. The second reason is the inaccuracy of the masks which limit the training capability of machine learning models. Due to these reasons Image segmentation was approached by different techniques because of its unique nature. The breakthrough in image segmentation performance happened in 2018 when Kaiming He et.al. from Facebook AI Research group introduced Mask R-CNN [2] to the computer vision community. Without any tricks, they managed to outperform all existing models on the COCO dataset, which is one of the little datasets that have labeled images with their associated masks.

Since Image segmentation is considered as an extension to R-CNN, we are going to introduce R-CNN and its variants before Mask R-CNN [2]. R-CNN stands for Region based CNN which is an object detection approach that uses an algorithm prior to CNN in order to produce a manageable amount of candidate boxes that might contain an object within an image. This approach is much faster when compared to other approaches such as sliding window and YOLO which requires scanning the whole image to find objects. R-CNN is considered a meta-algorithm which defines an approach rather than an implementation. This makes R-CNN very powerful as it is upgradable and very flexible. This is also the reason why R-CNN has many variants the most significant of which are Fast R-CNN and Faster R-CNN.

In the original R-CNN paper by Ross Girshick et.al. [13] they used an algorithm called selective search in order to produce the candidate boxes and then they run a CNN on each box separately to classify it. The process is summarized in figure 2.5.

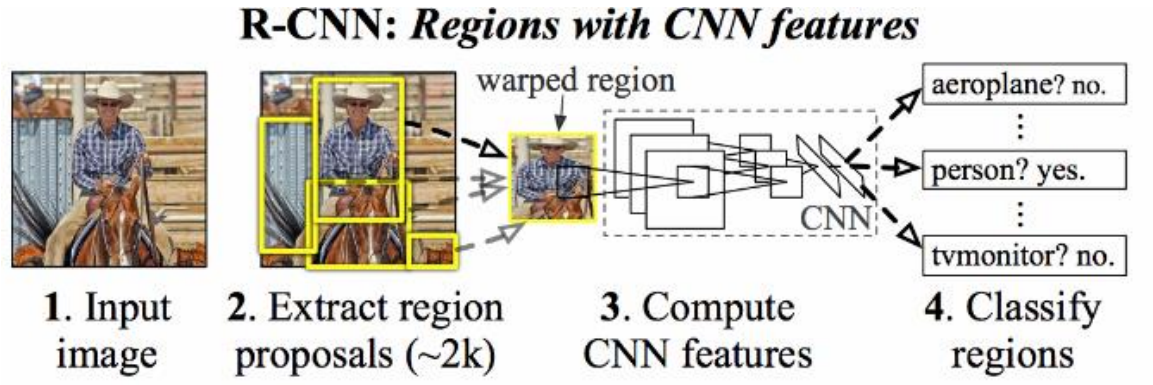


Figure 2.5

This sequential approach is inefficient and moreover CNN are run independently on each box which is extremely inefficient. The second version of R-CNN called Fast R-CNN exploits the fact that the CNN computation can be shared among the different boxes. In Fast R-CNN Ross Girshick introduced the Region Of Interest Pooling layer (ROI Pool for short) which maps parts of a feature space produced from a CNN to their respective boxes. So, to sum up, in Fast R-CNN the input image first passes by a CNN to produce a feature space and in parallel selective search proposes candidate boxes. After that, the ROI pool layer maps

the features to their boxes and finally this output passes by a final fully connected network which classify the boxes and adjust the bounding boxes, the process is summarized in figure 2.6.

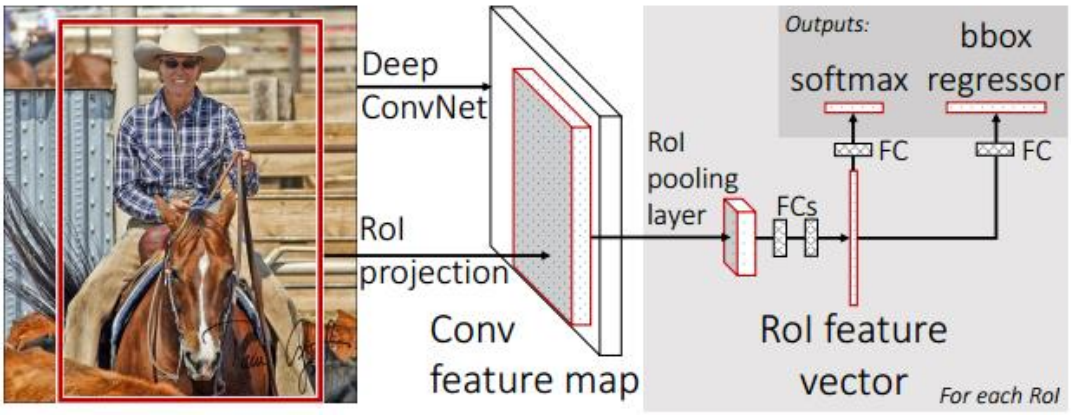


Figure 2.6

The selective search algorithm for region proposal was the bottle neck of Fast R-CNN which made it much less practical and the performance suffered a lot. In 2016, Shaoqing Ren et.al. introduced Faster R-CNN which was a breakthrough in the object detection field. The main idea is that to propose regions in any image one need features, and CNN calculate features that best match the application. So, Faster R-CNN takes advantage of this fact to exclude any region proposal algorithm that acts as a bottleneck in the detection process. Instead, it replaces it with a new simple CNN called Region Proposal Network (RPN for short). This network proposes candidate boxes in place of selective search algorithm. Moreover, to make the algorithm more efficient it shares the same feature space as the other parts of the network. So, now we run a deep CNN on an input image only once to produce the feature space which is used by the remaining subnetworks. RPN uses this feature space to propose candidate boxes which is then used with the feature space by the ROI pool layer to feed the fully connected layers which classifies the boxes and at the same time adjusts the bounding boxes. This approach made Faster R-CNN [14] much more efficient and at the same time very fast. The process is summarized in figure 2.7.

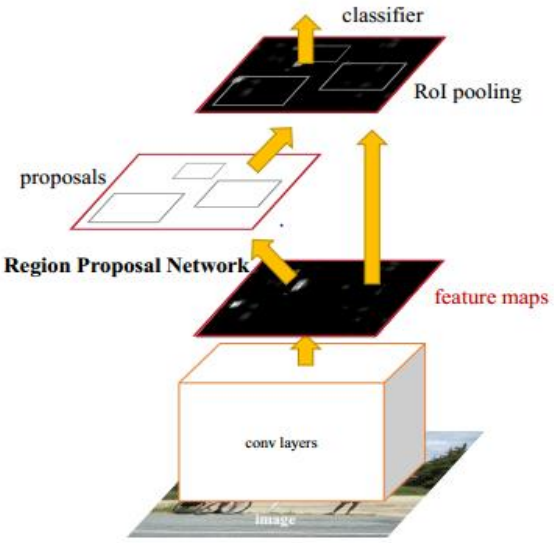


Figure 2.7

Since Mask R-CNN is an extension of Faster R-CNN, we are going to mention the implementation details of Faster R-CNN. Faster R-CNN consists of three main networks the backbone, the RPN and the Fast R-CNN network. The backbone network is a deep CNN network that extracts the features from the images. The RPN and Fast R-CNN network share the same backbone network and thus the feature space needs to be calculated only once per image. In the original paper, they used ZF and VGG-16 networks. However, now there are more powerful backbone networks such as FPN and Resnet with its variants. We are going to use FPN with ResNet or DenseNet as they provide superior performance. Due to the large number of layers in these networks, we are going to use them with pretrained weights to decrease our training time.

Next comes the RPN which is used to produce candidate boxes that might contain objects. The RPN takes the feature space from the backbone network as an input and then it uses a sliding window with anchor boxes to produce a bounding box dimensions and objectness score per anchor. The objectness score is the probability that the box contains an object. The algorithm starts by sliding an $n \times n$ window with $n = 3$ over the feature space. Each window has a set of k boxes with predefined dimensions called anchor boxes to differentiate between multiple objects in the same window.

The window is fed to a simple CNN that maps the window to a lower-dimensional feature vector which is finally fed to two fully connected layers a box regression layer that adjusts the dimensions of anchor boxes and the other layer is a box classification layer that gives each anchor box two probabilities being an object or a background. Thus, the size of the output of the box regression layer is $4k$ and that of the classification layer is $2k$. In order to make the implementation efficient the sliding window is implemented convolutionally which leads to the computation being shared among all windows. The network is summarized in figure 2.8.

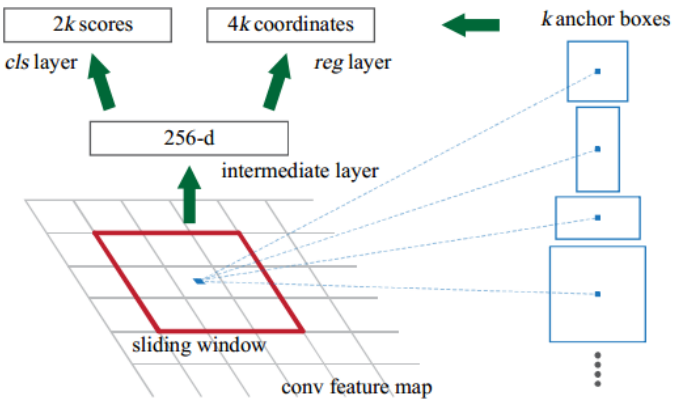


Figure 2.8

It is obvious that many anchor boxes will detect the same object. To solve this problem Non-Maximum Suppression (NMS for short) is used. In NMS the intersection over union (IOU) of the neighboring boxes is calculated and if the IOU over a certain threshold about 0.7 the anchor box with the highest objectness score is chosen to represent the object while the other boxes are suppressed.

After getting the anchor boxes we move to the Fast R-CNN subnetwork which takes as an input both the feature space and the anchor boxes. A new problem arise which is mapping the bounding boxes from the input image to the feature space. To solve this problem, we must note that the CNN computations preserve the spatial ratio of the image and the only effect is scaling the image down. So, the bounding boxes dimensions is scaled down by a factor that depends on the backbone network and its scaling factor. It is worth to note that approximation is performed which leads to a degree of misalignment between the image and the feature space. After that, the ROI pool layer performs max pooling with a window dimension that depends on the dimension of the bounding box. This variable window dimension is used to fix the output size of the layer to make it for example 7×7. The process is shown in figure 2.9.

