FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Learning for identification and quantification of oncocytic cells in microscopic images

Luís Costa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Camacho (FEUP)

Co-Supervisor: Luís Teixeira (FEUP)

March 13, 2019

Deep Learning for identification and quantification of oncocytic cells in microscopic images

Luís Costa

Mestrado Integrado em Engenharia Informática e Computação

March 13, 2019

Abstract

In the recent past, digital pathology has become much more powerful and handful. Slide scanners are able to give, in full detail, images of cells from tissues from patients with different diseases with all kinds of resolutions allowing pathologists to deeply diagnose tissues resultant by biopsies. However, identifying all the problematic cells and regions of interest is a laborious task and it is always dependant on the expert experience and subjectivity.

As big data grows, the need to have powerful tools to analyze complex data increases. Deep learning takes an important role in the image processing field, providing a whole set of models powerful enough to extract the most peculiar features.

Bioinformatics benefits from evolution of these technologies and histopathological image analysis has become a vast field of study. Different models have been proposed on a variety of diseases and its success earned the interest of *Deep Learning* enthusiasts.

The aim of this project is to ease the pathologist task of speeding up and automating the tasks of manually annotating tumor cells and do the counting in order to understand the disease severity. Therefore, we propose a tool for friendly annotation and a framework that processes high resolution images of thyroid tissue to identify and classify *oncocytes*, a type of tumor cells that can be found in a few organs. The detection of these cells can help on the diagnosis and treatment of several diseases such as thyroid cancer. These *oncocytic* tumor cells are larger, have a rounder nucleus, less intercellular space and a more intense eosinophilic staining. In our methodology, we explore these characteristics and present the performance of different *Deep Learning* architectures and their results.

The framework relies on an end-to-end image processing pipeline that includes a tool for image annotation with pre-segmented nucleus. The task of the pathologist is to correctly label each of those images providing a reliable dataset that is used to the model construction.

The results show that image processing is very helpful on histopathological images analysis. Different staining, light conditions and hardware specifications can contribute to image quality degradation. Our image processing module helps to increase state of the art *deep learning* models accuracy.

Keywords: Neural Networks, Deep Learning, Machine Learning, Bioinformatics

Resumo

Nos últimos anos, a patologia digital tem-se tornado cada vez mais útil e poderosa. O uso de equipamentos para digitalização de amostras de tecido resultante de biópsias fornecem, em completo detalhe, imagens de diversas doenças com múltiplos níveis de resolução que permitem ao patologista uma análise profunda e minuciosa.

O objectivo deste projecto é implementar uma ferramenta que ajude o patologista no processo de idenficação e contagem de células cancerígenas que permitem fazer o diagnóstico e ter conhecimento do grau de severidade da lesão. Propomos uma ferramenta de utilização fácil e um processo de construção de modelos para a detecção de oncócitos. Oncócitos são um tipo de célula cancerígena que aparecem em diversos órgãos, nomeadamente a tiróide. Estas células sao conhecidas por terem núcleos mais arredondados e uma forma eosinofílica e elevado número de mitocondria no citoplasma. Na nossa metodologia, exploramos estas características morfológicas e apresentamos o desempenho de diferentes modelos de *Deep Learning* e suas arquitecturas.

O modelo contém um processamento completo de imagens de alta resolução de tecidos da tiróide que depois são apresentadas ao especialista para serem anotadas como cancerígenas ou não. Desta anotação surgem os dados que são utilizados para o desenvolvimento do modelo de classificação.

Os resultados demonstram que o processamento das imagens de alta resolução é muito importante pois existem diversos factores que podem prejudicar a qualidade da imagem. Visto que a principal característica destas imagens é a quantidade de informação, todo o processamento em prol da melhoria da sua qualidade é bastante benéfico para a sua futura avaliação. Os modelos utilizados para a classificação das células cancerígenas demonstram que, com o devido poder computacional, é possível ter uma eficácia elevada, equiparável à classificação feita por especialistas.

Keywords: Redes Neuronais Artificiais, *Deep Learning*, Aprendizagem Computational, Bioinformática

Acknowledgements

I would like to thank my supervisors, professor Rui Camacho and Luís Teixeira of the Faculty of Engineering of the University of Oporto. It was not always easy but the effort was worth it and you always managed to get me in the right path.

I would also like to thank to my friends that have been with me throughout this course and at the end of the day, they are the only ones that can exactly feel your pain.

I am thankful to my family. My mother, my brother and my grandparents for all the support and for reminding me that there is no place like home.

And to my beloved girlfriend that I could never praise enough, I want to thank her for all the effort, all the love and all the nerve. She inspired me to always want something more and she inspired me to do this.

And thank you *Rufus*, my dog, for looking at me while I was blabbering about the all the hyper-parameters that were consuming my soul.

You are the best.

Luís Costa

'Fitter, happier,
More productive,
Comfortable,
Not drinking too much
Regular exercise at the gym, three days a week
Getting on better with your associate employee contemporaries
At ease
Eating well, no more microwave dinners and saturated fats
A patient, better driver
A safer car, baby smiling in back seat
Sleeping well, no bad dreams
No paranoia
Careful to all animals, never washing spiders down the plughole
Keep in contact with old friends, enjoy a drink now and then
Will frequently check credit at moral bank, hole in wall
Favours for favours, fond but not in love
Charity standing orders on sundays, ring-road supermarket
No killing moths or putting boiling water on the ants
Car wash, also on sundays
No longer afraid of the dark or midday shadows, nothing so ridiculously teenage and desperate
Nothing so childish
At a better pace, slower and more calculated
No chance of escape
Now self-employed
Concerned, but powerless
An empowered and informed member of societ, pragmatism not idealism
Will not cry in public
Less chance of illness
Tires that grip in the wet, shot of baby strapped in backseat
A good memory
Still cries at a good film
Still kisses with saliva
No longer empty and frantic
Like a cat
Tied to a stick
That's driven into
Frozen winter stuff, the ability to laugh at weakness
Calm, fitter, healthier and more productive
A pig in a cage on antibiotics'

Thomas Edward Yorke

Contents

List of Figures

List of Tables

Abbreviations

AUC	Area Under Curve
CapsNet	Capsule Networks
CNN	Convolutional Neural Networks
CPU	Central processing unit
CV	Computer Vision
DenseNet	Densely Connected Convolutional Networks
DL	Deep Learning
DP	Digital Pathology
FE	Foreground Extraction
FPR	False Positive Rate
GPU	Graphics processing unit
LR	Learning Rate
MIA	Medical Images Analysis
ML	Machine Learning
NN	Neural Network
RDD	Resilient Distributed Database
ResNet	Residual Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
TPR	True Positive Rate
WSI	Whole Slide Imge

Chapter 1

Introduction

1.1 Context and Framing

In the past few years, as computational power and digital pathology have grown, it is possible to store, process and analyze high resolution medical images, such as slide images from biopsies. Two areas that proved to be useful when processing these images are *Computer Vision* and *Data Mining*. *Computer Vision* is an interdisciplinary field that aims at giving a high-level understanding of the world to a computer from images and videos. Computer Vision has the ability of "making a computer see" and tries to do what a human brain does with the retinal input. *Data Mining* provides a set of techniques and methods that help to classify and build models from data. Combining these two areas, it is possible to extract a set of complementary features that can give us important insights about the phenomenon that produced the data that could not be retrieved by manual methods. The work covered by this thesis is concerned about these technologies and how they can be used to identify, count and classify oncocytes in *Whole Slide Images* (*WSI*). *WSI* are used in Digital Pathology (*DP*) and is currently regarded as one of the most promising avenues of diagnostic medicine in order to achieve even better, faster and cheaper diagnosis, prognosis and prediction of cancer and other important diseases.

DP is the process by which histology slides are digitized to produce high-resolution images. *DP* is becoming increasingly common due to the growing availability of whole slide digital scanners. These digitized slides lead to the possibility of applying image analysis techniques to *DP* for application in detection, segmentation and classification.

1.2 Problem Description

Recent studies on DNA mutations have shown that these are intimately related to complex diseases. One example is the formation of oncocytes which originate tumors. To identify these problems, pathologists have to analyze the phenotype of each cell and its morphology which can a be labor intensive process. Lately, with the growth of digital pathology, it is possible to access high resolution images, with a high number of characteristics and information. These images are

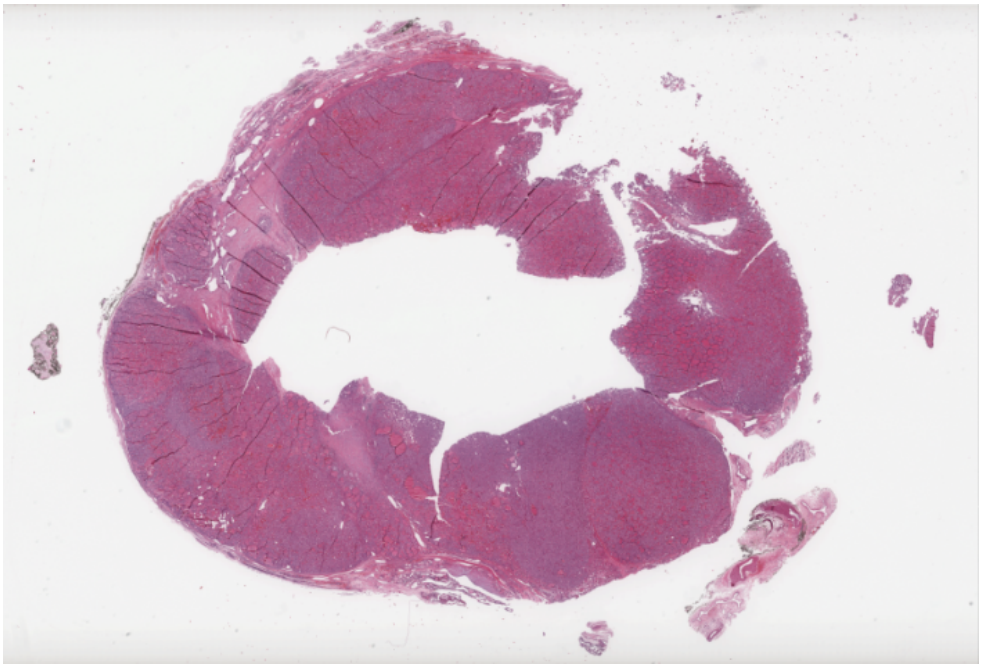


Figure 1.1: Whole Slide Image example at lowest magnification (0 zoom level)

from a microscopic viewing with fine-grained attributes that might be ignored by the person who is examining or escape the human eye. Combining the recent events in Digital Pathology with the evolution of computational power, tools can hopefully be developed to automate and speed up the identification of abnormal cells. A previous study [6], showed that it is possible to do this classification recurring to traditional Machine Learning (ML) algorithms. One such example are Deep Learning techniques that proved to be more effective and robust to either do segmentation and classification than traditional ML methods. In this thesis we try to improve the previous results with a methodology that keeps up with the state of the art in Medical Image Analysis (MIA). Furthermore, we focus on the image processing phase, using state of the art models to do the identification and segmentation of cell nucleus. The high quality microscopic images that will be used are available through the National Institutes of Health (NIH), and one example can be seen in Figure 1.1. In order to successfully classify oncocytic cells, we need to analyze the images with the help of experts. They will help us understand whether a cell shows abnormal characteristics that result in an oncocyte and which features are useful to classify them.

1.3 Motivation

In [6], traditional machine learning algorithms have shown useful and efficient in many contexts as they have the capacity to not only significantly reduce the laborious and tedious act of manually analyze every slide image and provide accurate quantification (e.g. severity of the disease, number of oncocytes), but to act as a complementary insight that can reduce the subjectivity of

each pathologist analysis. With our approach, we intend to improve further the classification reliability, providing a method to ease the quantification (e.g. oncocytes counting) or tissue grading (classification). An increased performance would mean a better diagnosis which leads to a better treatment and better prognosis prediction.

This is a very hard task even for an experienced pathologist. Slide images provide a high level of detail and its analysis require full observation of the tissue and its components. This evaluation relies on the quality of the data provided by digital scanners and on the expert perception which can lead to faulty information.

1.4 Goals

The main objective of this dissertation is to provide a method that would improve the identification, count and classification of oncocytes in digital microscopic images. We focus on the identification methodology proposing some state of the art methods for delineating an accurate boundary for cytologic primitives (i.e cells, nuclei) so that precise morphological features can be retrieved. To accomplish that, few intermediary steps are needed to improve the process. Each of these guidelines need to be taken into account carefully since they have an influence on further analysis.

Our goals include:

- preprocess high resolution images to make the analysis can be computationally feasible
- identify the cells, their nuclei or both taking into account overlapping cells
- distinguish between oncocytic and non-oncocytic cells
- quantify the number of oncocytes present in the slide image
- produce an heatmap that shows regions of interest of a given slide. The heatmap is a friendly representation of our model results.

1.5 Thesis Contribution

We intend to show that DL methods can achieve a superior performance compared with traditional Machine Learning algorithms and reach a performance very close of the classification done by experts. We also propose an Image Processing module that increases the accuracy of several DL models.

1.6 Structure of the Report

Besides this introduction, this dissertation has other 4 chapters. In Chapter 2 we present and discuss state of the art related to the research areas involved in our problem domain that will give us insights about the best approach to be taken. Our methodology can be seen as a threefold

model, sequentially organized. For each topic, we present current solutions that can be applied in our project. In chapter 3 presents the approach that we chose to reach the desired product and we present an experiment conducted to support our methodology and the results obtained by our prototype. In chapter 4 we present our annotation tool, *Oncofinder*, show the application interface and its main features. At last, in chapter 5 discuss our results and point out future improvements.

Chapter 2

Domain and technological background

In this chapter, we provide a meaningful background about our problem and the methods that will be applied to address it. To start, we provide a biological background illustrates some primitive concepts about cells' morphology. This chapter will help to understand which histologic parts we need to take into account in the identification process. The major domains concerning our project and research are associated with *Computer Vision* and *Deep Learning* techniques. We present a set of different methods that can be applied to solve our problem. For each method, we discriminate the advantages and drawbacks in order to choose the best approach and the one that would take us closer to the results we pursue.

2.1 Biological Background

In this chapter we present a brief introduction to the biological concepts that are helpful to understand what is being done and why. First, it is crucial to have an overview of the structure of an animal cell and its histological primitives (e.g, nuclei, mitochondria, ribosomes). Figure 2.1, gives an overview of a cell structure.

As shown in Figure 2.1, one unique cell is composed by a set of components, each of them with a particular function. For this project purpose, the most relevant component are the mithocondrion and the nucleus. Mithocondrion is a double-membrane organelle that is responsible for energy production from oxygen and glucose. The nucleus maintains the integrity of genes and controls the activities of the cell by regulating gene expression—the nucleus is, therefore, the control center of the cell. The main structures making up the nucleus are the nuclear envelope, a double membrane that encloses the entire organelle and isolates its contents from the cellular cytoplasm, and the nuclear matrix (which includes the nuclear lamina), a network within the nucleus that adds mechanical support.

Cancer or malignant tumor is a group of diseases with the potential to invade or spread to several organs. Cancer can initiate in any part of the body and it starts when cells begin to grow out of control and crowd out normal cells [42]. On the other hand, benign tumor do not spread to other organs and can be controlled. Cancer is among the leading causes of death worldwide. In 2012,

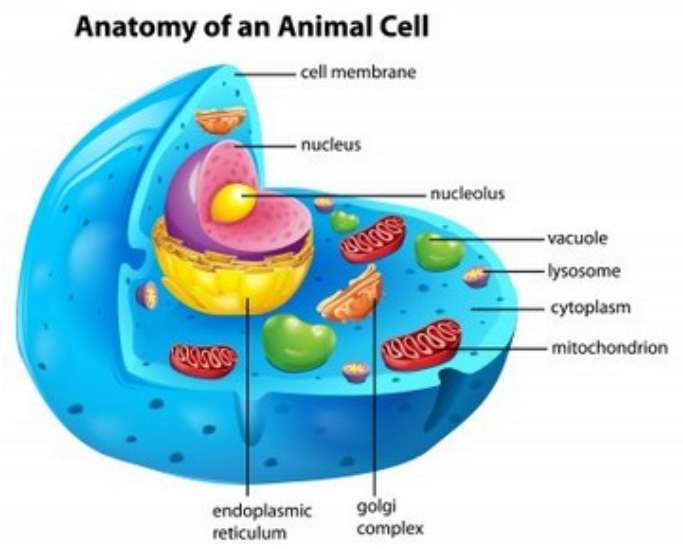


Figure 2.1: Cell Structure[1]

there were 14.1 million new cases and 8.2 million cancer-related deaths worldwide. The number of new cancer cases per year is expected to rise to 23.6 million by 2030. In 2018, an estimated 1,735,350 new cases of cancer will be diagnosed in the United States and 609,640 people will die from the disease. The most common cancers (listed in descending order according to estimated new cases in 2018) are breast cancer, lung and bronchus cancer, prostate cancer, colon and rectum cancer, melanoma of the skin, bladder cancer, non-Hodgkin lymphoma, kidney and renal pelvis cancer, endometrial cancer, leukemia, pancreatic cancer, thyroid cancer, and liver cancer. In this project we will focus on thyroid cancer [26]. Thyroid gland, or simply the thyroid, is an endocrine gland in the neck, below the Adam's apple. The thyroid gland secrets thyroid hormones, which are primarily influence in the metabolic rate and protein synthesis. Thyroid cancer can be detected through oncocytes. An oncocyte is an epitheleal cell characterized by an abnormal number of mitochondria and hence displaying a grainy, eosinophilic appearance and a swollen cytoplasm. Oncocytes also display a bigger and less pleomorphic (more rounded) nuclei. As the cytoplasm is swollen, the cells are bigger which increases the distance between the nuclei and neighbor cells. These characteristics are important to distinguish oncocytes from normal thyroid tumor cells. This celular phenotype can also occur in other diseases but most of these oncocytomas are usually benign displaying low invasiveness, although a few can become malignant, especially in the thyroid. Tumors can contain a mix of cells with and without this phenotype and what determines a tumor as oncocytic depends on the fraction of oncocytic cells within a tumor passing a relatively high threshold - typically 75% [9].