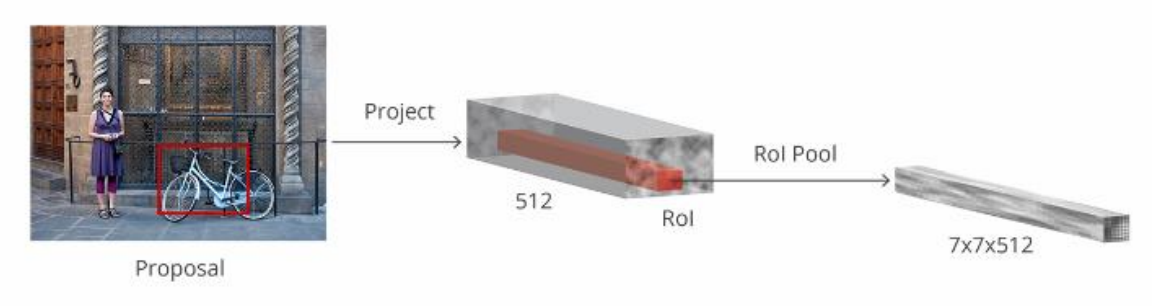


Figure 2.9

Following the ROI pool layer comes the final subnetwork which is the remaining of the Fast R-CNN network which takes the ROI pool fixed size output, flattens it and feeds it to two fully connected layers that finally feeds two layers one is SoftMax to determine the class of the bounding box and the other layer is a regression layer that fine tunes the dimensions of the bounding box. The process is summarized in figure 2.10.

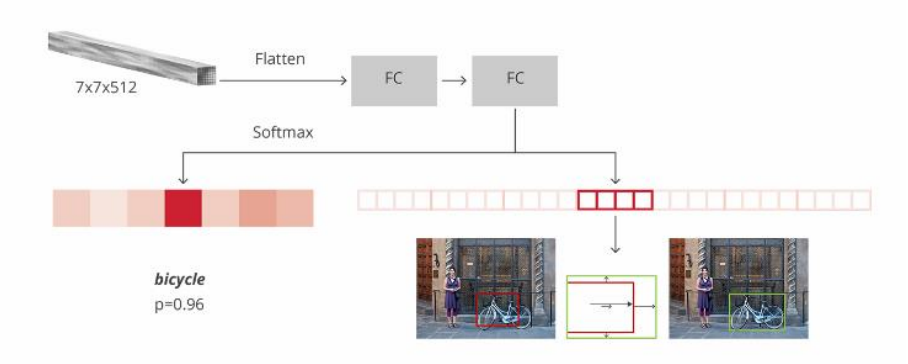


Figure 2.10

The whole Faster R-CNN architecture can be summarized as in figure 2.11.

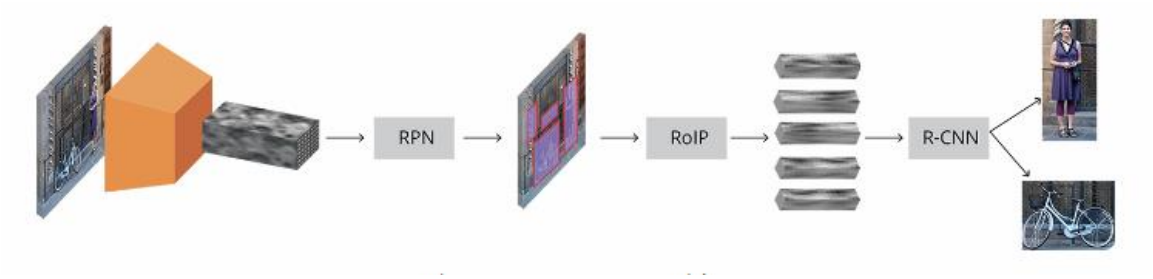


Figure 2.11

There are some important notes that must be taken care of while training. The first is that the backbone is shared between RPN and Fast R-CNN subnetworks. In the original paper they tried training in multistep approach by first training the RPN and then use the weights to initialize training of the Fast R-CNN network. They also tried to train from end to end by alternating the training between the RPN loss and the Fast R-CNN loss. Another important note is that anchors from different images have very different features and if during the training we alternate between different boxes from different images, the training speed becomes extremely slow as the network regards the inputs as noise. To solve this problem and image centric sampling strategy is followed where the mini batch consists of boxes sampled from the same image with the number of positive boxes approximately equal to the number of negative ones. We now move to the changes introduced by Mask R-CNN.

Mask R-CNN is simple to train and use, it is also very fast thus it can be used for real time applications. Mask R-CNN have the same architecture as Faster R-CNN but with an added Fully Convolutional Network (FCN for short) that produce the masks for input images. It also replaces the ROI pool layer with a ROI align layer.

The first change is the introduction of the ROI align layer instead of the ROI pool. The problem with the ROI pool is the approximation performed when mapping from the input to the output and when deciding the dimensions of the pool window. This approximation leads to a misalignment between input image and the feature space window. While this misalignment was not a problem for classification, it is a major problem for making the masks of the image which depends on classifying each pixel. The ROI align performs the same task as the ROI pool but without any approximation. The ROI align uses bilinear interpolation to exactly calculate the value of the required window and pixel place as shown in figure 2.12.

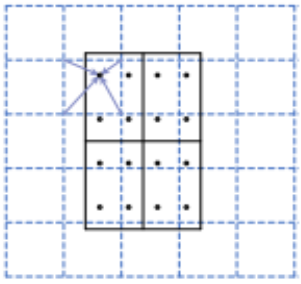


Figure 2.12

The second change is the addition of a new mask branch in parallel to the Fast R-CNN branch that forms the mask for each anchor box. The mask branch is a small Fully Convolutional Network (FCN for short) that takes the output of the ROI align, after normalizing it to a default size, and up sample it using deconvolution layers to return to the same size as the original image with the mask on it. The separation between the Fast R-CNN and the mask branches makes the implementation much more powerful because we can train the mask branch using many classes without coupling the classification and segmentation. Two branches corresponding to two different backbones are shown in figure 2.13.

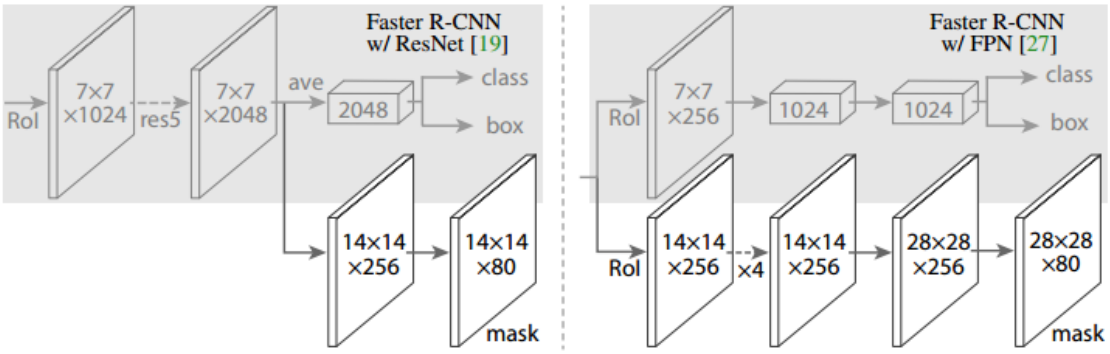


Figure 2.13

Training the Mask R-CNN network follows the same strategy as the Faster R-CNN by alternating the training of the backbone and by choosing the samples in each mini batch. In addition, the mask branch can be pre-trained to increase its performance and initialize it with trained weights.

We now introduce the different backbone architectures that we will use in our project.

iv. Deep Residual Learning for Image Recognition (ResNet):

Deeper neural networks are more difficult to train, ResNets ease the training of networks that are substantially deep. Recent evidence reveals that network depth is of crucial importance, but when deep networks start converging, a degradation problem arises. With the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.

The ResNet paper [3] addresses the degradation problem by introducing a deep residual learning framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layer, residual networkscan be realized by feed forward shortcut connections as seen in figure 2.14.

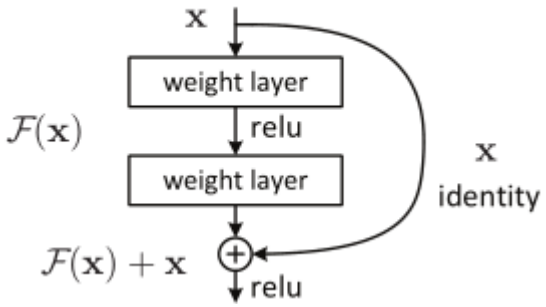


Figure 2.14

These shortcut connections add neither extra parameter nor computational complexity. The extremely deep residual nets are easy to optimize, but the counterpart “plain” nets (that simply stack layers) exhibit higher training error when the depth increases. Also, deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.

The ResNet was tested on ImageNet classification challenge, the architecture is seen in figure 2.15.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.15. Architectures for ImageNet. Building blocks are shown in brackets, with the numbers of blocks stacked. Down sampling is performed by conv3 1, conv4 1, and conv5 1 with a stride of 2.

Results for the plain nets vs ResNets are shown in figure 2.16.

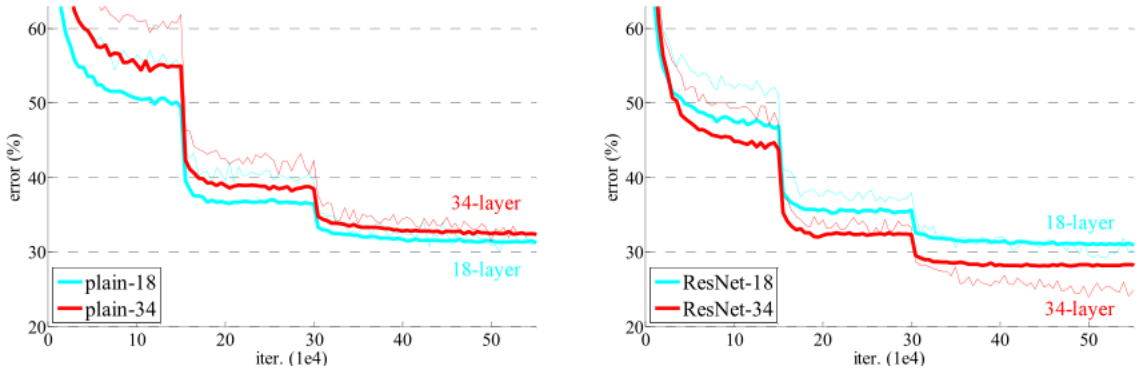


Figure 2.16. Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2.2 18 layers and 34 layers Resnet vs plain Validation Error

The 34-layer plain net has higher training error throughout the whole training procedure, even though the solution space of the 18-layer plain network is a subspace of that of the 34-layer one.

The 34-layer ResNet is better than the 18-layer ResNet (by 2.8%). More importantly, the 34-layer ResNet exhibits considerably lower training error and is generalizable to the validation data. This indicates that the degradation problem is well addressed in this setting and we manage to obtain accuracy gains from increased depth.

-Deeper ResNets are constructed as follows:

50-Layer ResNet: each 2-layer block is replaced in the 34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet. This model has 3.8 billion FLOPs.

101-Layer and 152-layer ResNets:101-layer and 152-layer ResNets are constructed by using more 3-layer blocks. Remarkably, although the depth is significantly increased, the 152-layer ResNet (11.3 billion FLOPs) still has lower complexity than VGG-16/19 nets (15.3/19.6 bil- lion FLOPs).

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 2.3 Error rates (%) of single-model results on the ImageNet validation set

The 50/101/152-layer ResNets are more accurate than the 34-layer ones by considerable margins as seen in table 2.3 We do not observe the degradation problem and thus enjoy significant accuracy gains from considerably increased depth.

Studies on the CIFAR-10 dataset are conducted, which consists of 50k training images and 10k testing images in 10 classes, the results are shown in table 2.4.

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 2.4 Classification error on the CIFAR-10 test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)"

Exploring Over 1000 layers, an aggressively deep model of over 1000 layers is explored. The ResNet 1202 is trained as described above. This 1000 -layer network is able to achieve training error <0.1%. Its test error is still fairly good (7.93%). But there are still open problems on such aggressively deep models. The testing result of this 1202-layer network is worse than that of our 110-layer network, although both have similar training error. It's thought that this is because of overfitting.

v. Densely Connected Convolutional Networks:

DenseNets are those in which all layers are connected (with matching feature-map sizes) directly with each other, as defined in the DenseNet paper [4]. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Features are combined through concatenation not addition.

Hence, the l^{th} layer has l inputs, consisting of the feature-maps of all preceding convolutional blocks. Its own feature-maps are passed on to all $L - l$ subsequent layers. Figure 2.17 shows an example of a 5-layer DenseNet.

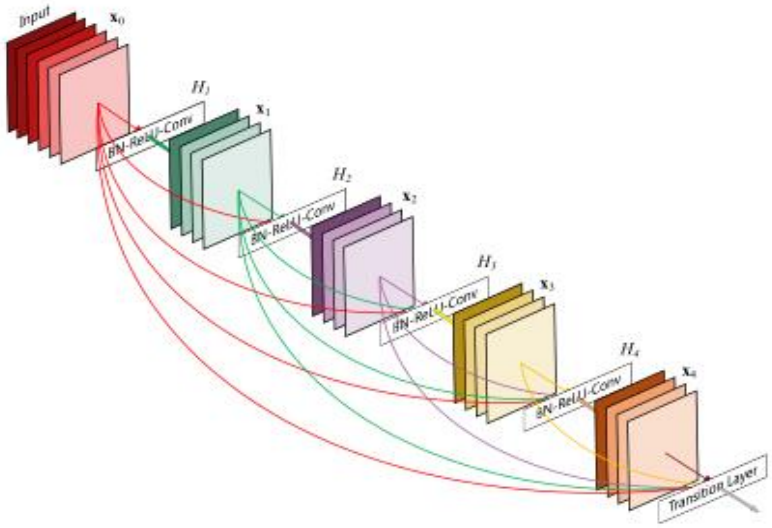


Figure 2.17. A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input.

A possibly counter-intuitive effect of this dense connectivity pattern is that it requires fewer parameters than traditional convolutional networks, as there is no need to relearn redundant feature-maps.

Besides better parameter efficiency, one big advantage of DenseNets is their improved flow of information and gradients throughout the network, which makes them easy to train. Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision. This helps training of deeper network architectures. Further, we also observe that dense connections have a regularizing effect, which reduces overfitting on tasks with smaller training set sizes. Another example of a DenseNet is shown in figure 2.18.

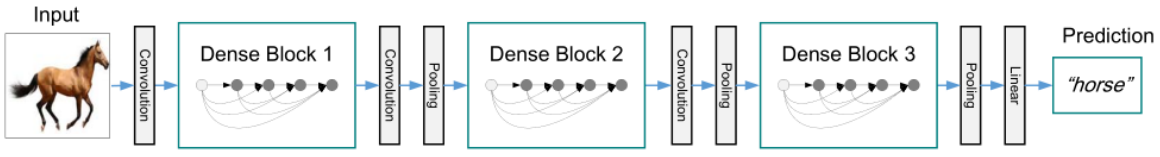


Figure 2.18. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling

Also detailed structure for a DenseNet used on the ImageNet data set is shown in table 2.5.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 2.5. DenseNet architectures for ImageNet. The growth rate for all the networks is k = 32. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

The DenseNet is tested on the dataset benchmark CIFAR and SVHN, the results are compared to state-of-the-art architectures (Table 2.6).

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 2.6. Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are bold and the overall best results are blue. "+" indicates standard data augmentation (translation and/or mirroring). ∗ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

The C10 denotes CIFAR with ten classes, C100 denotes CIFAR with a hundred classes. As seen in the Table 2.6, DenseNet outperform many of the state-of-the-art architects. The results in blue are the best results overall architectures, and it can be seen that all the blue is DenseNet results, this shows how important DenseNets are.

vi. Mobile Nets

In computer vision applications, the demand for higher accuracy is the main addressed issue in the field, and therefore, more complicated deeper networks are surfacing. However, in exchange for the high accuracy, latency and speed have been sacrificed, and thus families of simplified networks were introduced that are actively more efficient regarding speed and size. MobileNets [5], being one of them, is built upon a type of factorized convolution called **depth wise separable convolutions [7] ,** in which the number of parameters is reduced, consequently, the speed increases.

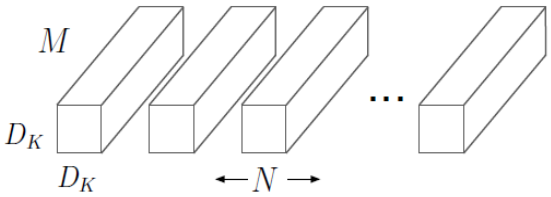


Figure 2.19 standard convolution filters

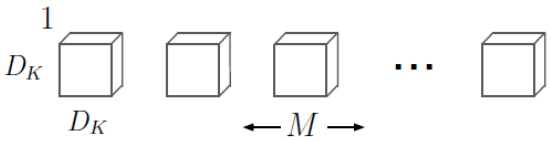


Figure 2.20a Depthwise conv. Filtering stage

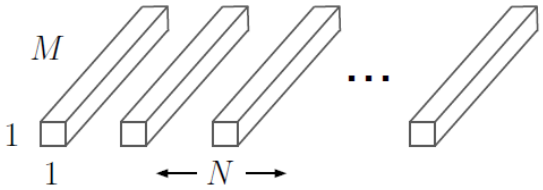


Figure 2.b Pointwise convolution stage

Unlike the regular convolution filters shown in figure 2.19, the depth wise separable convolution is done in 2 stages. Figure 2.20a illustrates the first stage where the convolution is done on the channel level, each filter is applied on one channel. The resulting output will have the same depth as the input volume, only the height and width change. The second stage shown in Figure 2.20b operates on each new entry resulting from the first stage, meaning that the new filters are applied on each pixel and its counterparts in other channels and thus, the output from that stage will have the same height and width of the first stage but different number of channels, specifically the number of filters of the second stage. The output dimensions

are the same in both types of convolution, however, what makes the depth wise convolution superior in terms of efficiency and speed is that it has significantly less computational cost, specifically by a factor of $\left(\frac{1}{N}+\frac{1}{D_k^2}\right)$ where N is the number of output channels and D_k is the dimension of the kernels used in stage 1 of the depth wise separable convolution or the standard convolution.

The structure of the MobileNet, as shown in Table 2.7, is composed of a total of 28 layers, counting the depth wise and pointwise convolutions as separate layers. With the exception of the first and last layers, all layers are the depth wise convolution layers explained above, the first layer, however, is a standard convolution layer and the last one is a fully connected layer followed by a SoftMax layer.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2.7 MobileNet Structure

Further simplification to the base MobileNet can be achieved by introducing two new hyper parameters, Width multiplier ($\boldsymbol{\alpha}$) and resolution multiplier ($\boldsymbol{\rho}$). Both parameters aim to decrease the computational cost even further but sacrificing accuracy in the process. The width multiplier ($\boldsymbol{\alpha}$)**,** which has a range of $(0,1]$, is added to thin out the network, where it's multiplied by the number of input channels (M) and the number of output channels (N), therefore changing the computational cost from $[D_k^2 * M * D_f^2 + M * N * D_f^2]$ to $[D_k^2 * \alpha M * D_f^2 + \alpha M * \alpha N * D_f^2]$ where D_f and D_k represent the spatial dimensions of a square input and the spatial dimensions of a square output respectively. On the other hand, the resolution multiplier ($\boldsymbol{\rho}$), having values < 1, is applied to the dimensions of the input image namely (D_f), this reduction is therefore propagated to every other layer and the total computational cost considering both multipliers become $[D_k^2 * \alpha M * (\rho D_f)^2 + \alpha M * \alpha N * (\rho D_f)^2]$. Values of ρ are typically set so that the input resolution is 224, 192, 160 or 128. Thorough experimentation of the performance of these multipliers and modifications was conducted in [5] against other networks and variations of the MobileNet itself, some results are shown in Tables 2.8a and 2.8b.

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 2.8a Effect of the width multiplier

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Table 2.8b Effect of the resolution multiplier

vii. Feature Pyramid Networks for Object Detection

One of the recent technique to get good results in computer vision is to use pyramids, pyramid representation is representing the image in scale space. There are two types of pyramids low pass and bandpass, the basic idea is to start with an image, filter it, subsample it (usually with a factor of 2) and repeat the process again until you end up with a reasonable size for the image.

What the paper proposes [9] is to use conv-nets to extract the strongest features from the image while in the same time creating scales smaller in size, so the goal of this paper is to naturally leverage the pyramidal shape of a Conv-Net's feature hierarchy while creating a feature pyramid that has strong semantics at all scales without any additional computational complexity.

FPNs take a single-scale image of an arbitrary size as input, and outputs proportionally sized feature maps at multiple levels, in a fully convolutional fashion. This process is independent of the backbone convolutional architectures.