In conclusion, the presence of oncocytes does not determine the existence of a malignant tumor. Therefore, a quantitative and qualitative observation of these cells is needed in order to evaluate the disease severity.

[1] https://biology.tutorvista.com/animal-and-plant-cells/animal-cell.html

(a) (b)

Figure 2.2: Example of an oncocyte (Figure 2.2a) and non-oncocyte (Figure 2.2b)

The difference between oncocytes and non-oncocytes can be seen in the two tissues shown on 2.2. The Figure 2.2a shows a cell with a rounded nuclei in the upper left corner, a swollen cytoplasm and consequently with a higher distance from neighbor cells. This represents an oncocyte. On the other hand, in Figure 2.2b, the image shows how a non-oncocytic (normal thyroid cell) is: more flatten and with no swollen cytoplasm, resulting in a smaller inter-nuclei distance.

2.2 Image Data and Pre-Processing

Biological Images

In Biological Imaging, to highlight distinct structures in microscopy images of tissue samples, tissue staining is commonly used. Frequently two stains, such as hematoxylin and eosin (H&E), are applied for purposes such as discriminating cell nuclei and cytoplasm. Variations in staining results can be normalized by using fully standardized staining protocols. However, a precise control over the stain color and is not possible: stains fade over time, colors may differ slightly, slides may have been imaged on different microscopes.

After the tissue staining with H&E, the histological slides digitization is possible by means of digital scanners or microscopes equipped with cameras. This technique can replace physical slides for educational purposes, remote consultancies and, mainly, for the development of automated images-based systems.

Whole Slide Tissue Images (WSI) refers to scanning of conventional glass slides in order to produce digital slides. This is the most recent imaging modality being employed by pathologists departments worldwide. *WSI* continues to gain traction among pathologists for diagnostic, educational, and research purposes[28]. In Figure 2.3 [16] we can see an example of a WSI.

These images can have a resolution in the 40 000 pixels order both on width and height, which added the problem of being a wide image to display on a computer screen. Downsampling is not an option, since discriminating details could be lost and that would imply a loss of features[16]. Therefore, to solve these problems the WSIs need to be split into smaller tiles. The idea is to take an image and divide it in smaller images that can be computationally processed in reasonable time. However, tiling have some drawbacks that need to be taken into account. As we stated before, these images are very rich and can contain a lot of useful information, when cropping the

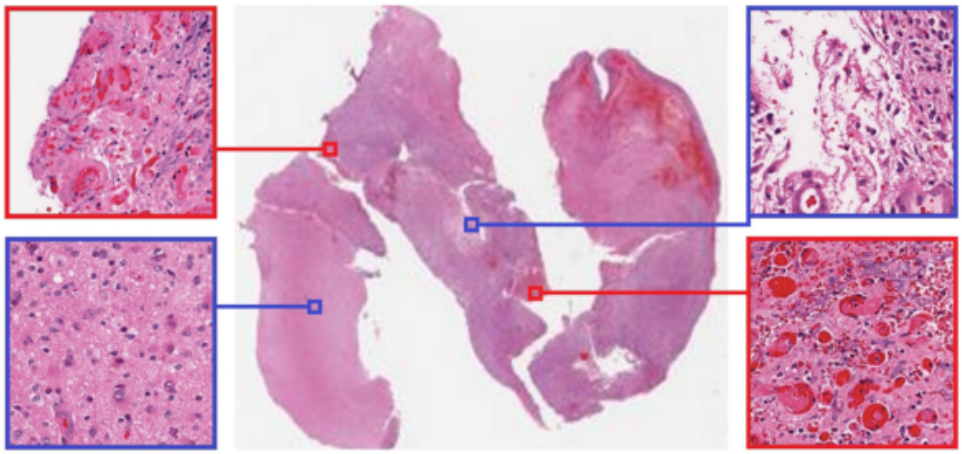


Figure 2.3: A gigapixel Whole Slide Image in the center and 4 tiles and the left and right side.

image we need to be sure that we will not jeopardize its content by losing important components (e.g. cropping a cell in half).

Computational methods are being widely developed for histological image-based applications, since they improve analyses through segmentation until diseases classification. In this project we used Image Processing for the preprocessing phase for foreground extraction, staining normalization and image enhancement.

Available libraries

OpenSlide[2]

OpenSlide is a C library that provides a simple interface for reading whole-slide images, also known as virtual slides, which are high-resolution images used in digital pathology. These images can occupy tens of gigabytes when uncompressed, and so cannot be easily read using standard tools or libraries, which are designed for images that can be comfortably uncompressed into RAM. Whole-slide images are typically multi-resolution; OpenSlide allows reading a small amount of image data at the resolution closest to a desired zoom level. OpenSlide can read WSI in several formats. The WSI provided by NCI are in Aperio (.svs, .tif) format.

OpenCV[3]

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it is free for both academic use. It contains C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was created for computational efficiency and with focus on real-time applications. Written in optimized C/C++, the library can take advantage of

[2]https://openslide.org/
[3]https://opencv.org/

multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

2.3 Deep Learning Background

Deep Learning is a field of a broader family of Machine Learning algorithms. So, before getting into Deep Learning in detail, it is mandatory to provide some foundations of what is Machine Learning and how computers are able to learn. Learning is the process by which an entity acquires information by either experimentation/studying or getting taught. There are two types of learning, supervised and unsupervised.

Many *Deep Learning* algorithms are currently the state-of-art image classifiers [4, 15, 21, 22, 16]. However, due to high computational cost, deep learning algorithms cannot be applied to very high resolution images, such as microscopic images due to the high number of features and lack of computational power to process them. MIA requires a careful and detailed preprocessing in order to decrease the dataset magnitude and make it suitable for feature extraction.

Supervised and Unsupervised Learning

Supervised Learning

Supervised is the type of learning where we help the computer determine what it needs to achieve by giving it the desired results along with the input in the training data. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. A common problem of this kind of approach is overfitting. *Overfitting* happens when the model learns the data instead of learning the functions, so when presented with new data that was not present initially it performs poorly because it is too adapted to the training set. This phenomenon usually happens when the dataset is too small or homogeneous.

Unsupervised Learning

Unsupervised Learning is the task of inferring a function that describes the structure of *unlabeled* data. In supervised learning our data is classified or categorized and we know the result and by inferring a function from the initial dataset we can map new examples. This does not happen in unsupervised learning, since the data is unlabeled there is no straightforward way to evaluate the accuracy of the structure that is produced by the algorithm. Instead it is used to find relationships among the given data, for instance, clustering, which is one of the most commonly unsupervised learning techniques.

In our project, only Supervised Learning is used.

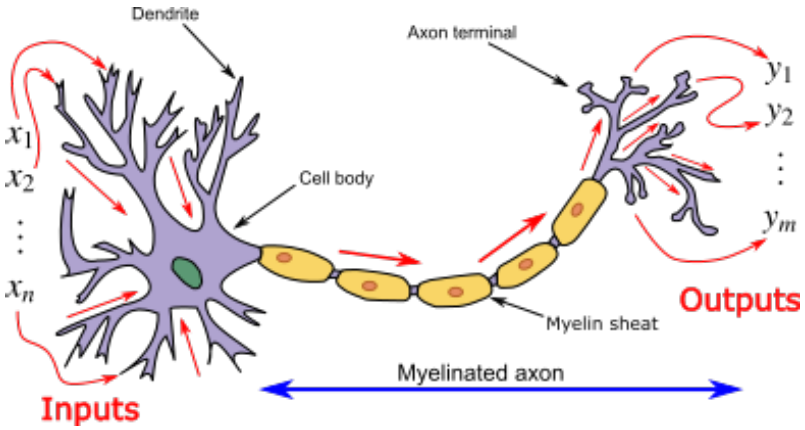


Figure 2.4: Biological neural network[4]

Deep Neural Networks

Neural network or artificial neural network is one of the frequently used buzzwords in analytics these days. Neural network is a machine learning technique which enables a computer to learn from the observational data. Neural network in computing is inspired by the way biological nervous system process information.

Biological neural networks consist of interconnected neurons with dendrites that receive inputs. Based on these inputs, they produce an output through an axon to another neuron (Figure 2.4).

In the computing world, neural networks are organized on layers made up of interconnected nodes which contain an activation function. These patterns are presented to the network through the input layer which further communicates it to one or more hidden layers. The links between the output of a neuron and the input of another one have numeric weights assigned that represent the strength of the connection between the two nodes. The hidden layers perform all the processing and pass the outcome to the output layer (Figure 2.5).

There is no general consensus about the definition of deep neural network but it can be seen as any artificial neural network that has several hidden layers, typically more than two. By having multiple layers these networks are able to learn representations of the data with multiple levels of abstraction [13]. Deep Learning solves the central problem in representation learning by introducing representations that are represented in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts. Figure 2.6 shows how a deep learning model can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are defined in terms of edges [13].

In 2.6 we provide an illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. Learning or mapping this mapping seems insurmountable if tackled directly. So,

[4]https://en.wikipedia.org/wiki/Biological_neuron_model

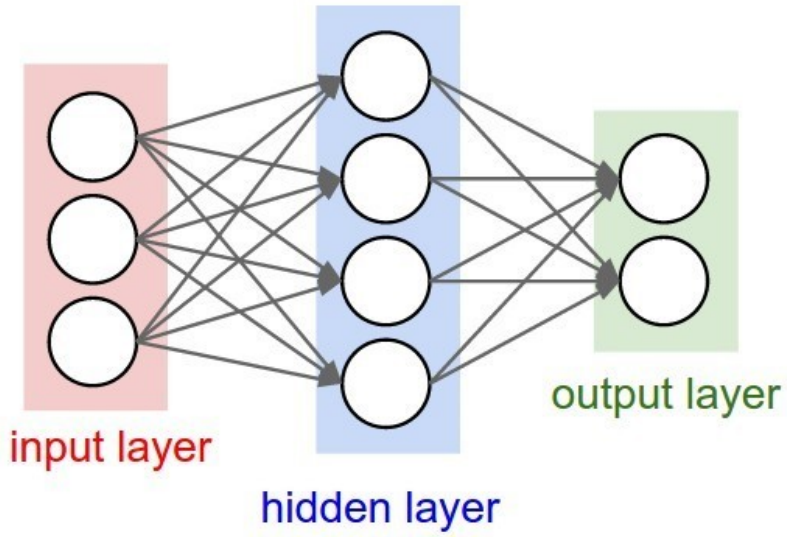


Figure 2.5: Simple Feedforward Neural Network[5]

Deep Learning methods solve this problem by dividing the desired complex mapping into a series of nested simple mappings, each described by a different layer of the model. If we look at the picture, the **visible layer** that, as the name says, contains variables that are actually visible in the image (e.g the color of the person's shirt, the skin color, the foreground, etc). Then we have a series of **hidden layers** that increasingly extract abstract features from the image. These features cannot be observed in the image, that is why they are called *hidden*. Instead the model should be able to determine which concepts are useful for explaining the semantics in the given data.

Given the pixels, the first layer can easily identify edges, by comparing the brightness and neighboring pixels. The first hidden layer searches for corners and contours, which are recognizable as a collection of edges. The second hidden layers give information about the image in terms of edges and contours and then the third hidden layers can detect entire parts of specific objects, by finding specific collections of contours and corners [44]. The description of this image in terms of geometric primitives can be used to recognize objects in the image.

Training the models - Backpropagation

Backpropagation is the common method to use when training an artificial neural network. It is commonly used with an optimization method called gradient descent. Gradient Descent is an optimization algorithm that is used to finad a local minimum of a function, called the cost or error functions. There are three variations of gradient descent [33]:

[5]https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223

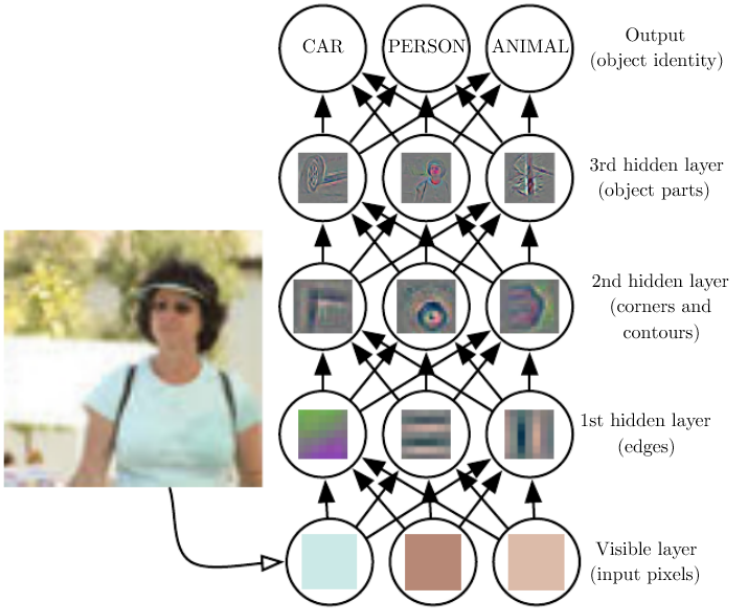


Figure 2.6: Deep Learning system to that represents the concept of an image of a person [13]

Batch Gradient Descent

Batch Gradient Descent computes the gradient of the cost function with respect to the parameters θ for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

In this algorithm the gradient is calculated to the entire dataset before performing one update, which can make the process slow for big datasets or even very difficult to do when the dataset does not fit in memory.

Stochastic gradient descent

Instead of computing the cost function for the whole dataset every time an update is done, Stochastic gradient descent (SGD) performs a parameter update for each training sample:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Where $x^{(i)}$ and $y^{(i)}$ are a sample of the training set. While **batch gradient descent** has redundancy problems (the cost functions need to be recomputed every time there is an update), SGD does away with this redundancy by performing an update at a time. This approach solves the problem where the dataset is too large to fit in memory and, in addition to this, has a faster convergence,

since since it avoids the computation of similar values on each parameter update. Nevertheless, it can lead to some fluctuations on the minimization of the objective function.

Mini-batch gradient descent

Mini-batch gradient descent takes the best of both previous approavhes and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Where $x^{(i:i+n)}$ and $y^{(i:i+n)}$ are elements from a batch with size n This approach has two major advantages:

- Reduces the variance of the parameter updates, which can lead to more stable convergence
- can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient with respect to a mini batch very efficient.

By choosing a small batch of data, this approach reduces the fluctuations of the parameter updates in comparison to the updates from the stochastic approach. An *epoch* is a forward pass and a backward pass of all training examples. In other words, an *epoch* is complete as soon as all the examples in the dataset are processed. On the other hand, an iteration is completed after each mini-batch is processed. The number of iterations in an *epoch* depends on the size of the mini-batch.

In the case of neural networks, gradient descent is used to find the optimal value for each weight, since its optimal value is at the global minimum, which sometimes cannot be satisfied because gradient descent can get stuck in a local minimum, being one of its limitations. Choosing a learning rate can help with this issue. The **learning rate** tells the optimizer how far to move the weights in the direction of the gradient for a mini-batch. If the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny. On the other hand, if the learning rate is hight, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse.

There are multiple ways to select a good starting point for the learning rate. A naive approach is to try different values and see which one gives you the best loss without sacrificing speed of training. When we start with a large learning rate, the loss does not improve and probably even grows while we run the first few iterations of training. When training with a smaller learning rate, at some point the value of the loss function starts decreasing in the first few iterations. This learning rate is the maximum we can use, any higher value doest not let the training converge. Even this value is too high: it won't be good enough to train for multiple epochs because over time the network will require more fine-grained weight updated.