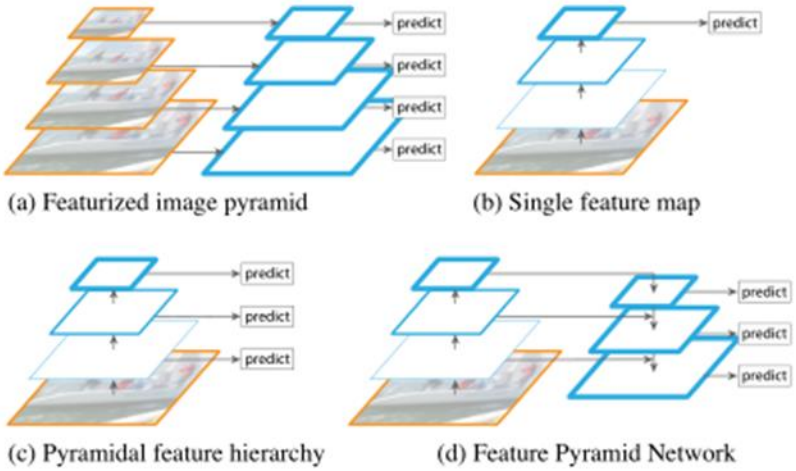


Figure 2.21

The construction of the pyramid involves a bottom-up pathway, a top-down pathway, and lateral connections, as introduced in the following.

The bottom-up pathway is the feedforward computation of the backbone Conv-Net, which computes a feature hierarchy consisting of feature maps at several scales with a scaling step of 2. There are often many layers producing output maps of the same size and we say these layers are in the same network stage. For our feature pyramid, we define one pyramid level for each stage. We choose the output of the last layer of each stage as our reference set of feature maps, which we will enrich to create our pyramid. This choice is natural since the deepest layer of each stage should have the strongest features.

The top-down pathway creates higher resolution features by up sampling the semantically stronger feature maps from higher pyramid levels. These features are then enhanced with features from the bottom-up path way via side connections. Each side connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway. The bottom-up feature map is of lower-level semantics, but its activations are more accurately localized as it was subsampled fewer times. The up sampling of the spatial resolution is by a factor of 2 (using nearest neighbor up sampling for simplicity). The up sampled

map is then merged with the corresponding bottom-up map (which undergoes a 1×1 convolutional layer to reduce channel dimensions) by element-wise addition.

This process is iterated until the finest resolution map is generated. To start the iteration, we simply attach a 1×1 convolutional layer on C 5 to produce the coarsest resolution map. Finally, we append a 3×3 convolution on each merged map to generate the final feature map, which is to reduce the aliasing effect of up sampling. This final set of feature maps is called {P2, P3, P4, P5}, corresponding to {C2, C3, C4, C5} that are respectively of the same spatial sizes.

FPN is a generic solution for building feature pyramids inside deep Conv-Nets, it has been shown that RPN and R-CNN give better results when using FPNs.

1) With RPN

In table 2.9, we can clearly see that using FPN a higher AR (Average recall, which is similar to accuracy) is achieved.

.

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Table 2.9. Bounding box proposal results using RPN [29], evaluated on the COCO minival set. All models are trained on trainval35k. The columns "lateral" and "top-down" denote the presence of lateral and top-down connections, respectively. The column "feature" denotes the feature maps on which the heads are attached. All results are based on ResNet-50 and share the same hyper-parameters

2) With Fast and Faster R-CNN

Using FPN with Fast R-CNN and Faster R-CNN helps increase the AP as seen in table 2.10.

Fast R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(a) baseline on conv4	RPN, $\{P_k\}$	C_4	conv5			54.7	31.9	15.7	36.5	45.5
(b) baseline on conv5	RPN, $\{P_k\}$	C_5	*2fc*			52.9	28.8	11.9	32.4	43.4
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	**45.8**
Ablation experiments follow:										
(d) bottom-up pyramid	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓		44.9	24.9	10.9	24.4	38.5
(e) top-down pyramid, w/o lateral	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*		✓	54.0	31.3	13.3	35.2	45.3
(f) only finest level	RPN, $\{P_k\}$	P_2	*2fc*	✓	✓	56.3	33.4	17.3	37.3	45.6

Table 2.10. Object detection results using Fast R-CNN evaluated on the COCO minival set on a fixed set of proposals all results are based on ResNet-50 and share the same hyper-parameters

Table 2.11 demonstrates that on the test-dev of the COCO dataset, the FPN method increases over the existing best results by 0.5 points of AP (36.2 vs. 35.7) and 3.4 points of AP@0.5 (59.1 vs. 55.7).

	image pyramid	AR	AR_s	AR_m	AR_l	time (s)
DeepMask [27]	✓	37.1	15.8	50.1	54.9	0.49
SharpMask [28]	✓	39.8	17.4	53.1	59.1	0.77
InstanceFCN [4]	✓	39.2	–	–	–	1.50†
FPN Mask Results:						
single MLP [5×5]		43.4	32.5	49.2	53.7	**0.15**
single MLP [7×7]		43.5	30.0	49.6	57.8	0.19
dual MLP [5×5, 7×7]		45.7	31.9	51.5	60.8	0.24
+ 2x mask resolution		46.7	31.7	53.1	63.2	0.25
+ 2x train schedule		**48.1**	**32.6**	**54.2**	**65.6**	0.25

Table 2.11. Instance segmentation proposals evaluated on the first 5k COCO val images. All models are trained on the train set. DeepMask, SharpMask, and FPN use ResNet-50 while Instance- FCN uses VGG-16.

It is worth noting that this method does not rely on image pyramids and only uses a single input image scale, but still has outstanding AP on small-scale objects. This could only be achieved by high-resolution image inputs with previous methods.

III. Statement of problems

Upon reviewing the related literature and in alignment with the main objective stated in previous sections, the dataset presented in the Data Science Bowl 2018 competition was chosen as an application for the proposed basis of the project. The objective is to automate nucleus detection, which would in turn speed up research for most diseases and facilitate treatment analysis, since identifying the cells' nuclei is the starting point for most analyses and the most time consuming as most of the subsequent steps are already automated, thus Identifying nuclei allows researchers to identify each individual cell in a sample, and by measuring how cells react to various treatments, the researcher can understand the underlying biological processes at work.

The dataset used is composed of microscopic images of segmented nuclei images. The images were acquired under a variety of conditions and vary in the cell type, magnification, and imaging modality (bright field vs. fluorescence). It is designed to challenge an algorithm's ability to generalize across these variations. Samples from the dataset are shown below in figure 3.1

Figure 3.1 Training set Samples

The challenging aspects of the dataset lie in the fact that the training set is relatively small, hence faulty data would propagate and affect the performance of the whole model. Upon analyzing the dataset, some problematic aspects that were deemed as performance hinderer, were detected. Those include inaccurate masks, missing ones, unidentified objects and poorly annotated masks. Figures 3.2 to 3.5 below illustrate some of those problems.

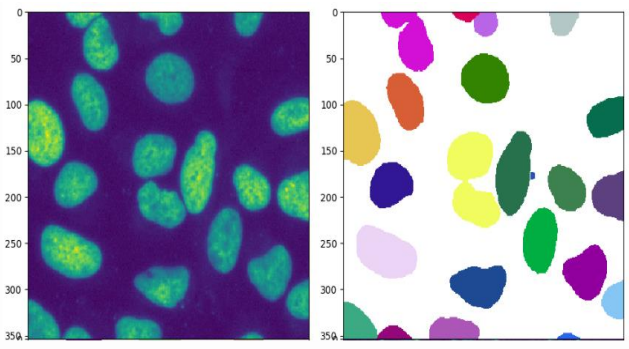


Figure 3.2 Poor annotation, where the two nuclei on the top left side should be separated, Moreover, the next one on the top edge which is marked as two should be considered as one

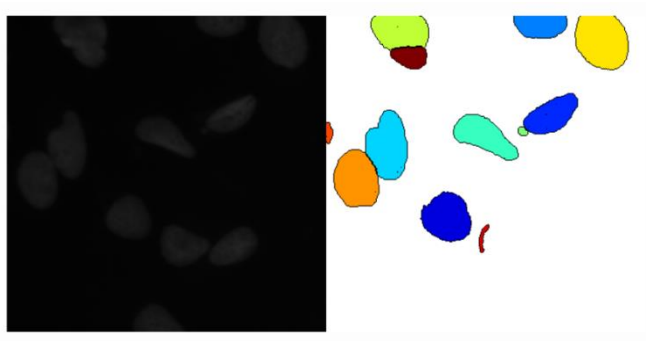


Figure 3.3 Missing nuclei masks

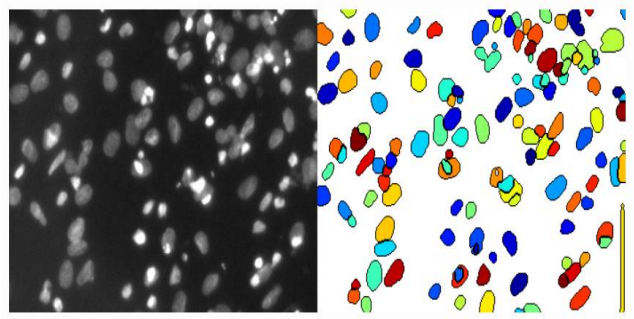


Figure 3.4 Unidentified objects in the bottom right corner

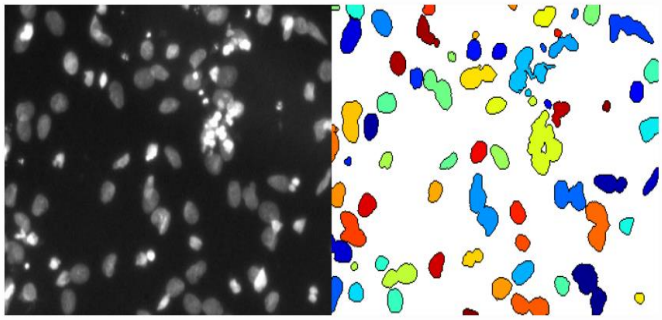


Figure 3.5 Inseparable objects

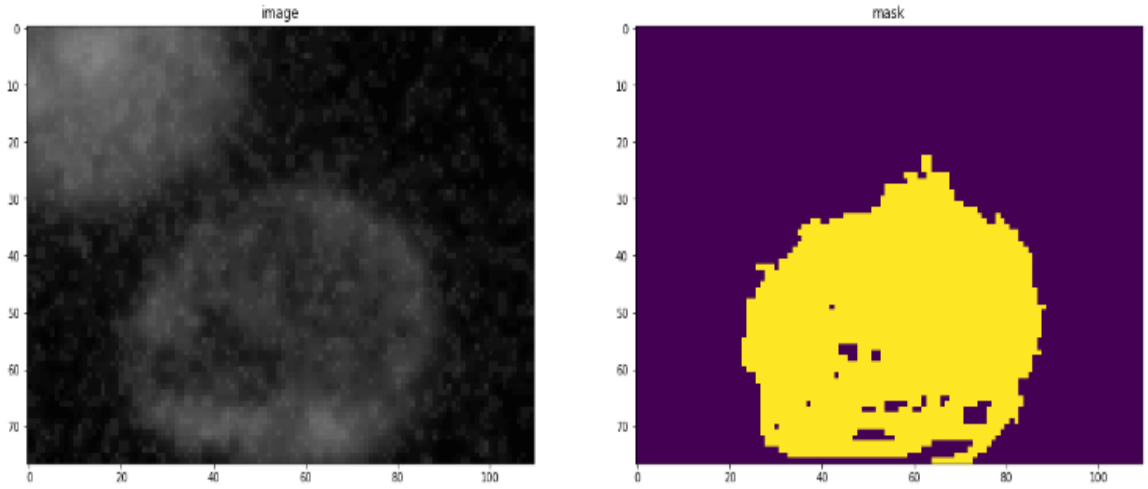


Figure 3.6 Faulty Masks

In order to study the diversity of the data, a statistical analysis approach was adopted, and the results summarized in Table 3.1 were obtained. As observed from the results, there is a significant variation in the sizes of the nuclei, giving a mean nucleus size of 560 pixels with a recorded maximum size of 1243 pixels and a minimum of 21 pixels and a standard variation of 359.9.

	mask_counts	nuclei_size_min	nuclei_size_max	nuclei_size_mean	nuclei_size_std
count	670.000000	670.000000	670.000000	670.000000	670.000000
mean	43.971642	69.270149	1243.029851	560.483462	306.627435
std	47.962530	197.402043	1346.599675	634.752932	359.936976
min	1.000000	21.000000	26.000000	26.000000	0.000000
25%	15.250000	24.000000	287.250000	150.521825	65.940617
50%	27.000000	33.000000	634.500000	265.434343	148.111038
75%	54.000000	63.000000	1733.000000	674.434370	394.974610
max	375.000000	4367.000000	11037.000000	7244.071429	1965.426189

Table 3.1 Summary of the results from the statistical analysis of the data

As the problem encountered is considered a segmentation problem, several algorithms already exist that can be adopted for the application at hand. One of those is the U-NET architecture which is composed of a contracting U-shaped path that enables fast and precise localization. One of the advantages of this approach is that it doesn't require as much labelled images for training as its counterparts in the field, and thus choosing it as a potential performance leader in the nucleus dataset is only logical.

Another potential architecture, which was recently published, is the Mask RCNN dealing with classification and segmentation on the pixel level, as reviewed in the sections above, and is significantly faster than its segmentation algorithms counterparts. It's also worth mentioning that the division of the whole architecture into three separate networks allows for numerous variations to experiment on and a lot of possibilities to explore. The backbone network, the first of these networks that handles extracting features, was the subject of most of the variations done in the model. Three different Backbone networks were tested namely Resnet-101, MobileNet and DenseNet. The performance of the Mask RCNN model with the three different backbones was compared and evaluated according to the chosen criteria, the speed, followed by fine tuning to obtain maximum possible accuracy within the speed criteria needed.

IV. Modeling and Simulation

For all the models we use the stage 2 dataset as a test set and the calculated score by Kaggle is the score we adopt, it is defined as $\frac{TP}{TP+FP+FN}$ based on the IOU result of all masks in all images.

i. U-net

As described in the literature review section, U-net has two paths; a contracting and an expansive path. The contracting path looks like a traditional convolutional neural network that is composed mainly of convolutional and pooling layers except for no fully connected layers. On the other hand, the expansive path consists mainly of convolutional and up sampling units in addition to a corresponding copy of the feature map produced in the contracting path as shown in Figure 2.20.

Model Description:

First, as we analyzed the dataset, we found that the minimum size and one of the most common sizes of the images both in the training and testing sets is 256x256. Hence, we thought it would be more convenient to resize all images to a fixed size for compatibility and manageability. We used the original model proposed in [8] as a guideline and added modification to suite our application and dataset, below we describe our model in details.

After resizing, the input to our model becomes 256x265 images. Every layer in the contracting path consists of two conv units of size of 3x3 with a ReLU activation function followed by a maximum pooling unit of size of 2x2. Hence, we begin with a conv unit of 8 filters followed by a max pool unit and then we follow the output of that by the same layer but with an increasing number of filters which are 16, 32, and 64. Eventually we reach a central layer that separates the contracting and the expansive path. This central layer has two successive conv units with a number of filters of 128. Every unit in the expansive path starts with up sampling unit of size 2x2 followed by a concatenation with the corresponding layer from the other path and finally two successive conv units. This structure allows the number of feature channels to increase in the contracting path while it decreases in the expansive path. Finally, we pass the output through a 1x1 conv layer with only one filter and a sigmoid activation function to ensure that the pixels belong to [0,1]. It is also to be noted that we used 10% of the training set as a validation set. Although this was the main structure of our model, we present many variations in order to enhance the accuracy and our score in the competition, below we present the variations used.

Variations in the model:

The first variation we looked at was the metric function which is was very crucial to the performance of the algorithm. The best metric function that can be used in our model is a one that resembles the one used in the Kaggle competition [10] which is the average precision of intersection over union (IOU) at different thresholds. The IOU is defined as:

$$IoU(A,B) = \frac{A \cap B}{A \cup B} \qquad \text{Eq2}$$

Using IOU for each threshold we can get values for True positive (TP), False negative (FN), and False positive (FP). The true negative value (TN) value is of no interest here since it describes the background. We can then calculate the average precision for each individual image as:

$$\frac{1}{|thresholds|} \sum_{t} \frac{TP(t)}{TP(t) + FP(t) + FN(t)}. \qquad \text{Eq3}$$

The score is then the mean of the above value taken over all images in the test set. We used a metric function that looks like the one described above where the threshold is varied from 0.5 to 1 with a step of 0.05; however, it seems to include the effect of the background as stated by other competitors and in some cases did not improve with time.

We then tried another metric function called dice coefficient which tests the similarity between two images and is very similar to the one proposed in the competition with slight change in the numerator value:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \qquad Eq4$$

As shown below in the results table it leads to better score.

One main strategy to enhance the performance is to include more layers in the network thus make it deeper enough to increase the training accuracy. However, this can yield an overfitting to the dataset and produces low scores when testing. In addition, such network would probably need more epochs to be trained enough. The initial model we used was 4 layers on each the contracting and expansive paths, with around 485,817 parameters to be learnt. When we add a layer on both the contracting and expansive paths, the number of parameters tremendously increased to 1,944,889 which is very huge number to be learnt at the same number of epochs as in the initial model. The results of both models are highlighted in the results table 4.1

below. As first we resized all images to 128x128 so we also did an experiment for this fixed sized and then we resize them all to 256x256.

Another variation we had to look at is the type of the optimizer used and its learning rate. The optimizers we used are Stochastic Gradient Descent (SGD) and Adam optimizer. SGD have been used in many reference papers [8] and [11], so we thought it would be convenient to apply it to the dataset. We varied the learning rate for each of the two optimizers in order to obtain the best value.

In our experiment, we used Keras libraries which provides efficient implementation of convolutional neural network and provides built in functions allowing us to build the U-net structure as described above. Conv2D and MaxPooling2D are examples of such built-in functions. In addition, we used Google colab which provides free Virtual Machines that speed up our training and testing runs as it offers Tesla K80 GPU, 2-core Xeon 2.2 GHz and 13 GB of RAM. In order to limit the progress of low-scoring algorithms we used and early stopping criteria in which the training is terminated if the metric function did not improve for successive 5 iterations.

The figures below represent a random test images with its predicted masks:

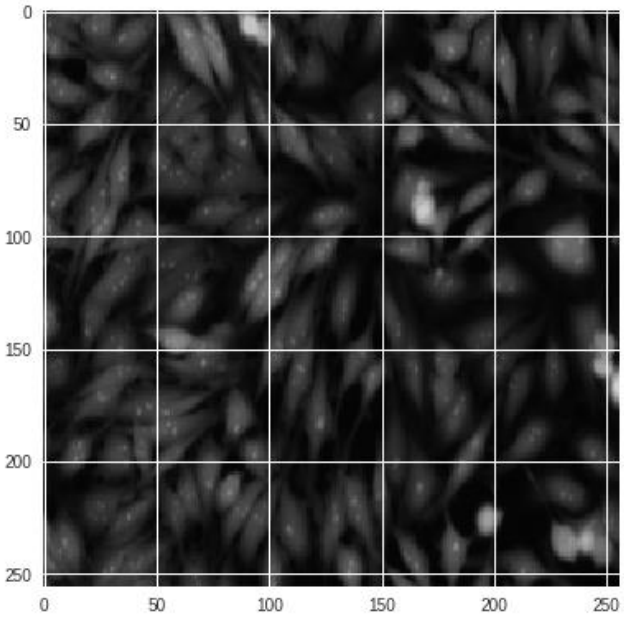


Figure 4.1 Random test image.

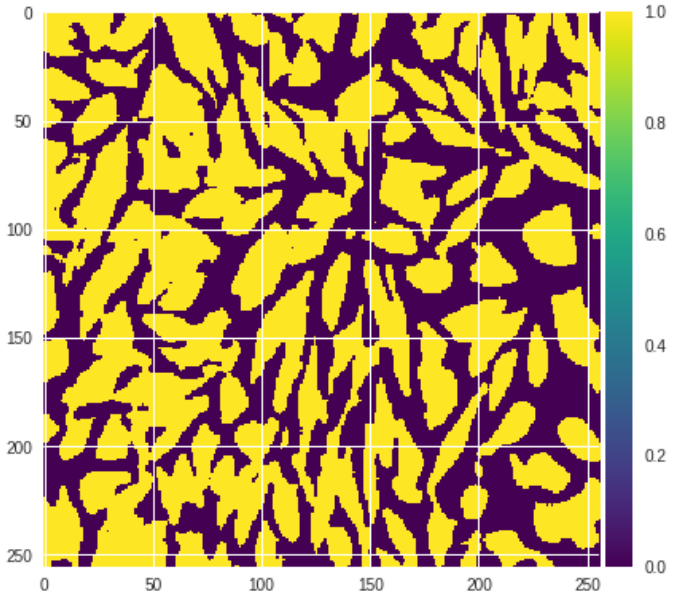


Figure 4.2 the predicted masks of the test image in Fig 4.1

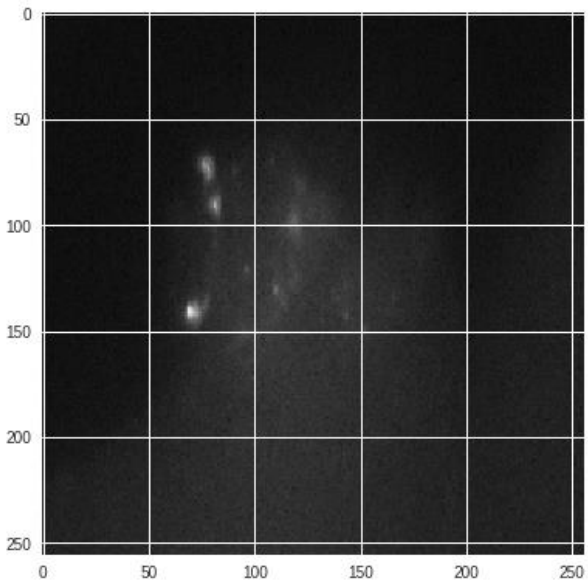


Figure 4.3 Random test image.