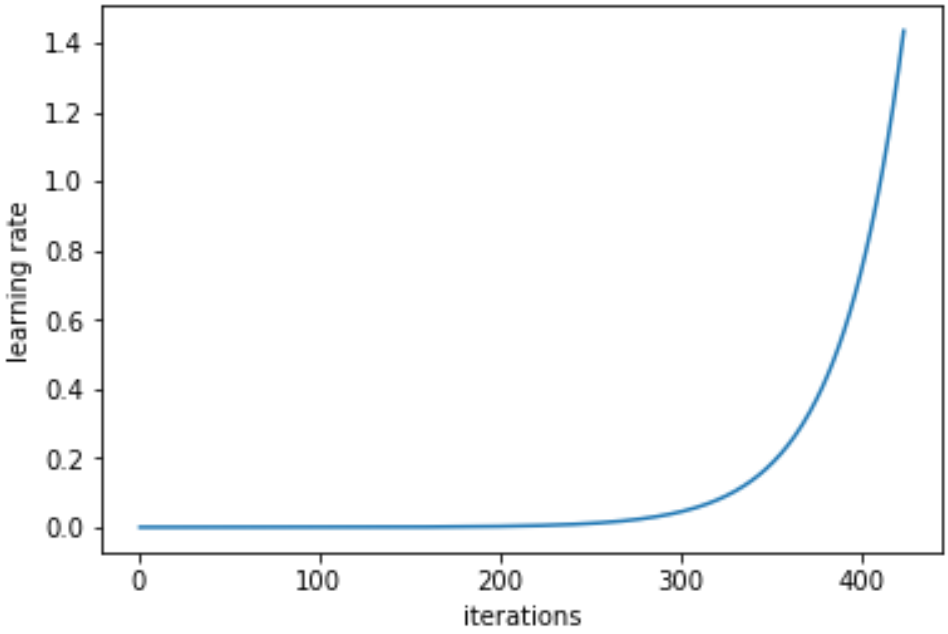


Figure 2.7: Learning rate increases after each mini-batch[6]

To ease the learning rate choose, Leslie N. Smith describes a powerful technique to select a range of learning rates for a neural network [33]. The idea is to train a network starting from a low learning rate and increase the learning rate exponentially for every batch (Figure 2.7).

Record the learning rate and training loss for every batch and then, plot the loss and the learning rate. Typically, it looks like this:

First, with low learning rates, the loss improves slowly, then training accelerates until the learning rate becomes too large and loss goes up: the training process diverges.

We need to select a point in the graph with the fastest decrease in the loss. In this example, the loss function decreases fast when the learning rate is between 0.001 and 0.01.

Selecting a starting value for the learning rate is just one part of the problem. Another thing to optimize is the learning schedule: how the learning rate evolves and changes during training. Learning rate schedules [35] practically eliminates the need to experimentally find the best values and schedule for the global learning rates. Instead of monotonically decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundaries values. Training with cyclical learning rates instead of fixed values achieves improved classification accuracy without a need to tune and often in fewer iterations.

In sum, backpropagation tries to find the minimum of the error function in the weight space using gradient descent. The general steps of the algorithm in a neural network the following:

1. Initialize network weights and biases

[6] https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

(a) SGD without momentum (b) SGD with momentum

Figure 2.8: SGD with and without momentum[7]

2. Weights are propagated forward through the network
3. The output error is calculated
4. Compute hidden and input layers weights by calculating the partial derivative of the error function with respect to the given weight
5. Update network weights by multiplying the negative of the computed partial derivatives with the learning rate
6. Repeat until stop condition is met

Gradient Descent Optimization Algorithms

In the following, we will provide a comprehensive overview of some algorithms that are widely used by the deep learning community to deal with the aforementioned challenges. Algorithms that are infeasible to compute (e.g Newton's method) are not presented.

Momentum

Stochastic Gradient Descent has trouble navigating areas where the surface curves much more steeply in one dimension than in another [36], which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum as in 2.8.

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Figure 2.8b, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum as in Figure 2.8a. It does by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal

[7] https://www.willamette.edu/ gorr/classes/cs449/momrate.html

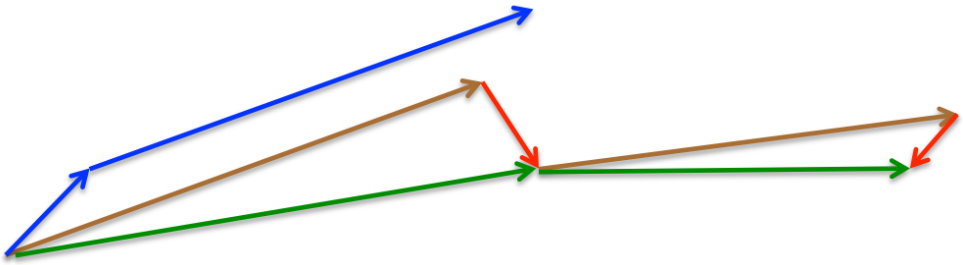


Figure 2.9: Nesterov Update Vector[8]

velocity if there is air resistance, i.e. $\gamma < 1$). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduces oscillation.

Nesterov accelerated gradient

However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We would like to have a smarter ball, a ball that has a notion of where it is going so it knows to slow down before the hill slopes up again. In the case of momentum, when reaching towards a minimum, momentum is often high and it does not slow down causing it to miss the minimum entirely and goind further to a not so good solution. In order to solve that issue, Nesterov accelerated gradient [25] was created.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

Instead of using gradient at the current location and then taking a big step in the direction of momentum, it first takes a bit step in the direction of the accumulated gradient and then makes a correction based on the gradient(Figure 2.9.

While Momentum first computes the current gradient (small blue vector on Figure 2.9), and then takes a big jump in the direction of the updated accumulated gradient (big blue vector), Nesterov approach first makes a big jump towards the previous accumulated gradient (brown vector), measures the gradient and then makes a correction (red vector), which results in the complete NAG update (green vector). Anticipating an updates helps us preventing going too fast and results in increased responsiveness, which has significantly increased the performance of Deep Learning methods, such as RNNs, on a number of tasks. Dean *et al* [7] have found that Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google, which

[8] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

- among other things - learned to recognize cats in Youtube videos [5]. Another research, by Pennigton *et al* used Adagrad to train *GloVe* [29] word embeddings, as infrequent words require much larget updates than frequent ones.

Adagrad

Adagrad [8] is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates. This is very useful for sparse data where features that are highly informative are not very abundant in the training data. When high informative features appear in the training set, they are weighted equally compared to features that are a lot more present in training samples and are not very informative.

The solution purposed by Adagrad is to increase the learning rate for more sparse parameters and decrease for less sparse ones. Previous we performed an update for all parameters θ at once as every parameter θ_i used the same learning rate η. As Adagrad uses a different learning rate for every parameter θ_i at every time step t, we first show Adagrad's per-parameter update, which we then vectorize. For brevity, we use g_t to denote the gradient at time step t. $g_{t,i}$ is then the partial derivative of the objective function with respect to the parameter θ_i at time step t:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

The SGD update for every parameter θ_i at each time step t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Adagard changes the learning rate η at each step t for every parameter θ_i based on the gradients previously computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\varepsilon}} \odot g_t$$

G_t represents a diagonal matrix that contains the sum of the squares of the past gradients of each θ_i and ε a small value, normally 10^{-8}, to avoid division by 0.

Adagard's main weakness is its accumulation of the sum of the squares of the past gradients. Since this sum is referred to the denominator, it makes the learning rate to shrink until it becomes infinitesimally small which makes the algorithm unable to learn additional knowledge. The following algorithms aim to resolve this flaw.

ADAM

Adaptative Moment Estimation(Adam) [20] is a another algorithm that computes adaptive learning rates for each parameter. ADAM is a generalization of AdaGrad. The update rule for Adam is based on the estimation of first (gradient mean) and second (uncentered variance) order moments

of past gradients. Previously, we compared Momentum as a ball that is thrown down a hill and its problems with slowing down to reach the optimal minimum. Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. We compute the decaying averages of past and past sum of the squared gradients m_1 and v_t respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t and $v:t$ are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, they are biased towards zero, especially during the initial time steps and when the decay rates are small, β_1 and β_2 are close to 1.

To handle these biases it is computed an estimate for first and second moment:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

As we said before, the update rule for ADAM is based on the estimation of the gradient mean and the uncentered variance. So, after we calculate both we just replace the function to update the parameters with the results:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

With these approach we get the correction bias to update parameters and we prevent the denominator to go infinitesimal small and it is possible to continue acquiring knowledge.

Cost functions

Cost functions can be viewed as the loss, or error of the model. Basically, they are the measure that determines how far is the actual result from the correct result. Backpropagation aims to minimize this function in order to infer the right weights. Here, we provide some of the most common cost functions:

Mean Squared Error

$$C_{MSE}(W,B,S^r,E^r) = 0.5\sum_j (a_j^L - E_j^r)^2$$

- W is the neural network's weights
- B is the neural network's biases

- S^r is the input of a single training sample
- E_j^r is the desired output of that training sample
- a_j^L is the activation value of the j^{th} neuron in the L layer

The MSE function measures the average of the squares of the errors, that is, the difference between the output value we got and what we estimated. Also known as quadratic error, maximum likelihood, and sum squared error.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C_{MSE} = (a^L - E^r)$$

Cross-entropy cost

Also known as Bernoulli *negative log-likelihood* and *Binary Cross-Entropy*

$$C_{CE}(W,B,S^r,E^r) = -\sum_j [E_j^r \ln\ a_j^L + (1-E_j^r)\ln\ (1-a_j^L)]$$

- W is the neural network's weights
- B is the neural network's biases
- S^r is the input of a single training sample
- E_j^r is the desired output of that training sample
- a_j^L is the activation value of the j^{th} neuron in the L layer

Normally used when the data is normalized as its results are bounded between 0 and 1 and can be represented as probabilities.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C_{CE} = \frac{(a^L - E^r)}{(1-a^L)(a^L)}$$

Exponential cost

This requires choosing some parameter τ that you think will give you the behavior you want. Typically it is needed to play with this variable until things work correctly.

$$C_{EXP}(W,B,S^r,E^r) = \tau \exp(\tfrac{1}{\tau}\sum_j (a_j^L - E_j^r)^2)$$

- W is the neural network's weights

- B is the neural network's biases
- S^r is the input of a single training sample
- E_j^r is the desired output of that training sample
- a_j^L is the activation value of the j^{th} neuron in the L layer
- $\exp x$ is simply shorthand for e^x

The gradient of this cost function with respect to the output of a neural network and some sample r:

$$C_{EXP}(W,B,S^r,E^r) = \tau \ \exp(\tfrac{1}{\tau}\sum_j (a_j^L - E_j^r)^2)$$

Hellinger Distance

$$C_{HD}(W,B,S^r,E^r) = \tfrac{1}{\sqrt{2}}\sum_j (\sqrt{a_j^L} - \sqrt{E_j^r})^2$$

- W is the neural network's weights
- B is the neural network's biases
- S^r is the input of a single training sample
- E_j^r is the desired output of that training sample
- a_j^L is the activation value of the j^{th} neuron in the L layer

This function needs to have positive values, and ideally values between **0** and **1**. The same is true for the following divergences.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$C_{HD}(W,B,S^r,E^r) = \tfrac{1}{\sqrt{2}}\sum_j (\sqrt{a_j^L} - \sqrt{E_j^r})^2$$

Kullback–Leibler divergence

Kullback–Leibler divergence is typically denoted:

$$D_{\mathrm{KL}}(P\|Q) = \textstyle\sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

$D_{\mathrm{KL}}(P\|Q)$ is a measure of information lost when Q is used to approximate P. Thus we want to set $P = E^i$ and $Q = \alpha^L$, because we want to measure how much information is lost when we use a_j^i to approximate E_j^i. This gives us:

$$C_{KL}(W,B,S^r,E^r) = \sum_j E_j^r \log \frac{E_j^r}{a_j^L}$$

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{E^r}{a^L}$$

Activation Functions

For any hidden layer to provide any useful information we need to use a non-linear activation function, otherwise it does not matter how deep the network is, the results will always yield a linear transformation, which won't produce any useful information given the non-linearity in real-world problems. Here we list some of the commonly used activation functions in this field.

Sigmoid

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

The sigmoid function is vastly used as an activation function. One of the reasons for it is that the derivatives needed for the gradient descent are easy to calculate. However, there are some downsides of using this function, one of them is the vanishing gradient, towards the end the gradient becomes exponentially smaller in the early layer which makes the network refuse to learn further or the learning process is drastically slow (depending on use case and until gradient/computation gets hit by floating point value limits). The output of the sigmoid function is always going to be in range $(0,1)$ compared to $(-\infty,+\infty)$ of linear functions. Having the activations bound avoids bias in the gradients.

Tanh

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

Tanh is also a widely used activation function that is quite similar to the sigmoid. However, this function's derivatives are steeper than sigmoid's so it has stronger gradients. It also has a greater range ($[-1,1]$) than the sigmoid's ($[0,1]$), avoiding bias in the gradients. As well as the sigmoid function, it also has the downside of vanishing gradient, due to the fact that we still need to calculate exponentials. Chossing between sigmoid function and tanh depends on the requirements we have for gradient strength.

ReLU

Rectified Linear Unit (ReLU) is the most popular non-linear function[13] which is simply the half-wave rectifier:

$$f(z) = max(z, 0)$$

In past decades, neural nets used smoother non-linearities, such as **Tanh** or **sigmoid**, but the ReLU typically learns much faster in networks with many layers, allowing training of a deep supervised network without unsupervised pre-training. ReLU is not bounded, it translates any negative input to 0 and all positive values are kept. The reason for ReLU's popularity is that it is fast, and doest not have problems with the vanishing gradient. that the sigmoid and Tanh have. However, ReLU has a problem usually called *dying RelU problem*. This occurs because the gradient can go towards 0. For activations in that region of ReLU, gradient will be 0 because of which the eights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/input (simply because gradient is 0, nothing changes). This problem can cause several neurons to just die and not respond making a substantial part of the network passive, unresponsive. There are variations of ReLU that mitigate this problem, such as *Leaky ReLU* or *Exponential ReLU*. Instead of translating a value to 0 if it is negative we can modify the flat side of the function for it to have a gradient and give the neuron a chance to recover.

LeakyReLU

$$\begin{cases} x & if\ x > 0 \\ 0.01x & if\ x \leq 0 \end{cases}$$

ELU

$$\begin{cases} x & if\ x > 0 \\ \alpha(e^x - 1) & if\ x \leq 0 \end{cases}$$

Softmax

$$S_j = \frac{e^{Z_i}}{\sum_{K=1}^{N} e^{a_k}} \forall j \in 1..N$$

The softmax function is mainly used as the activation function of an output layer using cross-entropy loss for multi-class classification. It takes an N dimensional vector of real values and produces another N dimensional vector of values in the range 0 to 1 that can be interpreted as the probability of a given input being of the given class.

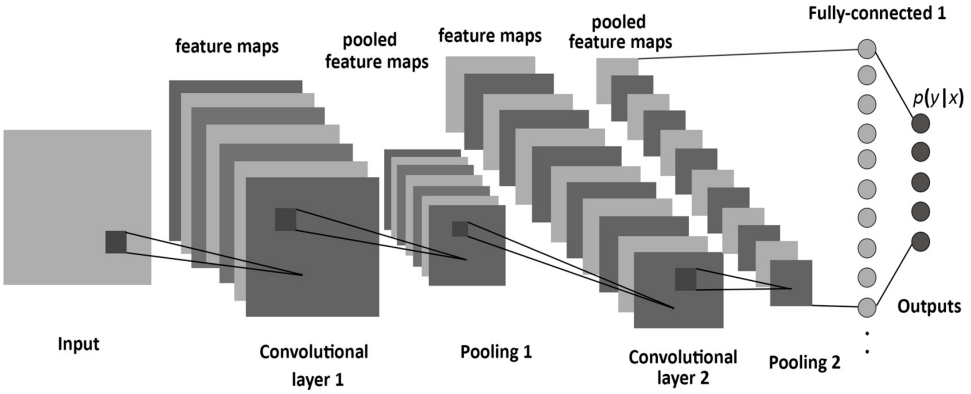


Figure 2.10: The structure of a CNN, consisting of convolutional, pooling and fully-connected layers[9]

2.3.1 Convolutional Neural Networks

When we feed our neural network, it is generally recommended to train and run the models on raw inputs, without manually extracting features. The reason for this is that a network trained is able to extract features on its own, but in contrast to working with prebuilt features, it would also be able to further optimize the feature extraction as the network improves. By *manually* extracting features, we might lose information due to some negligence and the neural network would be compromised since the beginning. If our input is an image, it would, therefore, be desirable to work with raw pixel values. One image consists in many of pixels and each pixel is possibly represented by multiple color values which makes the representation of that image in the input layer a really complex task. For example, an image with 1920x1080 pixels would require an input layer with 6 220 800 neurons (1920 x 1080 x 3). If we use a fully connected layer/architecure each neuron would be connected to each neuron on the subsequent layer. If the first fully connected layer contains 1000 neurons, the total number of parameters would amount to over six billion. Since the network has to optimize all these parameters, the training process could become very slow and storage intensive [2]. In order to mitigate this problem, a different kind of architecture is used, called *Convolutional Neural Networks (CNN)*. CNNs are mainly used to work with images. The reason it happens is because the neurons of a layer are organized in three components, height, width and depth, just like the pixels present in an image where the depth component differentiates the different color values. In addition to that, CNNs introduce two new types of hidden layers that are not fully connected, instead they are only connected to a subset of neurons to prevent the aforementioned problem. In the following, it will be presented an overview about some important concepts that help to understand how CNNs work and why they are the state-of-art for image recognition.

CNNs are usually trained by propagation via Stochastic Gradient Descent (SGD) to find weights and biases that minimize certain loss function in order to map the arbitrary inputs to the targeted outputs as closely as possible[1].

Convolution layers

In order to deal with the problems that fully-connected layers faced when processing images, Convolutional Layers are used in CNNs instead. What separates a convolutional layer from a fully-connected layer one is that each neuron is only connected to a small subset of neurons in the previous layer, which is a square sized region across the height and width dimensions. The size of this square is a parameter named *Receptive Field*. Since the convolutions always perform the whole depth, there is no need to define an hyperparameter for that. The reason for this is that the depth dimension usually has information about the different colors of the image and it is usually necessary to combine them in order to retrieve useful information.

Neurons of the convolution operator can perceive some local patterns of past layer's yield. Since the patterns that are perceived ought to be independent of their position in the image, all neurons will be compelled to recognize the same pattern bu making all of them share one single arrangement of parameters. This idea is alluded to as *Parameter Sharing*. In order to now recognize multiple different features within one layer, it is required to have several Filters, where each filter is a group of neurons that recognize a certain pattern at different locations in the image. In the convolutional layer, the depth dimension is then specifying to which filter a given neuron belongs.

Additionally, convolution operations are performed across all depth values because neurons in a convolutional layer, which are stacked on top of other, should have their feature jointly considered in the next layer. Therefore, a neuron in a convolutional layer will be connected to $r * d$ neurons on the underlying layer, where r is the size of the receptive field and d is the depth of the previous layer. For example, if we apply convolution in the input layer of an RGB image with $r = 3$ x 3, each neuron of the layer, will be connected to 27 input neurons, consisting of a 3 x 3 square of pixels with three neurons per pixel, as shown on the Figure 2.11.

The number of convolutions being conducted is defined by another parameter, named *Stride* that determines how large is the gap between two scanned regions. Without using any further hyperparameters we would always perform fewer convolutions on inputs close to the borders. Adjusting the *Padding* hyperparameter can make that more even, as it adds an additional borer of 0 values around the original input. *Padding* also makes the convolutional result to have a certain width and height, e.g. making the output have the same size as the input. In the Figure 2.12 we can see an example of the *stride* and *padding* given a receptive field.

The output of a convolutional layer is given by the following equation:

$$out = \frac{in - receptive\,field + 2 * padding}{stride} + 1$$

Pooling Layer

The third kind of layer that aims to mitigate some of the CNN problems is the pooling layer. Similarly to the convolutional layer, neurons in the pooling later are connected to a square size

[9] https://towardsdatascience.com/how-to-teach-a-computer-to-see-with-convolutional-neural-networks-96c120827cd1

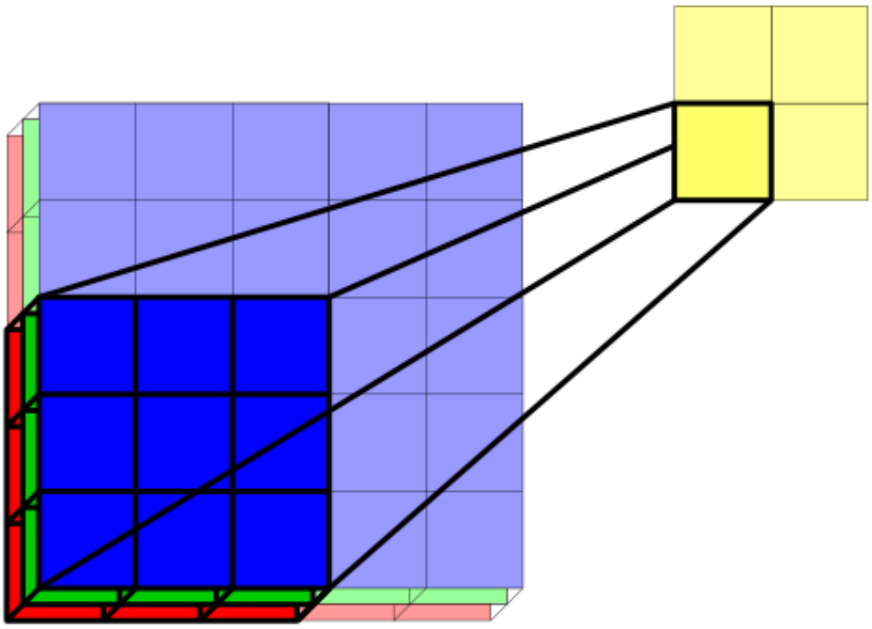


Figure 2.11: A neuron of a convolutional layer performing a convolution operation with a 3 x 3 receptive field on an RGB image [2]

region across the width and height dimensions of the previous layer. However, unlike the fully connected layer and the convolutional layer, the pooling layer is not parametrized. This means that neurons in the pooling layer do not have weights or biases that will be learned during the training process but instead perform some fixed functions on its inputs. Additionally, the pooling layer does not compute different depth values. Instead, the resulting pool layer will have the same depth as the previous layer and it will only combine local regions within a filter.

Max Pooling is an example of pooling. *Max pooling* is a **sample-based discretization process**. The objective is to down-sample an input representation reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. In *Max Pooling* the result of combining a number of neurons is the maximum value that any of them returned. Since all neurons in the convolutional layer can recognize the same patterns, this result can be interpreted as whether one pattern has been recognized in the pooling area or not, but the exact location will not be relevant anymore. An example of pooling can be seen in Figure 2.13

Max Pooling is typically used to prevent over-fitting by providing an abstracted form of the representation. Additionally, it helps mitigates the computational problem of CNNs by drastically reducing the number of parameters to learn.

2.3.2 ResNet

For conventional deep learning networks, convolutional layers are usually fully connected (FC) layers for classification task, without any skip/shortcut connection, in this dissertation we will

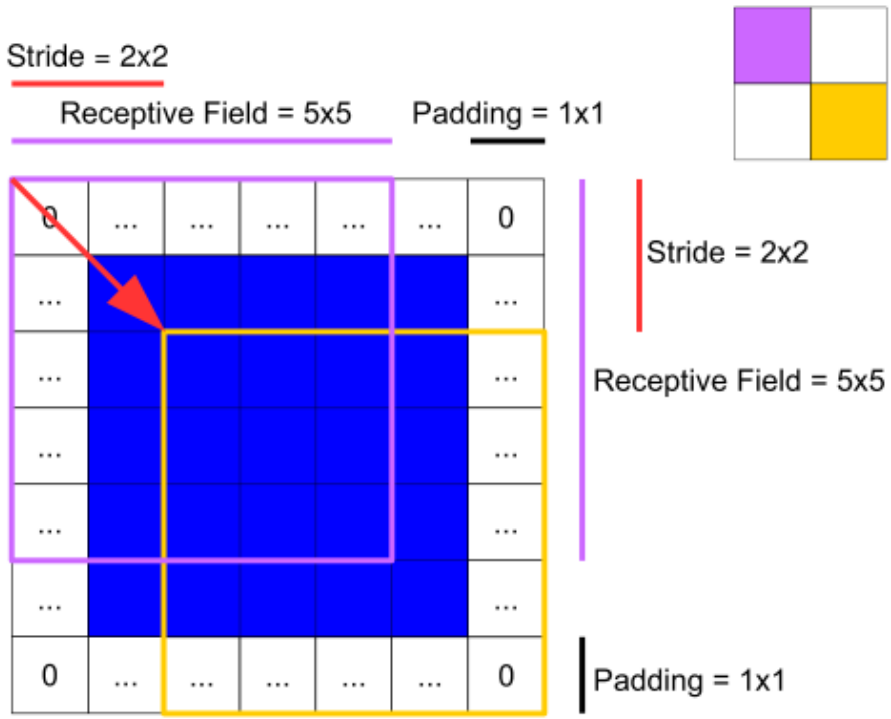


Figure 2.12: Receptive field, stride and padding [2]

name these NNs plain networks. When the plain network is deeper (layers are increased), the problem of vanishing/exploding gradients occurs [40].

We expect deeper networks will have a better accuracy due to the high number of features. However these deep conventional networks suffer from a **Degradation** phenomenon. During backpropagation, when partial derivative of the error function with respect to the current weight in each epoch of training, this has the effect of multiplying n of these small / large numbers to compute gradients of the "front" layers in an n-layer network. Degradation can be caused by two reasons:

- When the network is deep, and multiplying n of these small numbers will become zero (vanished).
- When the network is deep, and multiplying n of these large numbers will become too large (exploded).

To overcome this issue ResNet a skip/shortcut connection approach. The input x image is added after a few weight layers which means that even if we have a *vanishing gradient* we will still have the identity x to transfer back to earlier layers. In other words, identity mapping is proposed to promote the gradient propagation. Element-wise addition is used. Basically, a ResNet state is passed to another one. This kind of approach can ease the backpropagation using shortcuts

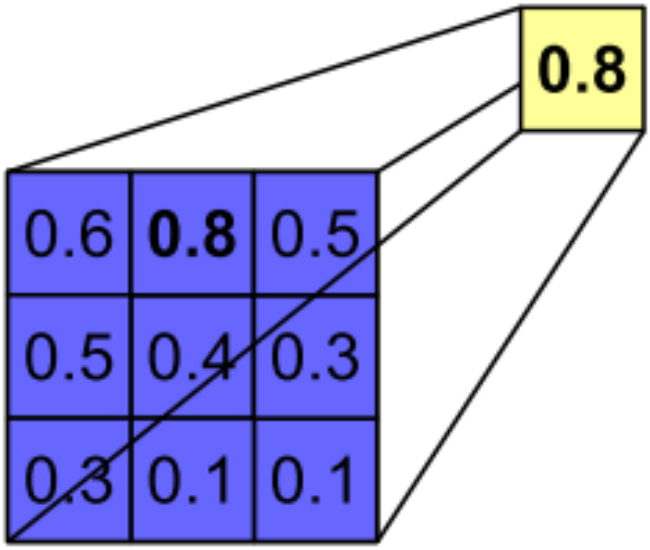


Figure 2.13: A neuron of a pooling layer performing a max pooling operation with a 3 x 3 receptive field on neurons of an underlying layer

and keeping the state of each module. An example of this skip/shortcut connection is shown on Figure 2.14.

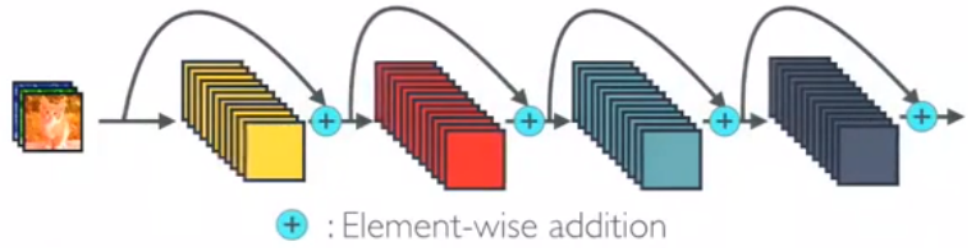


Figure 2.14: ResNet Concept[10]

2.3.2.1 ResNet Architecture

The figure 2.15 shows the ResNet architecture. The picture shows three different architectures. In the bottom, we have a *VGG19* neural network, which is a state-of-the-art approach in *ILSVRC 2014*. VGG19 is an improvement to standard CNNs by increasing the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3x3) convolution filters in all layers.

[10]https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

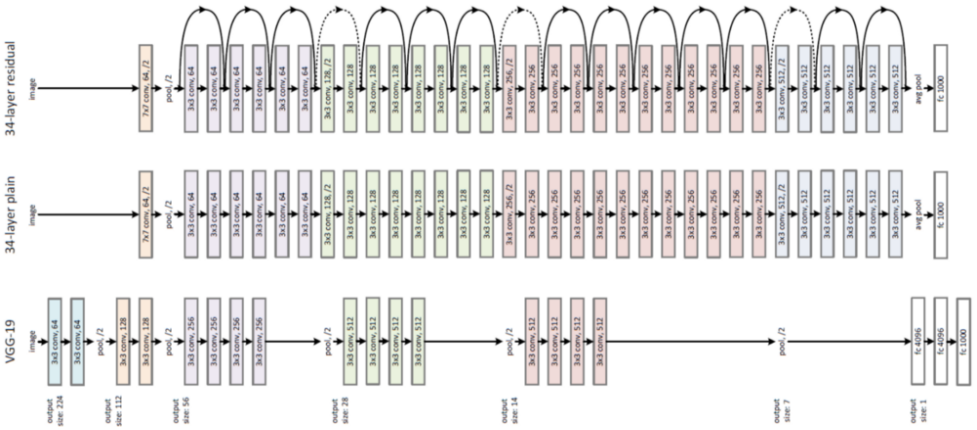


Figure 2.15: ResNet Architecture[10]

In the middle we have a 34-layer plain network is treated as the deeper network of VGG-19, i.e. more conv layers.

On the top we have 34-layer residual network (ResNet) is the plain one with addition of skip / shortcut connection.

For ResNet, there are 3 types of skip / shortcut connections when the input dimensions are smaller than the output dimensions.

1. Shortcut performs identity mapping, with extra zero padding for increasing dimensions. Thus, no extra parameters.

2. The projection shortcut is used for increasing dimensions only, the other shortcuts are identity. Extra parameters are needed.

3. All shortcuts are projections. Extra parameters are more than that of (2).

2.3.2.2 Bottleneck Design

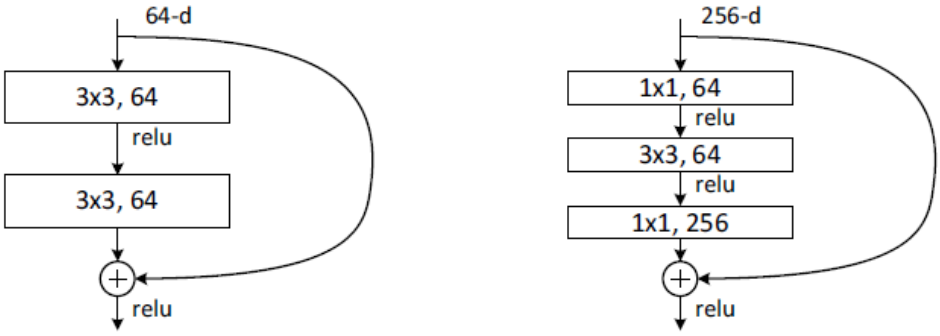


Figure 2.16: Bottleneck Design of Residual Neural Networks[10]

Since the network is very deep now, the time complexity is high. A bottleneck design is used to reduce the complexity as follows:

The 1x1 conv layers are added to the start and end of network as in the figure (right). This is a technique suggested in Network In Network and GoogLeNet (Inception-v1) [45]. 1x1 conv can reduce the number of connections (parameters) while not degrading the performance of the network so much.

With the bottleneck design, 34-layer ResNet become 50-layer ResNet. And there are deeper network with the bottleneck design: ResNet-101 and ResNet-152. The overall architecture for all network is shown in Figure 2.16.

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 2.17: ResNet overall architecture for all network[10]

2.3.3 DenseNet

Dense Convolutional Network (Densenet) is a model that connects each layer to every other layer in a feed-forward way. Traditional CNNs with L layers have L connections between each layer and its subsequent layer, Densenet has $L(L+1)$ direct connections, where $L+1$ is the subsequent layer. The features maps of all preceding layers are used as inputs for each layer and its own feature map is used as input in the subsequent layers. This approach has some compelling advantages: it reduces the vanishing-gradient impact, strengthen feature propagation and incentives feature reuse reducing the number of parameters [17].

Dense Block

In a Standard Convolutional Neural Network, the input image goes through multiple convolution and obrain high-level features (Figure 2.18) [39].

As we mentioned before, in **ResNet**, identity mapping is proposed to promote the gradient propagation. Element-wise addition is used. It can be viewed as algorithms with a state passed from one **ResNet** module to another one.

[11] https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

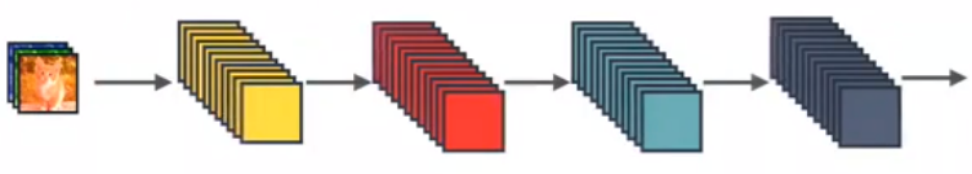


Figure 2.18: Standard Convolutional Neural Net Concept[11]

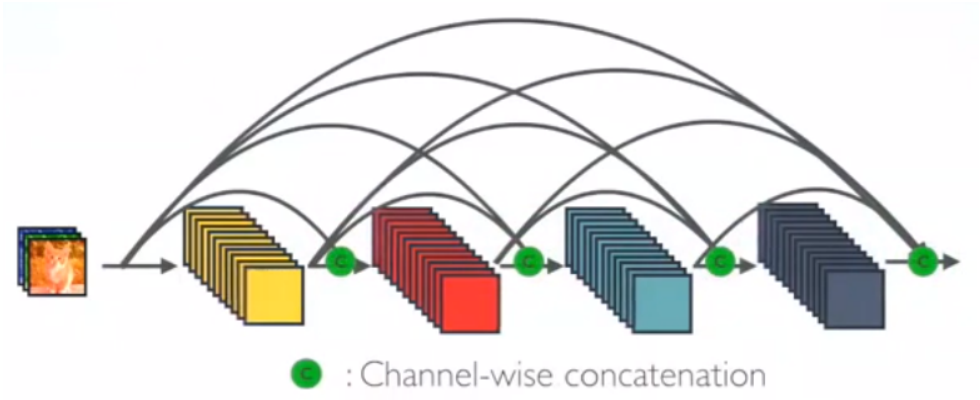


Figure 2.19: Standard Dense Block[11]

In **DenseNet**, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Concatenation is used. Each layer is receiving a "collective knowledge" from all preceding layers (Figure 2.19).