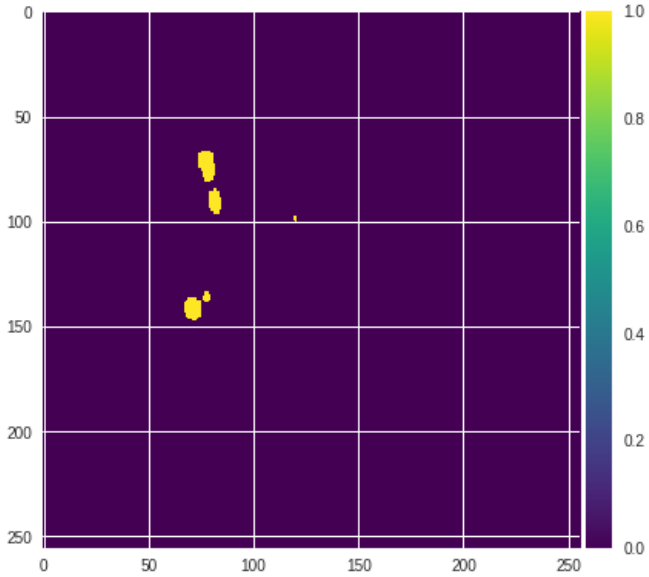


Figure 4.4 the predicted masks of the test image in Fig 4.3.

Results

Method parameters						Results	
Optimizer	Learning rate	Metric	No. of epoch	Resizing Size	No. of Layer	Score	Time
Adam	0.001	IOU	20	128	9	0.259	77 secs
Adam	0.001	dice coefficient	20	256	9	0.296	92 secs
Adam	**0.001**	**dice coefficient**	40	**256**	**9**	**0.326**	346 secs
Adam	0.001	dice coefficient	40	256	11	0.309	445 secs
Adam	0.0001	dice coefficient	50	256	9	0.317	510 secs
Adam	0.01	dice coefficient	50	256	9	0.236	507
SGD	0.01	dice coefficient	50	256	9	0.184	N/A
SGD	0.0001	dice coefficient	50	256	9	Failed	420 secs

Table 4.1 U-Net Results

Comments on Results:

In our experiment, in every test we made we change one parameter at a time in order to see its effect on the resulted score. On the first glance, stochastic gradient descent method did not succeed at producing good results. It yielded the lowest score of 0.184 and failed to score at a lower learning rate. As a result, we choose the Adam optimizer to perform more tests. Comparing the two-metric function IOU and dice coefficient, we found that the latter provide better score at the same learning rate.

Increasing the number of layers in the network does not seem to provide a better score as it increases the number of parameters and makes the network deep enough to overfit the dataset. Finally, changing the learning rate from 0.001, the default value of the Adam optimizer, decreases the score, hence we concluded that 0.001 is the best learning rate with the Adam optimizer.

Looking at the training time, we found as expected increasing the number of epochs or number of layers significantly boost the training time. However, in cases where we used early stopping for not improving metric function, the time cannot be determined for the whole epochs used such as SGD with 0.01 learning rate.

Combining these results highlighted above with the results obtained using Mask RCNN algorithms we present our whole comparison in the conclusion section.

ii. Mask RCNN

Kaiming He et.al. describes a pseudo algorithm in their Mask R-CNN paper [2], this algorithm can be implemented in many different ways and with many combinations. The model as stated earlier consists of four main part.

The first is the backbone network which is the feature extractor in the model. The second is the RPN which classifies the anchor boxes as whether they contain an object or background, it also makes adjustments of the bounding box. The third one is the classifier which gives the proposed boxes their object and make final adjustments to the bounding boxes. Finally, the Mask detection branch predicts the mask for the proposed boxes.

Since the theoretical study discusses a pseudo-algorithm, the implementation is more involved and have added parts to the theoretical study. Kaiming He et.al. implemented the model using Caffe and called it Detectron. The model was sophisticated as they used ResNet as a backbone and on top of it they used a five-layer Feature Pyramid Network (FPN). Each FPN layer is used as an independent feature map. This implementation proved to be very accurate and outperformed the model without FPN.

In recent years, Caffe's community is becoming less and less active due to the appearance of more powerful High-level frameworks such as Keras which is gaining popularity due to its ease of use, logical flow and very strong community support. That's why, we decided to go with Keras in order to try many combinations with relative speed and ease. We adopted the implementation by Matterport on GitHub [12] which is the same as the original implementation but on Keras framework.

Since deep neural networks requires very high computing power and due to the unavailability of a work-station, we had to rely on online solutions. Google Cloud Platform (GCP) and Amazon Web Services (AWS) were both very good choices, however their cost was very high and we couldn't afford it. Fortunately, google released Google Collab which offers free VMs. The VM have a Tesla K80 GPU, 2-core Xeon 2.2 GHz CPU, 13 GB RAM, and 33GB of storage. It offered a 12-hour consecutive runtime which is adequate for our application. However, we had to sacrifice the ease of use because the service is relatively new and without much documentation or tutorials. Another major drawback was the unreliability of the service where the runtime would crash and we lose our work. The service was also unstable and for some days it was unavailable. Despite these problems, it is an amazing tool that is growing rapidly and will help progress the field of deep learning.

The virtual machines provided by Collab run on Linux ubuntu and we can access the terminal from the jupyter notebook which is the only interface provided by Collab. To set the machine we first install TensorFlow 1.4.0 and Keras 2.0.8 which are best compatible with our code. We then use a GitHub API to import the code to the machine. We copy the data from Kaggle servers using the new Kaggle API which simplifies the process of moving the data to the VM. Finally, we install the required libraries to run our code.

Code flow:

The input train data is in the form of nuclei images accompanied by images of its masks. So, we need to do some preprocessing on the images before we can use them to train our model. Ultimately, our model needs the following input to be trained:

1) Image
2) Ground Truth (GT) class label: the class label for every mask in the image (only nucleus in our case).
3) GT bounding box: dimensions of the smallest box to bound each mask. It is used to train the classifier head. This is extracted by calculating the top, bottom, left and right extremes of each mask.
4) GT mask: the actual mask of every nucleus in the image, resized to a predetermined size to make training faster.
5) RPN match: This input has to be generated for every image input where we generate anchor boxes with 3 different dimensions for every pixel in the feature space of the image. We then calculate the IOU of these anchor boxes with the GT bounding boxes and for boxes with IOU ≥ 0.7 we make the RPN match positive for this box indicating that this box has an object. While for boxes with IOU ≤ 0.3 we make the RPN match negative indicating that this box is a background. For boxes in between, we mark them as neutral and they aren't used in training. It is worth to note that for every GT box there must be at least one positive anchor box, if not we loosen the 0.7 requirement and assign the highest IOU anchor box to this GT box.
6) RPN bounding box: the deltas that needs to be applied for the anchor box to make them match the GT bounding boxes. These are calculated only for positive anchor boxes.

The flow of the training is as follows:

1) The image is used to feed an image augmentation library called imgaug which produce variations of the input image to avoid overfitting and virtually increase the effective dataset. The used transformations are chosen to be safe on the masks and are shown in figure 4.5

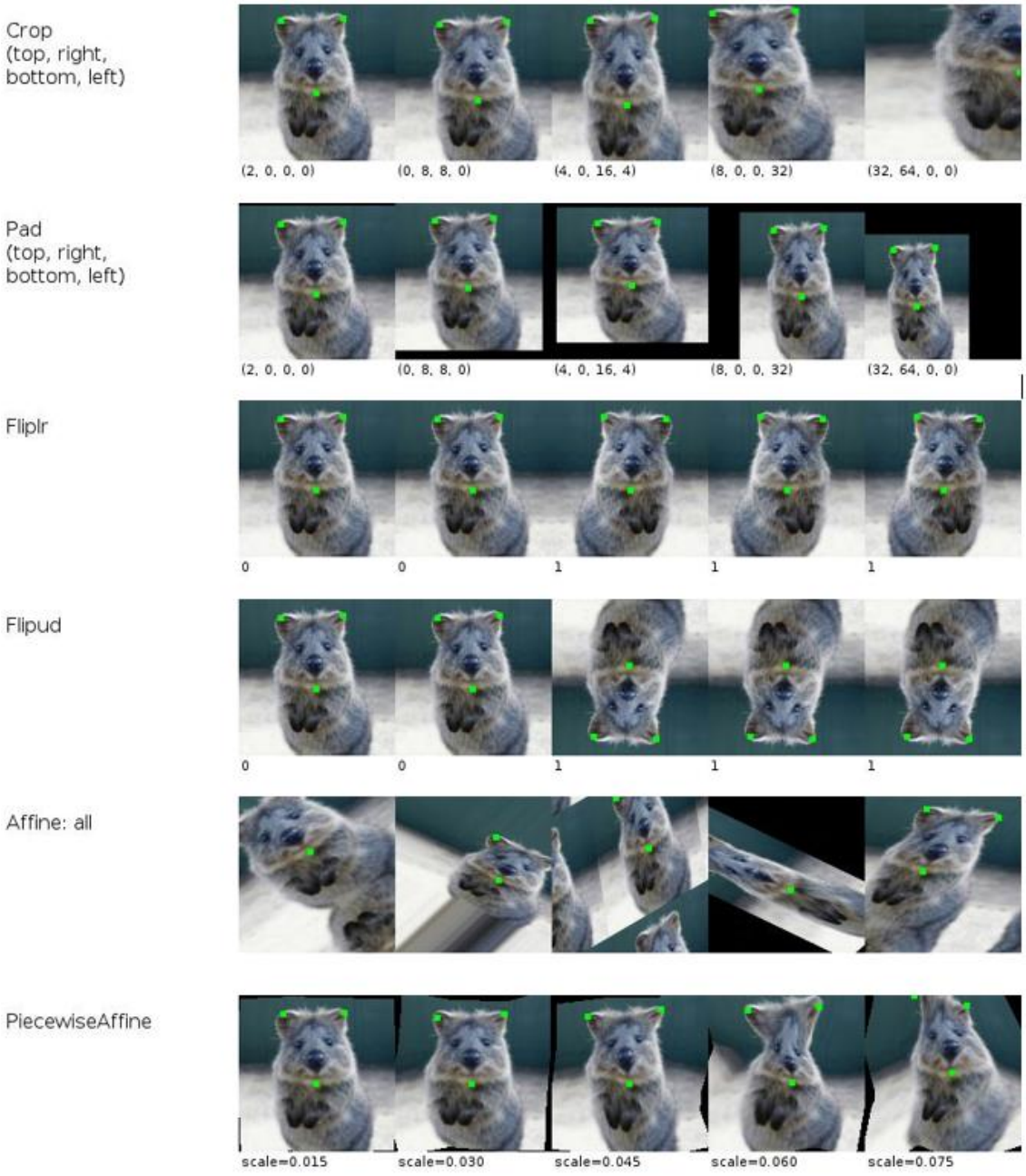


Figure 4.5 applied transformations

2) The feature map is calculated for the input image using the specified backbone.
3) The output of five different stages of the backbone is selected to make the FPN.