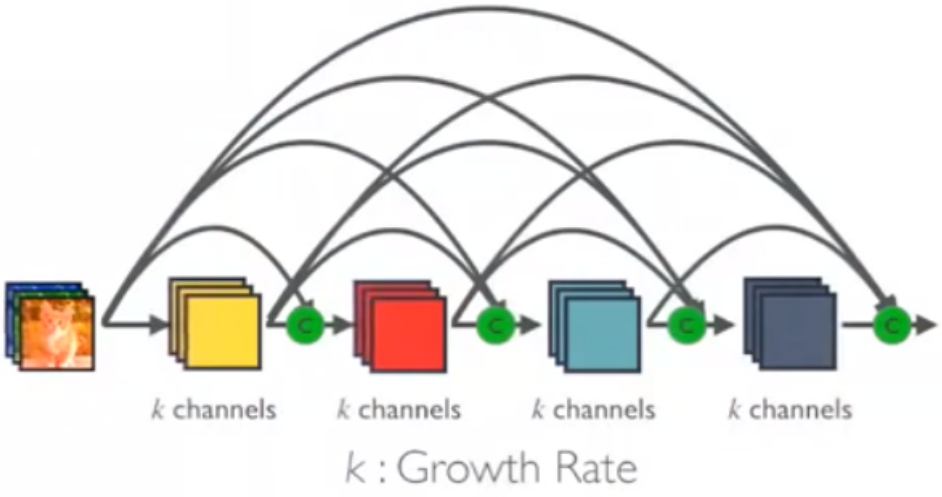


Figure 2.20: Dense Block with Growth Rate k[11]

Since each layer receives feature maps from all preceding layers, network can be thinner and more compact, i.e. we can have fewer channels which attenuates the overhead. The growth rate k

is the additional number of channels for each layer (Figure 2.20).

Therefore, it has higher computational efficiency and memory efficiency. Figure 2.21 shows the concept of concatenation during forward propagation.

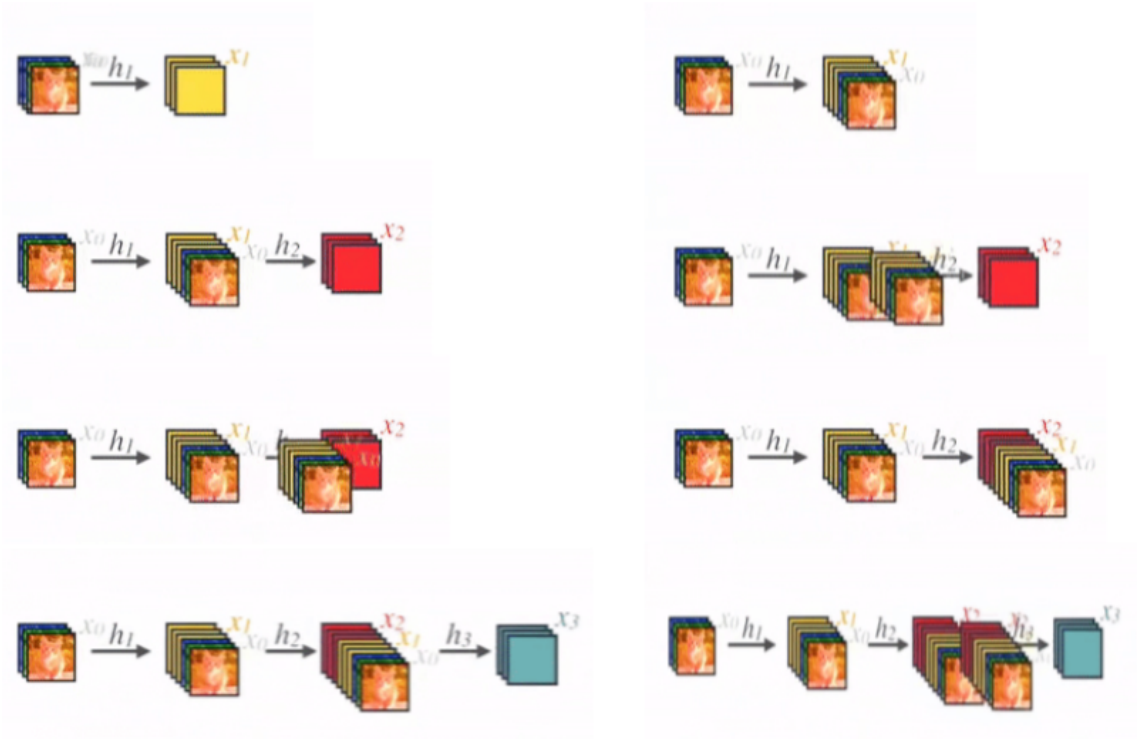


Figure 2.21: Concatenation during Forward Propagation. Each layer passes its own feature maps to the subsequent layer. The process is repeated by every layer.[11]

DenseNet Architecture

Basic DenseNet Composition Layer

For each composition layer, Pre-Activation **Batch Norm (BN)** and **ReLU**, then **3x3 Conv** are done with output feature maps of k channels, say for example, to transform $x0$, $x1$, $x2$, $x3$ to $x4$. This is the idea from Pre-Activation ResNet. An example of a composition layer is shown in Figure 2.22.

DenseNet-B (Bottleneck Layers)

To reduce the model complexity and size, **BN-ReLU-1x1 Conv** is done before **BN-ReLU-3x3 Conv** (Figure 2.23).

Multiple Dense Blocks with Transition Layers

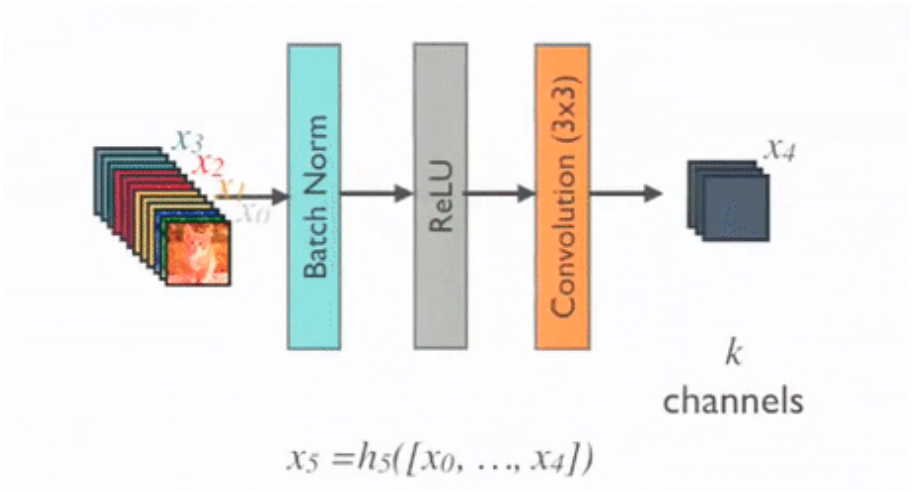


Figure 2.22: Composition Layer[11]

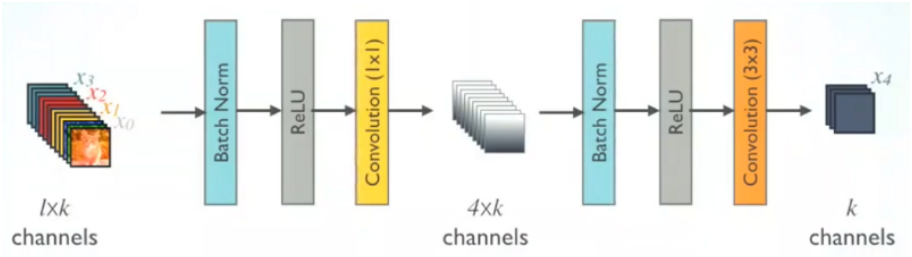


Figure 2.23: DenseNet-B[11]

1x1 Conv followed by 2x2 average pooling are used as the transition layers between two contiguous dense blocks. Feature map sizes are the same within the dense block so that they can be concatenated together easily. At the end of the last dense block, a global average pooling is performed and then a softmax classifier is attached. An example of multiple dense blocks with transition layers is shown in Figure 2.24.

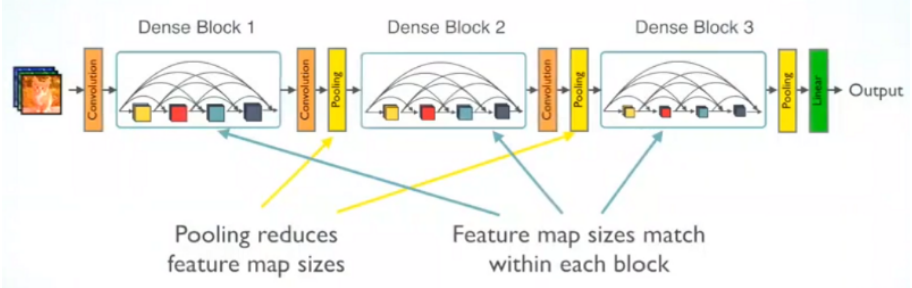


Figure 2.24: Multiple Dense Blocks[11]

DenseNet Advantages

DenseNets have a strong gradient flow. The error signal can be easily propagated to earlier layers more directly. This is a kind of implicit deep supervision as earlier layers can get direct supervision from the final classification layer (Figure 2.25).

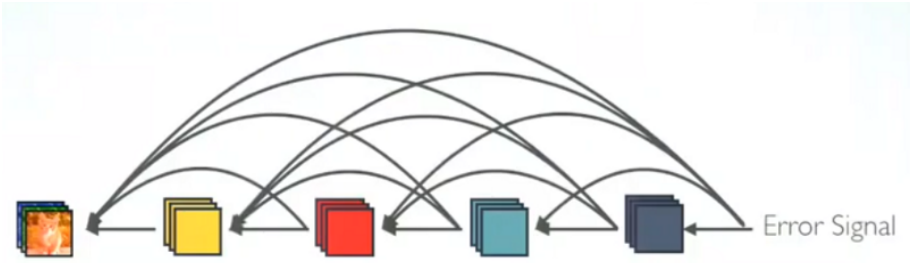


Figure 2.25: Implicit "Deep Supervision"[11]

DenseNet also brings Parameter & Computational Efficiency. For each layer, number of parameters in ResNet is directly proportional to CxC while the number of parameters in DenseNet is directly proportional to lxkxk.

Since $k << C$, DenseNet has much smaller size than ResNet. In Figure 2.26 is shown the difference between the number of parameters of ResNet and DenseNet.

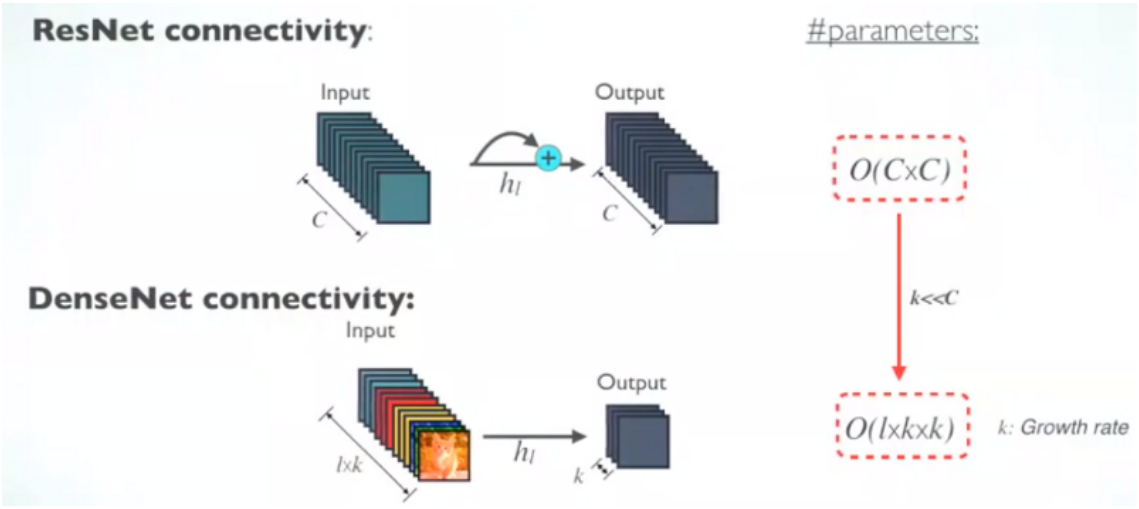


Figure 2.26: Number of Parameters for ResNet and DenseNet. Due to the concatenation during Forward Propagation, feature maps are reused and at the end of the last dense block, a global average pooling is performed with a *softmax* classifier reducing the output to the desired number of channels.[11]

DenseNet has more diversified features since each layer in DenseNet receive all preceding layers as input, more diversified features and tends to have richer patterns (Figure 2.27). Furthermore, DenseNet maintains low complexity features while in a standard CNN the classifier uses

high complex features since each convolution gets an higher abstraction from the features on the previous convolution.

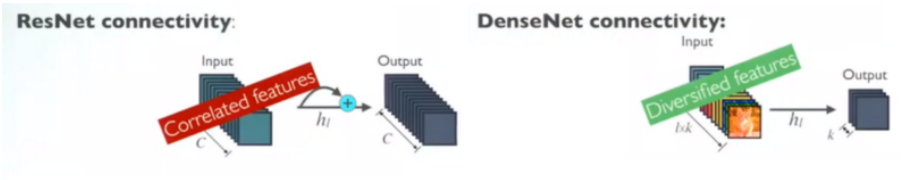


Figure 2.27: More Diversified Features. Densenet keeps high and low complex features which results in more diversity.[11]

In DenseNet, classifier uses features of all complexity levels. It tends to give more smooth decision boundaries. It also explains why DenseNet performs well when training data is insufficient (Figure 2.28).

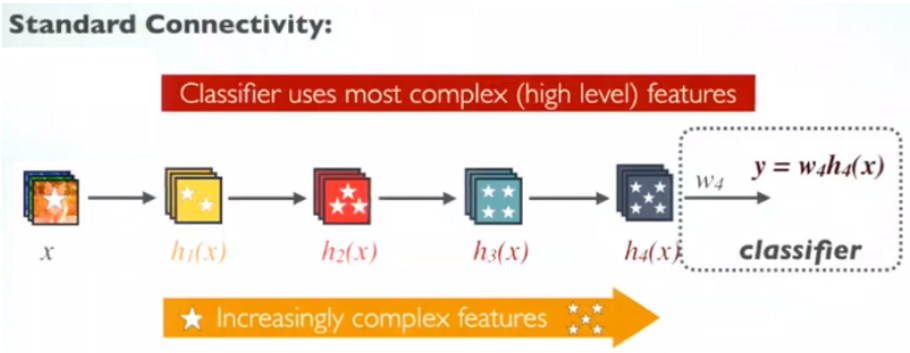


Figure 2.28: Standard ConvNet[11]

In Standard ConvNet, classifier uses most complex features (Figure 2.29).

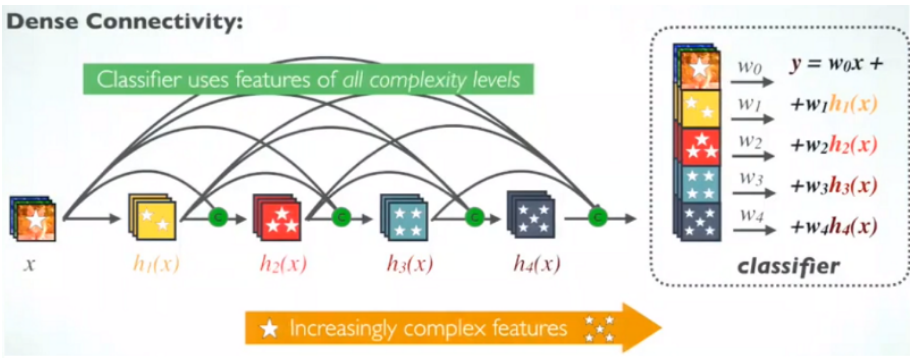


Figure 2.29: DenseNet overall architecture[11]

2.3.4 InceptionV3

The Inception deep convolutional architecture was introduced as GoogLeNet [37], formerly named Inception-v1. Later the Inception architecture was refined and improved in various ways, first by the introduction of batch normalization [19] and it was named Inception-v2. Later by additional factorization ideas in the third iteration [38] which is referred as InceptionV3.

The InceptionV3 model is composed by many modules. Modules are layers with a defined task. The most import modules in InceptionV3 are the **factorization** modules which aim to reduce the number of connections/parameters without jeopardizing the network efficiency.

Some of these factorization modules are presented below.

Factorization Into Smaller Convolutions

Two 3x3 convolutions replaces one 5x5 convolution as in Figure 2.30.

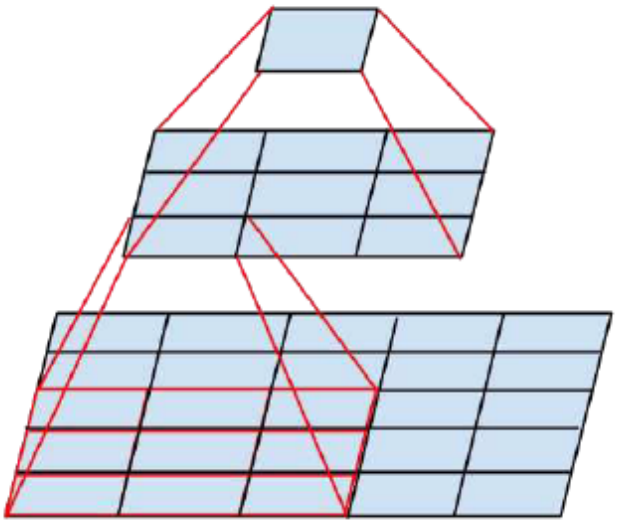


Figure 2.30: Two 3x3 convolutions replacing one 5x5 convolution[12]

By using 1 layer of 5x5 filter, number of parameters = 5x5=25 By using 2 layers of 3x3 filters, number of parameters = 3x3+3x3=18. Number of parameters is reduced by 28%

A similar technique was already presented in VGGNet [34]. With this technique, one of the new Inception modules (Inception Module A) can be seen in Figure 2.31.

Factorization Into Asymmetric Convolutions

One 3x1 convolution followed by one 1x3 convolution replaces one 3x3 convolution as shown in Figure 2.32.

[12] https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015

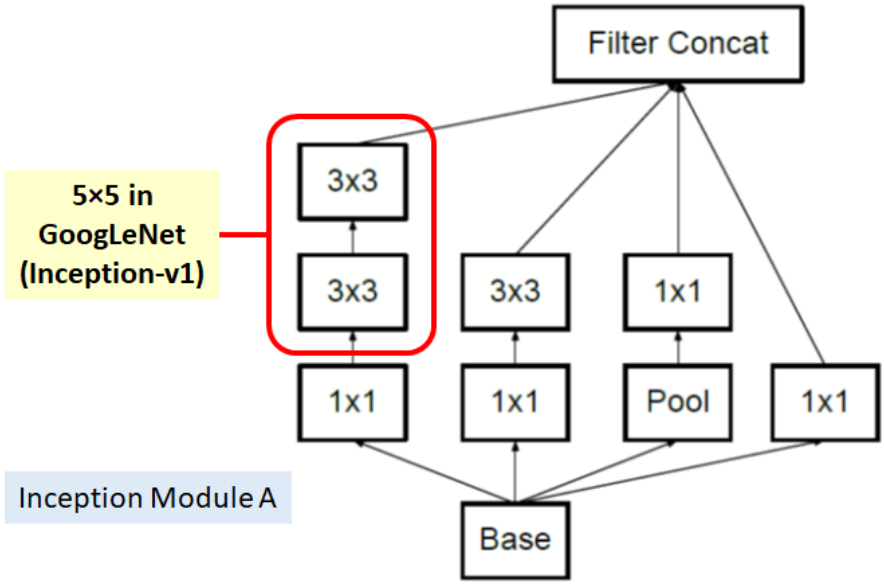


Figure 2.31: Inception Module A using factorization[12]

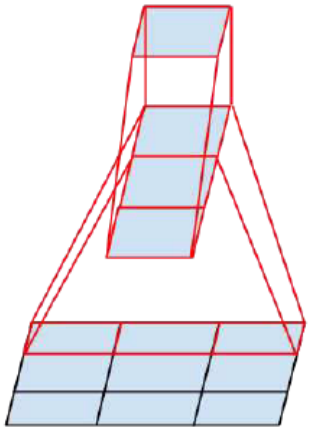


Figure 2.32: One 3x1 convolution followed by one 1x3 convolution replaces one 5x5 convolution[12]

By using 3x3 filter, number of parameters = 3x3=9 By using 3x1 and 1x3 filters, number of parameters = 3x1+1x3=6 Number of parameters is reduced by 33%

If we use two 2x2 filters, number of parameters = 2x2x2=8 Number of parameters is only reduced by 11

With this technique, a new module is created, **Module B**, shown in Figure 2.33

And Inception module C is created to promote high dimensional representations according to author descriptions as shown in Figure 2.34.

Auxiliary Classifier

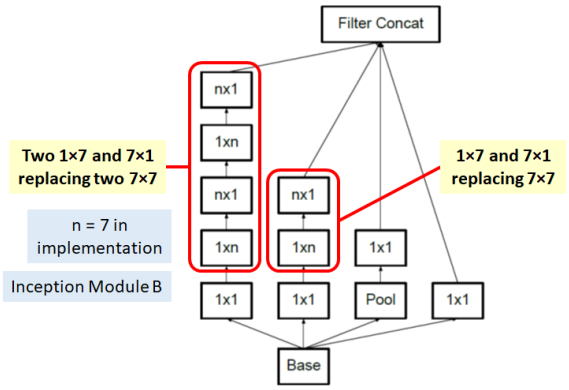


Figure 2.33: Inception Module B using asymmetric factorization[12]

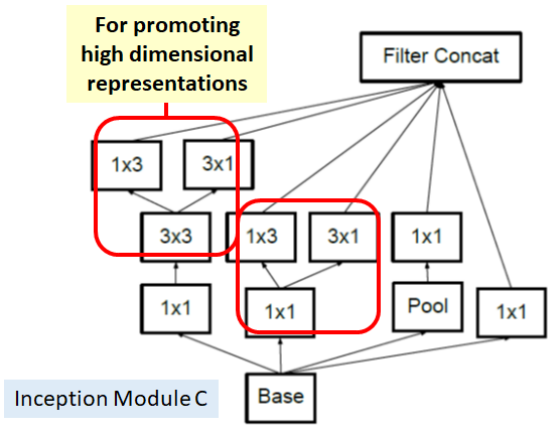


Figure 2.34: Inception Module C using asymmetric factorization[12]

Auxiliary Classifiers were already suggested in GoogLeNet / Inception-v1 [37]. There are some modifications in Inception-v3.

Only 1 auxiliary classifier is used on the top of the last 17x17 layer, instead of using 2 auxiliary classifiers. **Batch normalization**, suggested in Inception-v2 [19], is also used in the auxiliary classifier.framework based on decision tree algorithm, used for ranking and classification.

Efficient Grid Size Reduction

Conventionally, such as AlexNet and VGGNet, the feature map downsizing is done by max pooling. The drawback is either too greedy by max pooling followed by conv layer, or too expensive by conv layer followed by max pooling. Hence, an efficient grid size reduction is proposed and an example can be seen in Figure 2.36.

With this mechanism, 320 feature maps are done by conv with stride 2. 320 feature maps are computed by max pooling. These 2 sets of feature maps are concatenated as 640 feature maps and go to the next level of inception module.

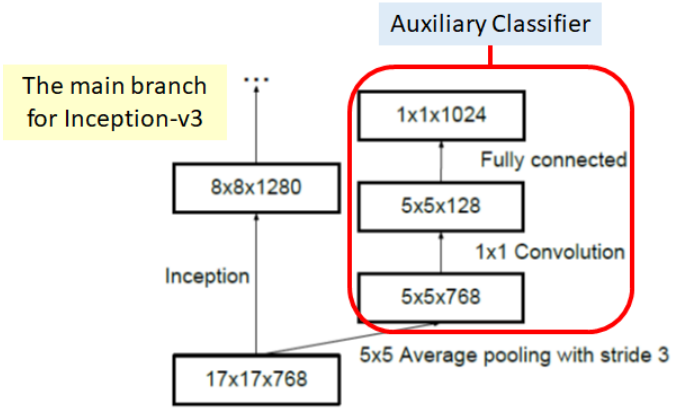


Figure 2.35: Auxiliary Classifier act as a regularization[12]

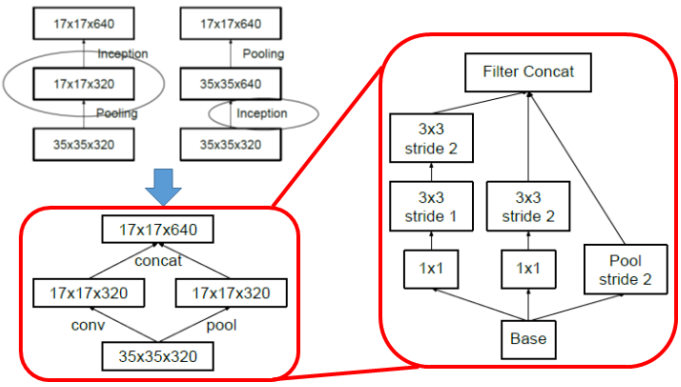


Figure 2.36: Conventional downsizing (Top Left), Efficient Grid Size Reduction (Bottom Left), Detailed Architecture of Efficient Grid Size Reduction (Right)[12]

Grid size reduction makes the model less expensive, i.e. less layers, and still efficient.

Overall architecture

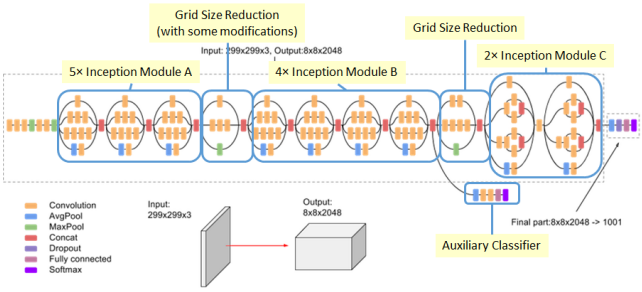


Figure 2.37: Inception-v3 Architecture (**Batch Norm** and **ReLU** are used after **Conv**)[12]

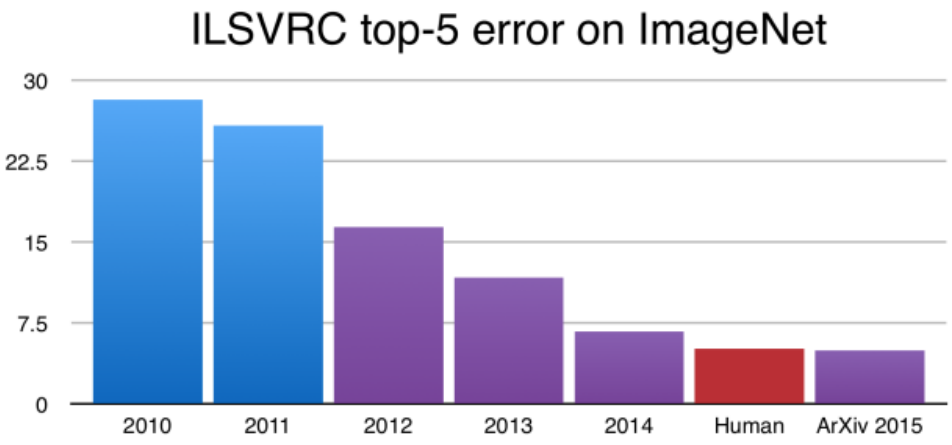


Figure 2.38: The *y*-axis is the error rate on ImageNet. In 2015 the error rate of humans is slightly higher than the CNN model presented[13]

With 42 layers deep, the computation cost is only about 2.5 higher than that of GoogLeNet [37] and much more efficient than that of VGGNet[34].

2.3.5 R-CNN

Since a group of researchers won the *ImageNet Large Scale Visual Recognition Challenge* in 2012 [21], Convolutional Neural Networks have become the gold standard for image classification. Since then, the growth of CNN architectures have improved in a way that in current ImageNet challenges, this methods outperform humans [12] as we can see in Figure 2.38.

R-CNN are Regions With CNNs. The goal of R-CNN is to take an image, and correctly identify where the main objects are (via a bounding box) in the image.

- **Inputs:** Image
- **Outputs:** Bounding Boxes + labers for each object in the image

The problem is to find the bounding boxes. R-CNN does what we might intuitively do as well - propose a bunch of boxes in the image and then see if any of them corresponds to the object that we are actually looking for. R-CNN creates these bounding boxes using a method called *Selective Search*. Selective Search results from the combination of the best of the intuitions of **segmentation** and **exhaustive search**. Like segmentation, Selective Search use the image structure to guide the sampling process. Like exhaustive search, it aims to capture all possible object locations [41]. At a high level, Selective Search (Figure 2.39) looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects.

[13] https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

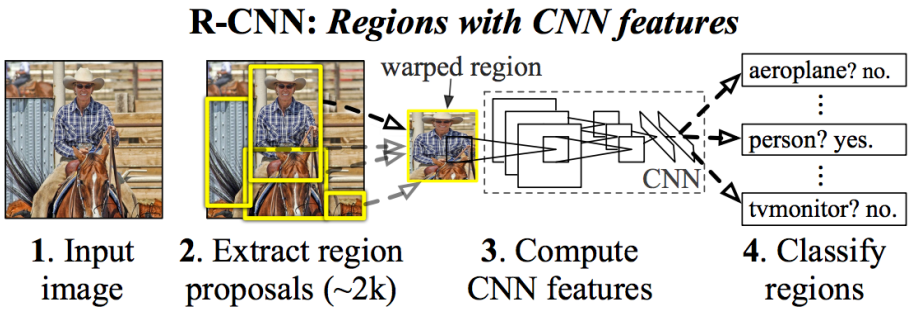


Figure 2.39: R-CNN architecture[13]

Once the region proposals are created, R-CNN warps the region to a standard square size and passes it through to a Convolutional Neural Network. This CNN is responsible for classification of the object. Therefore, on the final layer the CNN, R-CNN adds a Support Vector Machine (SVM) that simply classifies whether this is an object, and if so what object.

Improving the Bounding Boxes

When we find the object in an image, we can tight the box to fit the true dimensions of the object. This is the final step of a R-CNN. R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates to get our final result. Here are the inputs and outputs of this regression model:

- Inputs: sub-regions of the image corresponding to objects
- Outputs: New bounding box coordinates for the object in the sub-region.

So, to summarize the behaviour of R-CNN, we have:

1. Generate a set of proposals for bounding boxes
2. Run the images in the bounding boxes through a pre-trained CNN and finally an SVM to see what object the image in the box is.
3. Run the box through linear regression model to output tighter coordinates for the ox once the object has been classified

To summarize, the CNN model consists in three modules. The first generates category-independent region proposals. These proposals define the set of candidate detections available to the detector. The second module is a large convolutional neural network that extracts a fixed-length feature vector from each region. Finally, the third module is a set of class-specific linear SVM's. The last module depends on the classification method used. The main functionality of these algorithms is to analyze ROIs (Regions of Interest). Since those regions are identified, a standard classification algorithm can be used to identify different objects and label them accordingly [12].

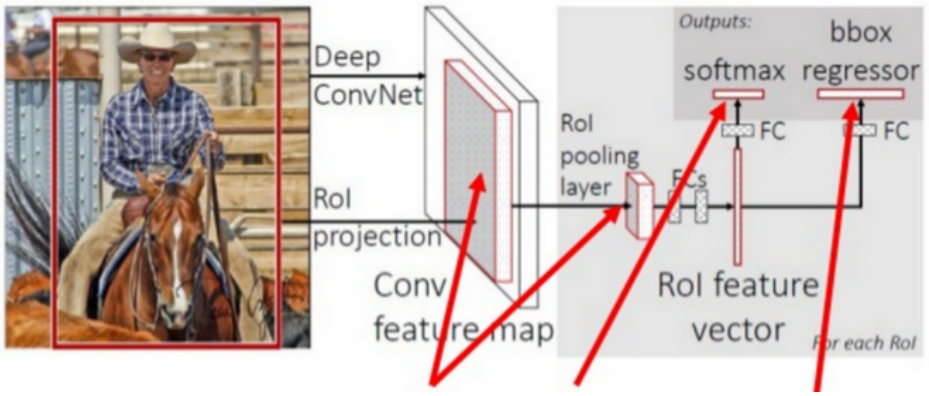


Figure 2.40: Fast R-CNN combined the CNN, classifer, and bounding box regressor into one, single network[13]

2.3.6 Fast R-CNN

Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy.

R-CNN performs quite slow for some circumstances:

1. It requires a forward pass of the CNN for every single region proposal for every single image

2. It has to train three different models separately - the CNN to generate image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train

In 2015, the first author of R-CNN, solve both of these problems, leading to a second algorithm, named **Fast R-CNN**.

For the forward pass of the CNN, the author realized that for each image, a lot of proposed regions for the image invariably overlapped causing the problem of having to run the same CNN computation all over again. The suggested approach was simple, run the CNN once per image and then find a way to share the computation across the set of proposals. The approach used was a *RoIPool (Regions of Interest Pooling*. *RoIPool* shares the forward pass of a CNN for an image across its subregions. In the Figure 2.40 , we can see how the CNN features for each region are obtained by selecting a corresponding region from the CNN's feature map. Then, the features in each region are pooled (max pooling). This way, all it takes is one pass of the of the image instead of all the passes that we would need to process for each region.