4) The final fifth stage is taken as the top of the FPN and then it is up-sampled, to match the dimensions of the fourth stage, and added to the fourth stage output and so on till the feature pyramid is complete.
5) The five different layers of the FPN are treated as separate feature maps over which the RPN is run.
6) For every layer anchor boxes (figure 4.2) are generated with three different dimensions for each pixel and different scales for each FPN layer. These IOU between these anchor boxes and the GT bounding boxes is calculated and depending on the score the anchor boxes are labeled either positive, neutral or negative. For the positive anchor boxes the deltas that refine them to be the same as the GT bounding boxes are also calculated and a sample is shown in figure 4.3. The top 32 positive anchors are chosen with the most negative 32 for calculating the loss with the RPN predictions. If the number of positive anchors is not sufficient the rest of the 64 batch is padded with negative anchors.

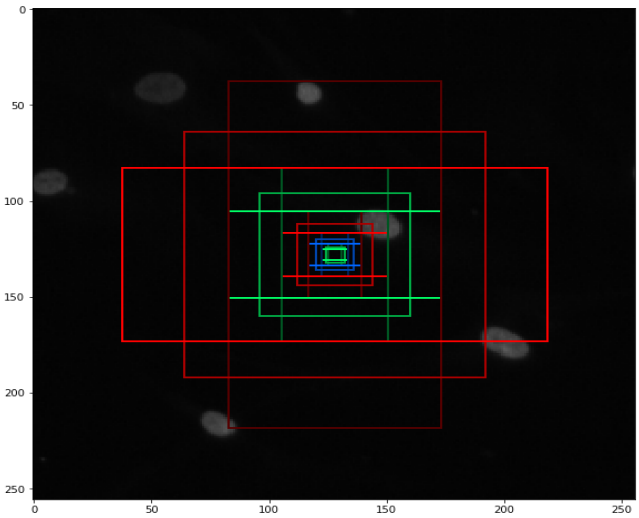


Figure 4.6 generated sample for one feature map pixel

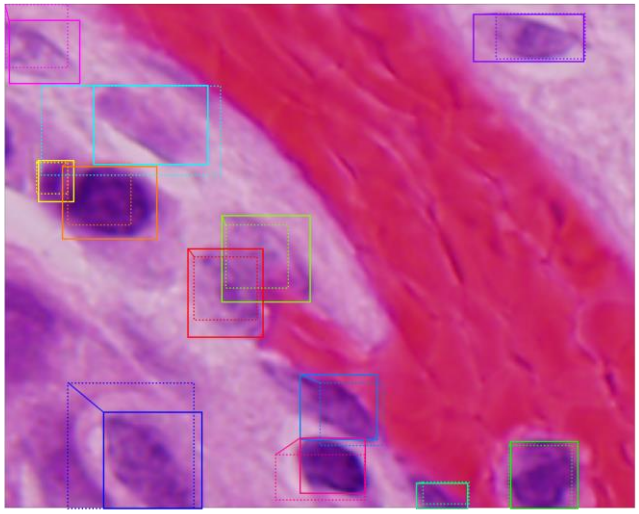


Figure 4.7 positive anchors before refinement (dotted) and after refinement (solid)

7) RPN match and RPN bounding boxes are then calculated from the RPN head. We then perform Non-Max Suppression which eliminates the anchor boxes with IOU ≥ 0.7 while keeping the anchor box with the highest objectness score. Figure 4.8

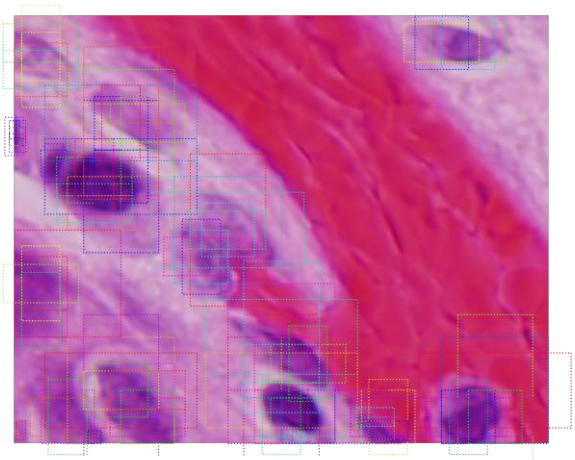


Figure 4.8a top RPN predictions without NMS

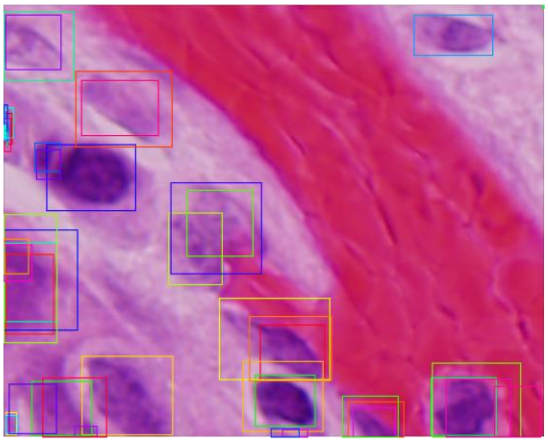


Figure 4.8b after NMS

8) The RPN model loss is calculated as the summation of the cross-entropy loss of the RPN match with the label of generated anchors and the smooth L1 loss of the RPN bounding box with the generated positive anchors.
9) After that we move to training the classifier by taking the RPN positive boxes and passing them to the Region Of Interest Align layer (ROI-Align) which aligns the box with the input image feature map using bilinear interpolation producing new refined boxes from the feature map.
10) The resulting refined boxes are then used as input to the classifier head which predicts the class label and the final bounding boxes adjustments. Finally, NMS is performed once again on the resulting boxes but this time per class to remove highly overlapping boxes with different scales, the result is shown in figure 4.9. These predictions are compared to the GT class label and GT bounding boxes and the loss is calculated in the same manner as the RPN loss to train the classifier.

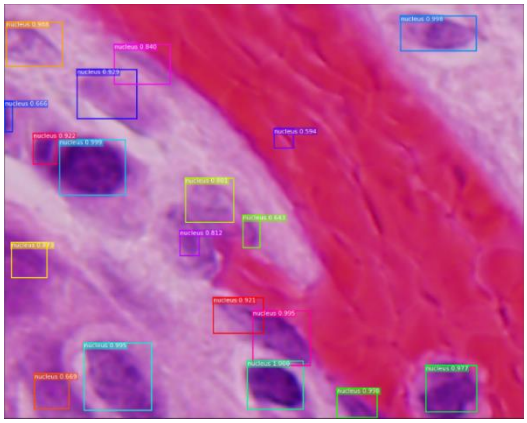


Figure 4.9 final prediction of bounding boxes

11) The same refined feature map boxes are passed to the mask detection branch and resized to a predetermined size. the model then uses Fully Convolutional Networks to predict whether each pixel belongs to the object or not. Finally, the resulting mask is compared to the GT mask and the loss is computer as binary cross-entropy per pixel. The mask branch is trained on this loss.
12) Finally, a total loss which is the average of the above losses is used to train the whole network using SGD.

The architectures of the RPN, Classifier and Mask heads are discussed Section II. We didn't use other networks because they are already simple and powerful.

As for the Backbone, it is the most critical network as it is the feature extractor in the whole model. That's why, we tried different architectures to tradeoff between accuracy and speed. The architectures we tried are ResNet 50, ResNet 101, MobileNet 224 with full width, DenseNet 121,169 and 201 with growth rate = 32. The architectures of these networks are discussed in Section II.

1. Mobile Net Backbone:

In testing, the Mobile-Net as a backbone feature extractor for the Mask RCNN model, the first approach taken was to train the network heads (all layers except the backbone network) first separately then train the model as a whole using the trained heads weights. This approach was adopted mainly since the backbone network is already pre-trained on the Image-Net dataset with available weights, while the weights for the rest of the model are randomly initialized. In addition, in order to find the optimum learning rate, the learning rate was dynamically tuned by comparing the losses curves as the run progresses and changing it accordingly, as shown in figure 4.10.

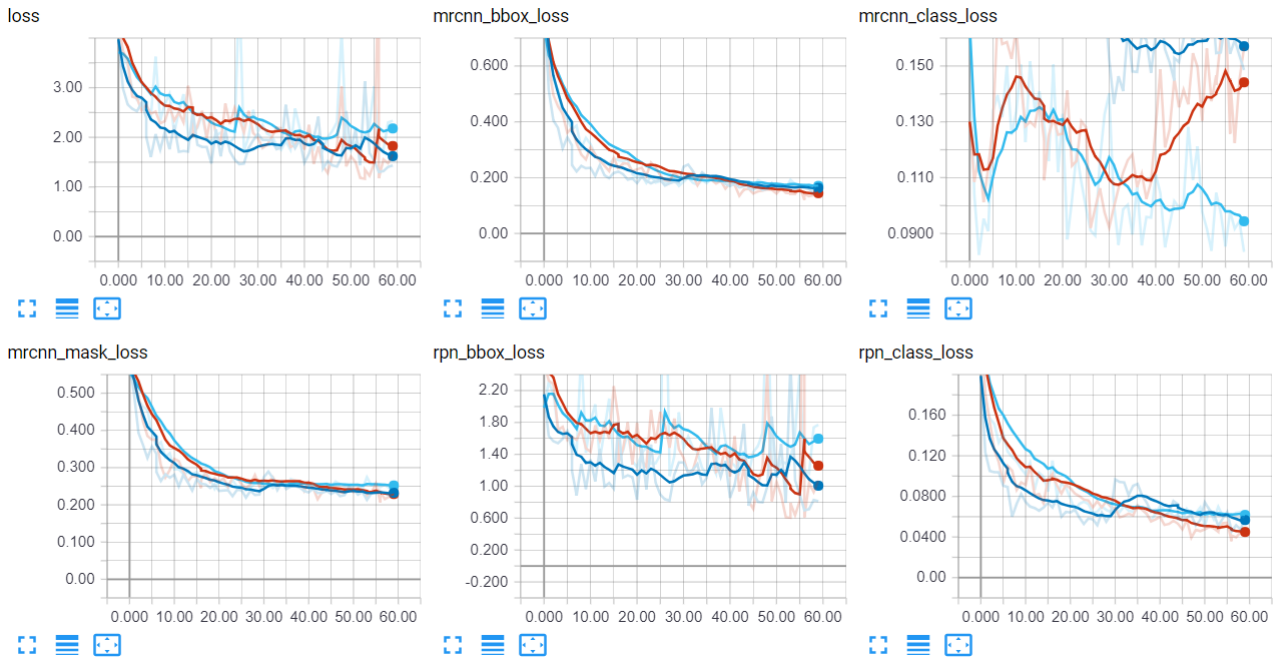


Figure 4.10 Tensorboard visualization of the losses from multiple runs (Runs 4, 5 and 6 in described in Table 4.2 below) for comparisons

Other parameters, besides the learning rate, were also tested, namely the RPN anchor scales, RPN non-max suppression threshold in testing, number of training epochs for the heads and the whole network. The main Results are summarized in Table 4.2.