The difference between Fast R-CNN and CNN is, that for the latter we needed a classifier that worked independently from the *extractor*, therefore we needed two different models to process our input. Additionally, an extra model is needed to tight the bounding boxes (regressor). Fast R-CNN instead used a single network to compute all three methods. Fast R-CNN replaces the SVM classifier with a softmax layer on top of the CNN to output a classification. It also needs a

linear regression layer parallel to the softmax layer to output bounding box coordinates. This way all the outputs needed come from a single network. To summarize the Fast R-CNN model:

1. **Inputs**: Image with region proposals
2. **Outputs**: Object classifications of each region along with tighter bounding boxes[11].

2.3.7 Faster R-CNN

Even though Fast R-CNNs have promising results, there was still one remaining bottleneck - the region proposer. In Fast R-CNN and R-CNN we have generate a set of potential bounding boxes or ROIs (regions of interest) to test. In Fast R-CNN , these proposals were created using **Selective Search**, a fairly slow process that was found to be the bottleneck of the overall process [31].

The main idea of Faster R-CNN is to reuse the CNN results for region proposals instead of running a separate selective search algorithm. So instead of depending on features of the image that were already calculated, the algorithm can reuse the output of the pass of CNN (first step of classification). This kind of approach is illustrated in the Figure 2.41

In the image above we can see how a single CNN is used to both carry out region proposals and classification. This way, only one CNN needs to be trained and we get region proposals in the most simplistic way, in a faster and computationally friendly way.

The inputs and outputs of this model are the following:

- **Inputs**: Images (regions proposals are not needed)
- **Outputs**: Classifications and bounding box coordinates of objects in the images.

However, we have to take a look on how the regions are generated in Faster R-CNN. To generate regions of interest, Faster R-CNN adds a Fully Convolutional Network on top of the features of the CNN creating what is known as the **Region Proposal Network**. An example of this technique is illustrated on the Figure 2.42.

The **Region Proposal Network** works by passing a sliding window over the CNN feature map and at each window, outputting k potential bounding boxes and scores how accurate these boxes are expected to be. For that we create **anchor boxes**. Anchor boxes work like our intuitively way of seeing things. When we want to identify an human region, our first perception is to bound them into a rectangular region. Likewise, we know we won't see many boxes that are very thin. For each such anchor box, we output one bounding box and score per position in the image. As soon as we can recognize these aspect ratios and sizes we can easily identify promising regions and discard regions with no interest (Figure 2.43).

Having this in mind we can discriminate the inputs and outputs to this Region Proposal Network:

- **Inputs**: CNN Feature Map

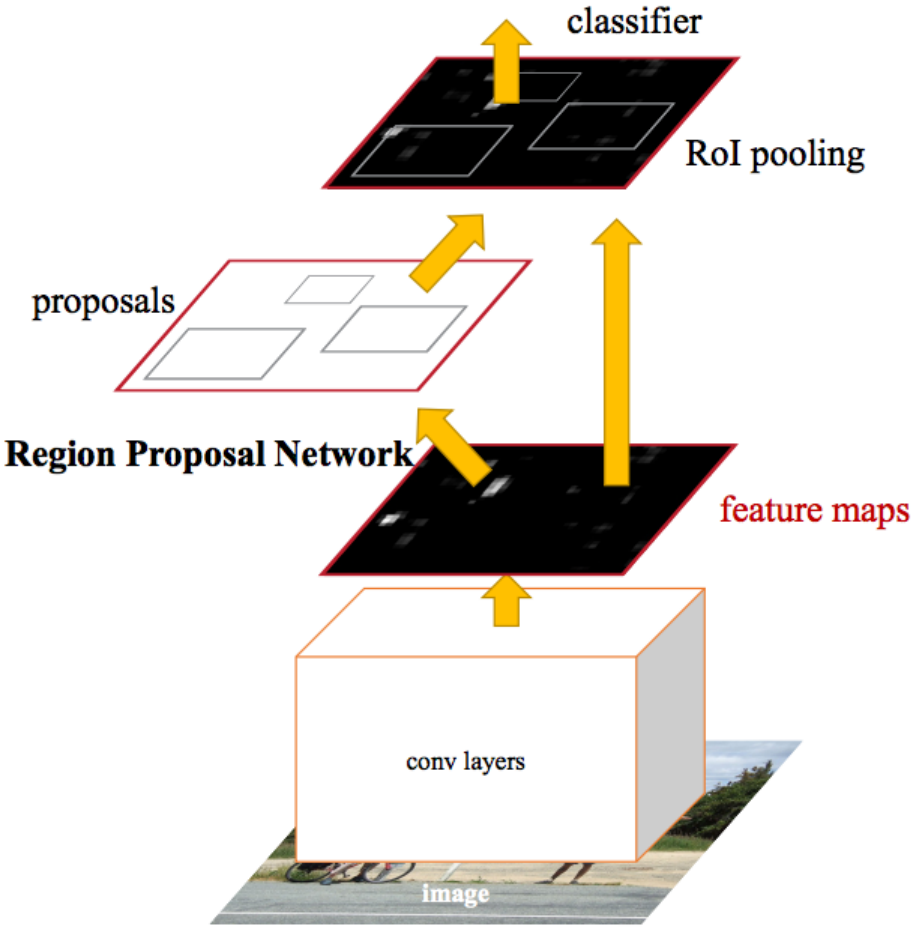


Figure 2.41: In Faster R-CNN is used for region proposals and classifications[13]

- **Outputs**: A bounding box per anchor. A score representing how likely the image in that bounding box will be an object.

To generate a classification, the bounding box is passed into a Fast R-CNN to generate a classification and tight the bounding boxes.

2.3.8 Mask R-CNN

The remaining question is: *Can we extend the previous techniques to go a little bit further and locate exact pixels instead of bounding boxes without compromising our image analysis and computational power?* This problem is known as image segmentation has been vastly explored by many researchers.

Given that Faster R-CNN works well with object recognition, one step forward would be to extend this recognition at pixel level [14].

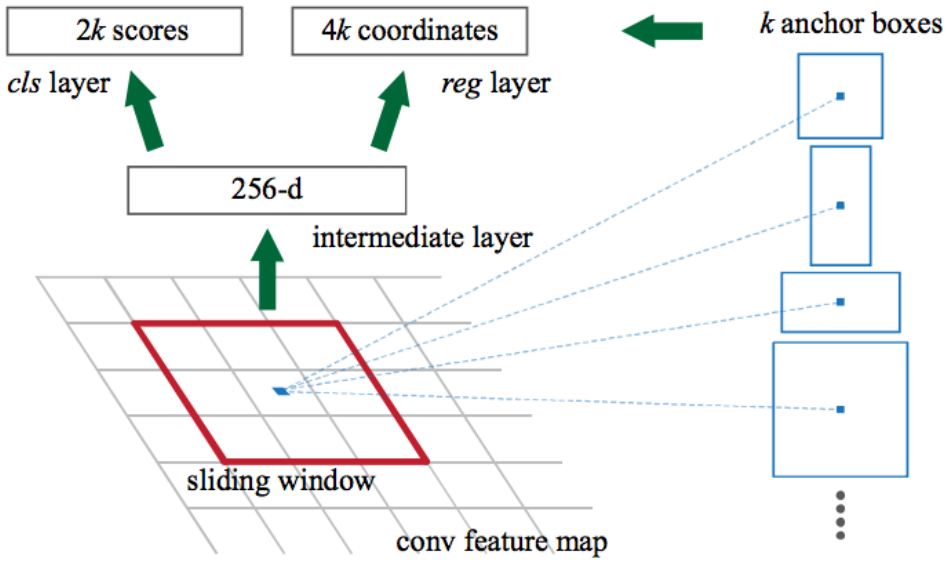


Figure 2.42: The Region Proposal Network slides a window over the features of the CNN. At each window location, the network outputs a core and a bounding box per anchor[13]

Mask R-CNN does this by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object. The branch is a Fully Convolutional Network on top of a CNN based feature map. The inputs and outputs are:

- **Inputs**: CNN Feature Map
- **Outputs**: Matrix with 1s on all locations where the pixel belongs to the object and 0s elsewhere (typically known as *binary mask*)

Mask R-CNN appeared due to some inaccuracy on the previous methods. Since either Faster R-CNN and Fast R-CNN rely on regions on the feature map selected by *RoIPool*, this can lead to inefficiency when we are analyzing the picture at the pixel level. To mitigate this problem a new approach for pooling was designed, called *RoIAlign*.

In the Figure 2.44 we have an image of size 128 x 128 and a feature map of 25 x 25. If we want the features corresponding to the 15 x 15 pixels in the original image (top left) that would be translated to $25/128$ pixels in the feature map. To select 15 pixels from the original image, we just select $15 * 25/128 = 2.93$ pixels. However, in *RoIPool* this value would be truncated and cause a slight misalignment. In *RoIAlign* is used bilinear interpolation to get a precise idea of what would be at pixel 2.93. This, at a high level, allows to avoid misalignments caused by *RoIPool*.

Once these masks are generated, Mask R-CNN combines them with the classifications and bounding boxes from Faster R-CNN to generate some precise and accurate segmentations, as we can see in Figure 2.45.

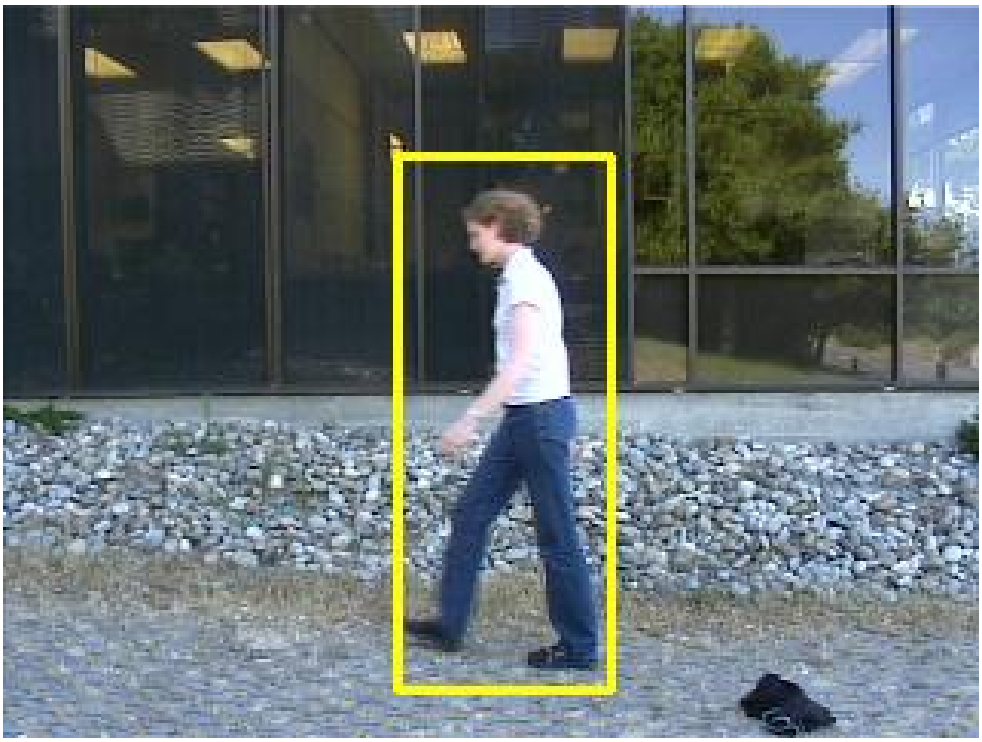


Figure 2.43: Regional Proposal Networks follow the same intuition of humans to recognize objects. We know that the bounding boxes for people tend to be rectangular and vertical. We can use this same principle to create an anchor of such dimensions[13]

2.3.9 U-Net

The typical use of convolutional networks is on classification tasks, where the output to an image is a single class label. However, in many visual tasks, especially in biomedical image processing, the desired output should include localization, i.e., a class label is supposed to be assigned to each pixel. On Mask R-CNN, a network is trained in a sliding-window approach to predict the class label of each pixel by providing a local region (patch) around that pixel as input.

The method has some drawbacks, such as:

- It is quite slow because the network must be run separately for each patch, and there is a lot of redundancy due to overlapping patches.
- There is a trade-off between localization accuracy and the use of context

The *U-net* approach follows a more elegant architecture, the so-called *fully convolutional network*. The *U-net* method modifies this architecture such that it works with very few training images and outputs more precise segmentations (Figure). The main idea of this method is to supplement a usual contracting network by successive layers, where pooling operators are substituted by upsampling operators. Hence, these layers increase the resolution of the output. In order to localize, the upsampled output is combined and aggregated with high resolution features from the

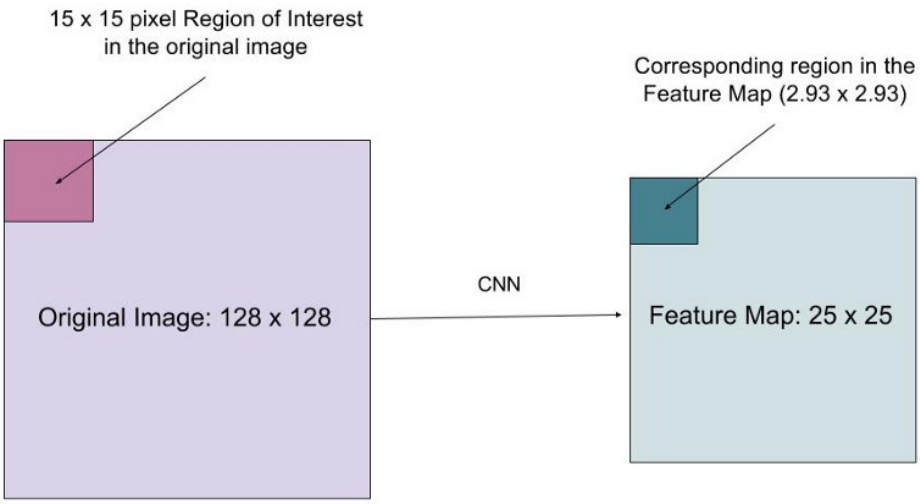


Figure 2.44: Mapping a region of interest onto a feature map[13]

contracting path. A successive convolution layer can then learn to assemble a more precise output based on this information.

The main difference from this approach to other conventional CNN architectures is that in the upsampling part, there are a large number of feature channels, which allows the neural network to propagate those features to higher resolution layers. The network does not have any fully connected layers and only uses the valid part of each convolution, i.e., the segmentation map only contains the pixels, for which the full context is available in the input image. The most important advantage of this approach when thinking about our project is the ability to predict the pixels in the border region of each patch since the missing context is extrapolated by mirroring the input image. This strategy allows the seamless segmentation of arbitrarily large images by an overlap-tile strategy (see Figure 2.46). This patch-based approach is very useful to process high quality images, since otherwise the resolution would be limited by the GPU memory[32].

In our context, detecting oncocytic cells in high resolution microscopic images, the main issue is how to deal with overlapping cells. Overlapped cells can't be analyzed as a whole since that each of them got a unique morphology and a set of fine-grained attributes that cannot be ignored. Following our method, the separation of touching objects can be solved by using a third channel that predicts the boundaries between cells. This is one of the advantages of U-Net is that it is possible to structure the network to output as many channels as we need and represent any class in any channel by using 1x1 convolution at the final layer. In our case, the output will be *background*, *nuclei* and *boundary*. An example of this approach can be seen in Figure 2.47.

U-Net architecture

In Figure 2.48 we have a possible *U-Net* architecture. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows a CNN architecture that consists

Figure 2.45: Mask R-CNN aggregates the best of the previous model by being able to segment and classify objects in an image[13]

of the repeated application of two 3 x 3 convolutions, each followed by a ReLU activation function and a 2x2 max pooling operation with stride 2 for down sampling. At each downsampling step the number of feature channels is doubled.

2.3.10 Capsule Networks

On the other hand, in the expansive path consists in a feature map that is upsampled followed by a 2x2 convolution that breaks the feature channels in half, a concatenation with the correspondingly cropped feature map from the contracting path and two 3x3 convolutions, followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution.

CNNs (convolutional neural networks) are one of the reasons deep learning is so popular today, they can do amazing things that people used to think computers would not be capable of doing for a long, long time. But they have their limits and some fundamental drawbacks and that is why Capsules neural networks are picking up pace, which introduce a new building block that can be used to overcome these limits & drawbacks of CNNs. Capsule networks (CapsNets) are a trendy new neural net architecture that may well have a profound impact on deep learning, in particular for computer vision.

A Capsule Neural Network (CapsNet) is a machine learning system that is a type of artificial neural network (ANN) that can be used to better model hierarchical relationships. The approach is an attempt to more closely mimic biological neural organization.