From the results obtained in Table 4.2, it can be observed that increasing the RPN anchor scales in the second run had a positive effect on the accuracy. This makes sense, since, as mentioned in section II, the mean nuclei size is 560 pixels and thus decreasing the scales further is not an option. A general trend can also be extracted, which is that decreasing the learning rate dynamically with increasing the number of epochs yields better accuracy, this is apparent when comparing the third run with the fourth, and by observing the adopted approach in runs 6 to 10.

The effect of changing the RPN Non-max suppression threshold is observable in the last 4 runs, where increasing it in run 8 improved the score, yet increasing it further yielded no change or worse score as a result. Unexpectedly, dropping out the approach of training the heads separately proved to be an improvement and was therefore adopted for the rest of the testing process.

In order to fine tune the model even further and obtain the optimum score for this model variation, the number of epochs was decreased along with the learning rate and the initiation weights were loaded from the runs that obtained the best results, as in runs 7 through 10.

Run #	Parameters								Result	
	Initialized Weights	Heads		All layers			RPN Anchor Scales	RPN Non-max suppression (Test)	Score	Time (sec)
		No. of epochs	LR	No of epochs	LR					
					50%	75%				
1	ImageNet	20	0.01	30	0.005	0.001	(4, 8, 16, 32, 64)	0.7	0.312	7268
2	ImageNet	20	0.01	30	0.005	0.001	(8, 16, 32, 64, 128)	0.7	0.359	7208
3	ImageNet	20	0.005	30	0.005	0.005	(8, 16, 32, 64, 128)	0.7	0.262	7292
4	ImageNet	25	0.005	35	0.005	0.001	(8, 16, 32, 64, 128)	0.7	0.330	7822
5	ImageNet	30	0.007	30	0.002	0.002	(8, 16, 32, 64, 128)	0.7	0.297	7773
6	ImageNet	-	-	60	0.002	0.001	(8, 16, 32, 64, 128)	0.7	0.362	8401
7	Run #6	-	-	20	0.001	10^{-4}	(8, 16, 32, 64, 128)	0.7	0.381	3763
8	**Run #7**	**-**	**-**	**20**	**0.001**	$\mathbf{10^{-4}}$	**(8, 16, 32, 64, 128)**	**0.8**	**0.387**	**3744**
9	**Run #7**	**-**	**-**	**20**	**0.001**	$\mathbf{10^{-4}}$	**(8, 16, 32, 64, 128)**	**0.85**	**0.387**	**3762**
10	Run #7	-	-	20	0.001	10^{-4}	(8, 16, 32, 64, 128)	0.9	0.383	3738

Table 4.2 Mask RCNN with Mobile Net Backbone Results

2. DenseNet Backbone:

The DenseNet was a very promising backbone candidate due to it being very efficient in the literature in extracting features and achieving very high accuracy on the ImageNet challenge compared to ResNet with considerably much fewer parameters [4]. Surprisingly, this wasn't the case and the results obtained using DenseNet was far more inferior to MobileNet. We tried three different DenseNet architectures, with four dense blocks, each with a different number of convolutional layers in the dense blocks. We tried the 121, 169 and the 201 models for all models we used a growth rate$= 32$ and compression factor $= 0.5$. The result improved as the number of layers increased however it was not as good as MobileNet nor close to ResNet. We tried disabling the batch normalization layers since our dataset is small, but this made the model explode and lead to failed trials. So, we used batch normalization in all

the runs. We used a decaying learning rate as the number of epochs progressed based on the loss curve obtained from Tensorboard. The resulting training and validation losses are shown in figure 4.11

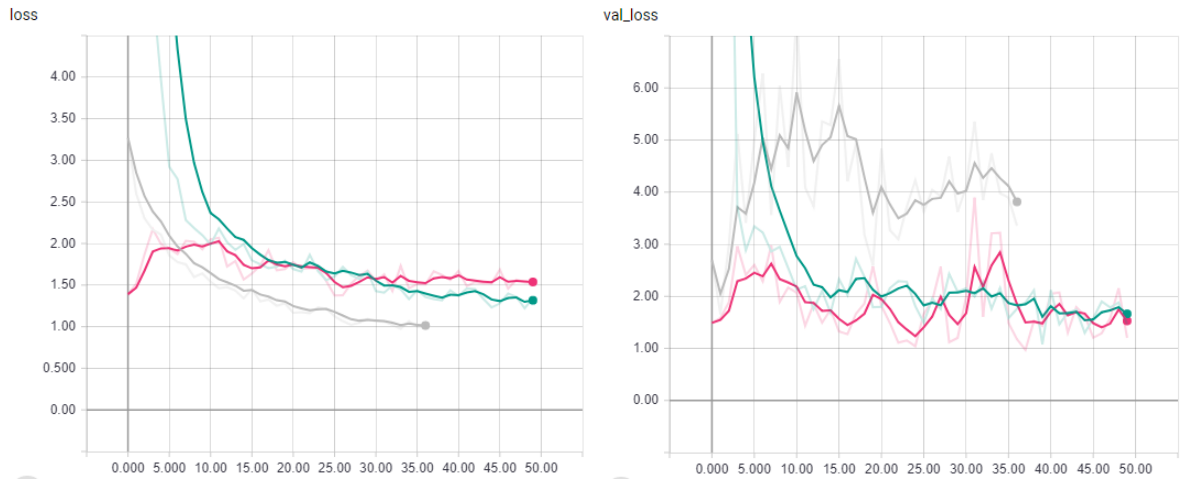


Figure 4.11 DenseNet 121: green, DenseNet 169: grey, DenseNet 201: red. On the left is the training loss. On the right is the validation loss

We used early stopping with DenseNet 169 as it was giving the same values as the DenseNet 201 and the Kaggle submission scores were close. The results can be found in table 4.3

depth	Parameters							Result	
	Initialized Weights	Heads		All layers			RPN Anchor Scales	Score	Time (sec)
		No. of epochs	LR	No of epochs	LR				
					50%	75%			
121	ImageNet	20	0.002	30	$2 * 10^{-4}$	$2 * 10^{-5}$	(8, 16, 32, 64, 128)	0.02	5670
169	ImageNet	20	0.002	17	$2 * 10^{-4}$	$2 * 10^{-5}$	(8, 16, 32, 64, 128)	0.167	6391
3201	ImageNet	20	0.002	30	$2 * 10^{-4}$	$2 * 10^{-5}$	(8, 16, 32, 64, 128)	0.250	8637

Table 4.3 Mask RCNN with DenseNet Backbone Results

From the results, it is obvious that the DenseNet is a bad choice in terms of both accuracy and speed. These results need to be further investigated to know the exact reason behind the failure of the DenseNet as a backbone to Mask R-CNN.

V. Conclusions and Recommendations

The final results are introduced in table 4.4

Architecture	Score	Time in seconds
ResNet 101	0.436	10440
ResNet 50	0.406	8750
MobileNet	0.387	7292
U-Net	0.326	510
DenseNet 201	0.25	8637

Table 4.4 summarized results

As shown in table 4.4 the results of ResNet are the top as they are the best feature extracting but at the same time they are the slowest. At the bottom end, the DenseNet 201 is the very bad in terms of both score and speed mainly because it is a very large network, this needs further investigation because DenseNet is supposed to be an efficient feature extraction network. Although The U-Net architecture is very promising in terms of accuracy and speed, it is limited as we tried a lot but we couldn't make the score any better. In order to do so, we would need to do pre and post processing tricks specific to the dataset which makes U-Net not very practical. On the other hand, the Mask R-CNN proved to be a very strong and flexible model that can be varied to tradeoff between speed and accuracy. It also offers a wide range of hyper-parameters that are general and independent. These parameters can be varied to cope with the application at hand. Also, the quick inference time that can reach 5 fps [2] makes it very practical in real-time applications. It is also very modular which makes transfer learning, where pre-trained modules are used, much more compatible.

Based on that, we recommend Mask R-CNN for instance segmentation algorithms due to its flexibility, that can be used to trade speed and accuracy, and the ease of adapting to different kind of problems compared to U-Net.

References

[1] Hu, H., Lan, S., Jiang, Y., Cao, Z., & Sha, F. (2017). FastMask: Segment Multi-scale Object Candidates in One Shot.

[2] He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2018). Mask R-CNN.

[3] 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). doi:10.1109/cvpr.2016.90

[4] Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2018). Densely Connected Convolutional Networks.

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017, April 17). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision. *Google Inc.* Retrieved from arXiv:1704.04861v1

[6] Sun, Z., & Zhang, J. (2017). Brain tissue segmentation based on convolutional neural networks. 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). doi:10.1109/cisp-bmei.2017.8301979

[7] Sifre, L., & Mallat, S. (2014). Rigid-Motion Scattering for Texture Classification. *International Journal of Computer Vision*.

[8] Olaf,R., Philipp, F., and Thomas, B. U-Net: Convolutional Networks for Biomedical Image Segmentation. In springer, LNCS, Vol.9351: 234--241, 2015.

[9] 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). doi:10.1109/cvpr.2017.106

[10] *2018 Data Science Bowl*. (2018). Retrieved from Kaggle: https://www.kaggle.com/c/data-science-bowl-2018

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," CVPR, 2015.

[12] https://github.com/matterport/Mask_RCNN adopted implementation of Mask RCNN

[13] R. Girshick. Fast R-CNN. In ICCV, 2015.

[14] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In TPAMI, 2017.