A CapsNet is composed of capsules rather than neurons. A capsule is a small group of neurons that learns to detect a particular object within a given region of the image, and it outputs a vector whose length represents the estimated probability that the object is present, and whose orientation

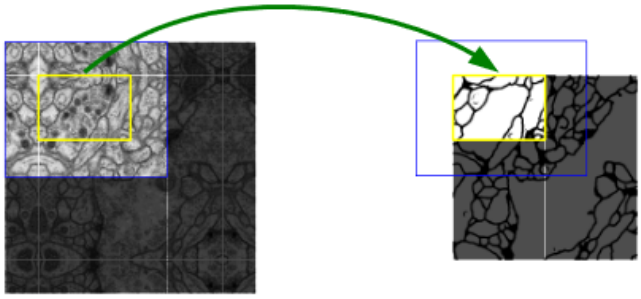


Figure 2.46: Overlap strategy for high resolution images [32]

encodes the object's pose parameters. If the object is changed slightly then the capsule will output a vector of the same length, but oriented slightly differently. Thus, capsules are equivariant.

Computer graphics deals with constructing a visual image from some internal hierarchical representation of geometric data, relationships between 3D objects can be represented by a so-called pose, which is in essence translation plus rotation. But how do we model these hierarchical relationships inside of a neural network? For a CNN, this task is really hard because it does not have this built-in understanding of 3D space, but there is type of neural network (CapsNet), for which it is much easier because these relationships are explicitly modeled. On Figure 2.49 we can see the main differences between a capsule and a traditional neuron.

CNNs perform exceptionally great when they are classifying images which are very close to the data set. If the images have rotation, tilt or any other different orientation then CNNs have poor performance. This problem was solved by adding different variations of the same image during training. Pooling layer helps in creating the positional invariance for CNN but what we needed was not invariance but equivariance. Equivariance makes CapsNet understand positional, orientational, proportional invariances. This leads us to the recent advancement of Capsule Networks [30]

A neuron receives input scalars from other neurons, then multiplies them by scalar weights and sums. This sum is then passed to one of the many possible nonlinear activation functions, that take the input scalar and output a scalar according to the function. That scalar will be the output of the neuron that will go as input to other neurons.

In essence, artificial neuron can be described by 3 steps:

- scalar weighting of input scalars
- sum of weighted input scalars
- scalar-to-scalar nonlinearity

while, the capsule has vector forms of the above 3 steps in addition to the new step, affine transform of input:

[14]https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-ii-how-capsules-work-153b6ade9f66

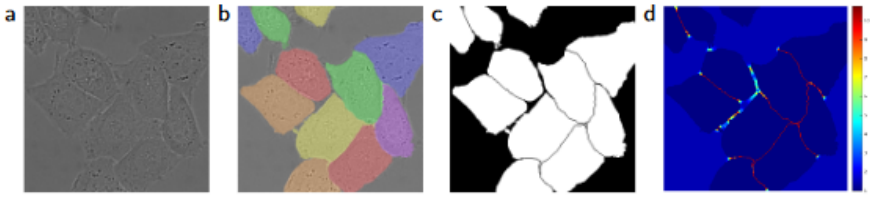


Figure 2.47: (a) raw image (b) overlay with ground truth segmentation. Different colors indicate different instances of cells. (c) generated segmentation mask (white:foreground, black: background). (d) map with a pixel-wise loss weight to force the network to learn the border pixels [32].

- matrix multiplication of input vectors
- scalar weighting of input vectors
- sum of weighted input vectors
- vector-to-vector nonlinearity

We see that the design of the capsule builds up upon the design of artificial neuron, but expands it to the vector form to allow for more powerful representational capabilities. It also introduces matrix weights to encode important hierarchical relationships between features of different layers. The result succeeds to achieve the goal of the designer: neuronal activity equivariance with respect to changes in inputs and invariance in probabilities of feature detection.

2.4 Image Processing

In this section, several image processing techniques that were used throughout this project are described.

2.4.1 Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986[[27]. Canny also produced a computational theory of edge detection explaining why the technique works.

Canny edge detection algorithm can be split into 5 different steps:

- Apply Gaussian filter to smooth the image in order to remove the noise
- Find the intensity gradients of the image
- Apply non-maximum suppression to get rid of spurious response to edge detection
- Apply double threshold to determine potential edges

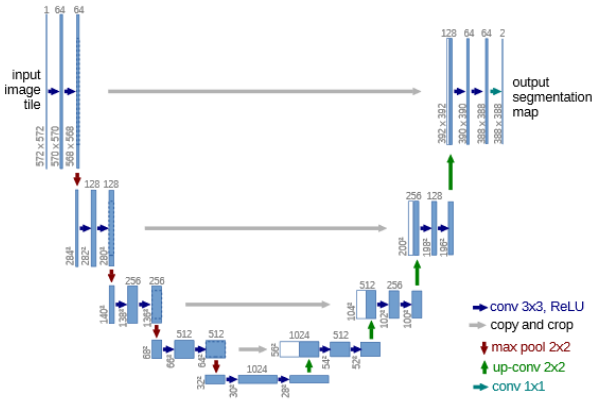


Figure 2.48: U-net architecture [32]

- Track edge by hysteresis: finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

In Figure 2.50 we show how this image processing technique looks like on *WSI*. It enhances the boundaries of each structure which can be used for nuclei identification but it can also be inconclusive and faulty without further image processing.

2.4.2 Color

Grayscale

In digital photography, computer-generated imagery, and colorimetry, a grayscale or greyscale image is one in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information. Grayscale images, a kind of black-and-white or gray monochrome, are composed exclusively of shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest.

Grayscale images are distinct from one-bit bi-tonal black-and-white images which, in the context of computer imaging, are images with only two colors: black and white (also called bilevel or binary images). Grayscale images have many shades of gray in between.

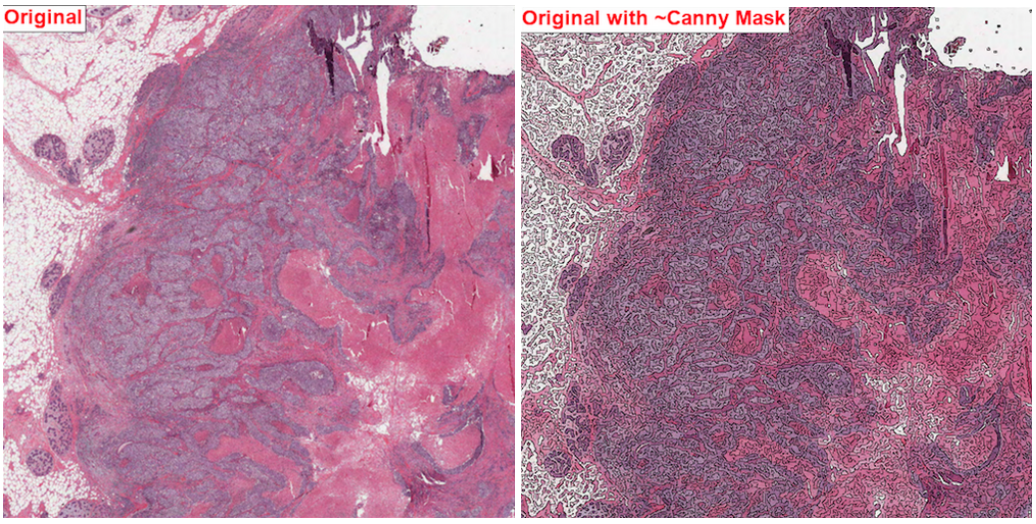
Grayscale images can be the result of measuring the intensity of light at each pixel according to a particular weighted combination of frequencies (or wavelengths), and in such cases they are monochromatic proper when only a single frequency (in practice, a narrow band of frequencies) is captured. The frequencies can in principle be from anywhere in the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.).

A colorimetric (or more specifically photometric) grayscale image is an image that has a defined grayscale colorspace, which maps the stored numeric sample values to the achromatic channel of a standard colorspace, which itself is based on measured properties of human vision.

In Figure 2.51 we can see the effect of grayscale technique on our patches.

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

Figure 2.49: Important differences between capsules and neurons[14].



(a) Original (b) Canny transformation

Figure 2.50: Canny Edge Transformation

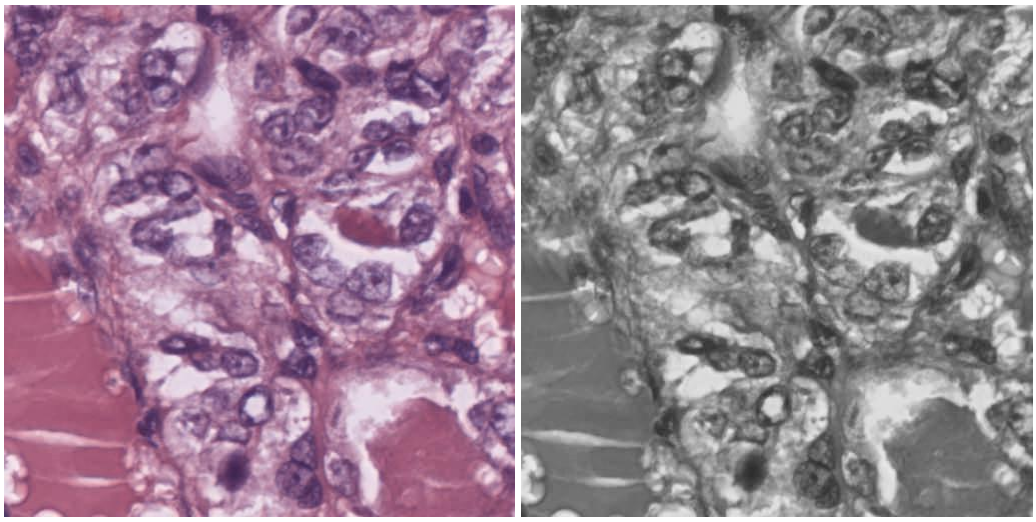
RGB to HED

Color deconvolution on the original RGB image to create HED (Hematoxylin, Eosin, Diaminobenzidine) channels. This is a very important technique since it allows to extract the channel color of each dye used for staining.

As we mentioned before, *Hematoxylin* is a dark blue or violet stain that is basic/positive. It binds to basophilic substances (such as DNA and RNA, which are acidic and negatively charged). Therefore, dyes like *hematoxylin* bind to DNA and RNA and stain them violet.

Eosin is a red or pink stain that is acidic and negative. It binds to acidophilic substances such as positively charged amino acid side chains (e.g. *lysine*, *arginine*). Most proteins in the cytoplasm of some cells are basic because they are positively charged due to the *arginine* and *lysine* amino acid residues. These form salts with acid dyes containing negative charges, like *eosi*n. Therefore, *eosin* binds to these amino acids/proteins and stains them pink.

Therefore, if we filter the *Hematoxylin* channel we enhance the basophilic substances, e.g. nucleus, and if we filter the *Eosin* channel we enhance the acidophilic substances, e.g. cytoplasm.

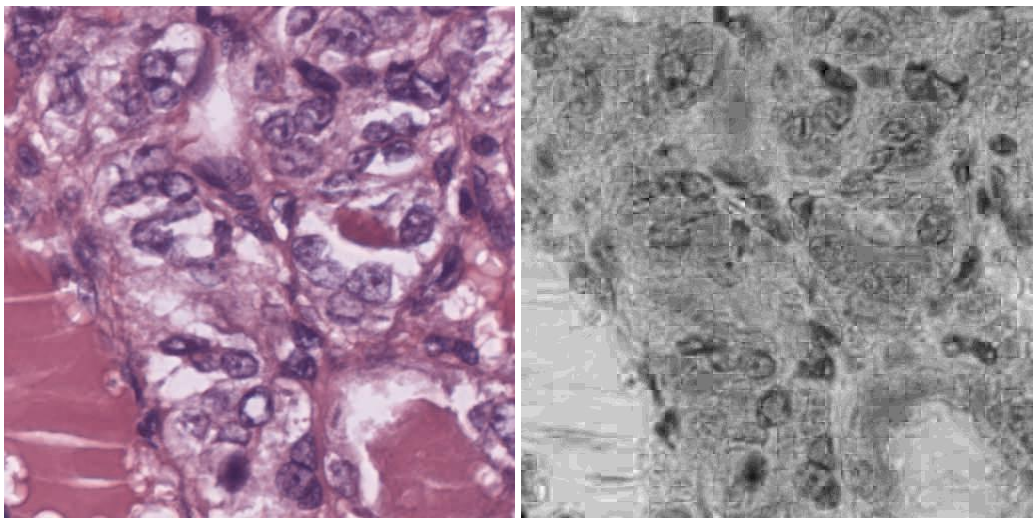


(a) Original (b) Grayscale image

Figure 2.51: Grayscale image processing.

HED to Eosin

Obtain Eosin channel from HED array and rescale it (e.g., to 0 to 255 for *uint8*) for increased contrast (Figure 2.52).

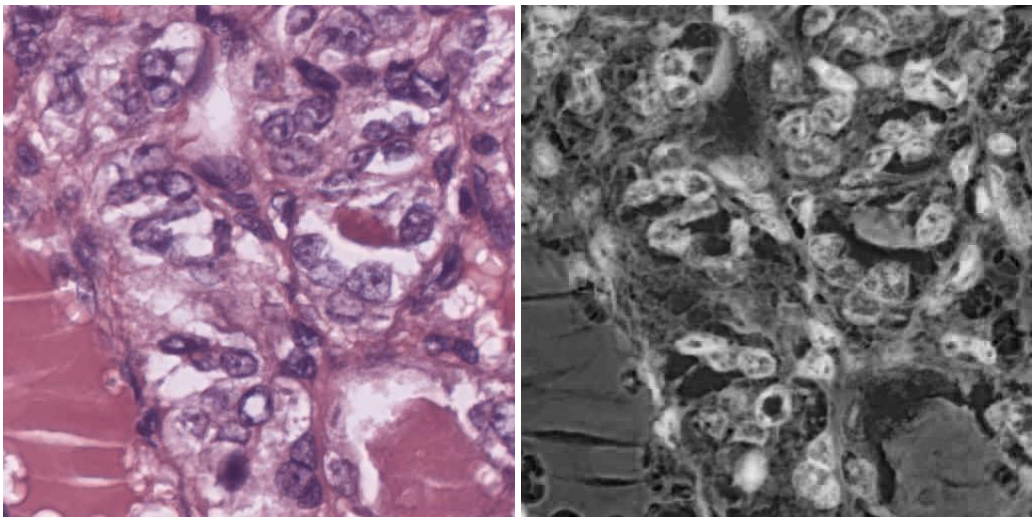


(a) Original (b) Eosin Channel Filter

Figure 2.52: Eosin channel filter image processing. The cytoplasm in the original that is highlighted with eosin staining gains a black color.

HED to Hematoxylin

Obtain Hematoxylin channel from HED array and rescale it (for example, to 0 to 255 for *uint8*) for increased contrast (Figure 2.53).



(a) Original (b) Hematoxylin Channel Filter

Figure 2.53: Hematoxylin channel filter image processing. The nucleus in the original image that is highlighted with hematoxylin staining gains a black color.

2.4.3 Contrast

Consider an image a histogram with the number of pixels (intensity on *y-axis*) plotted against the range of possible pixel values (*x-axis*, 0 to 255). Contrast is a measure of the difference in intensities. An image with low contrast is typically dull and details are not clearly seen visually. An image with high contrast is typically sharp and details can clearly be discerned. Increasing the contrast in an image can be used to bring out various details in the image (See 2.54).

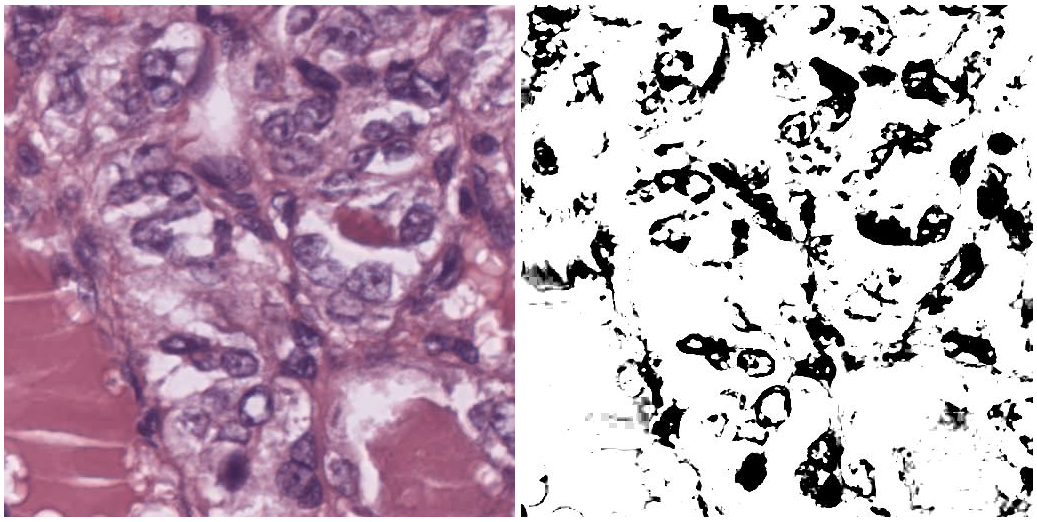
Contrast Stretching

One form of increasing the contrast in an image is contrast stretching. Consider that all intensities in an image occur between 100 and 150 on a scale from 0 to 255. If we rescale the intensities so that 100 now corresponds to 0 and 150 corresponds to 255 and we linearly rescale the intensities between these points, we have increased the contrast in the image and differences in detail can more clearly be seen. This is contrast stretching. See Figure 2.54 for an example.

Histogram Equalization

This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

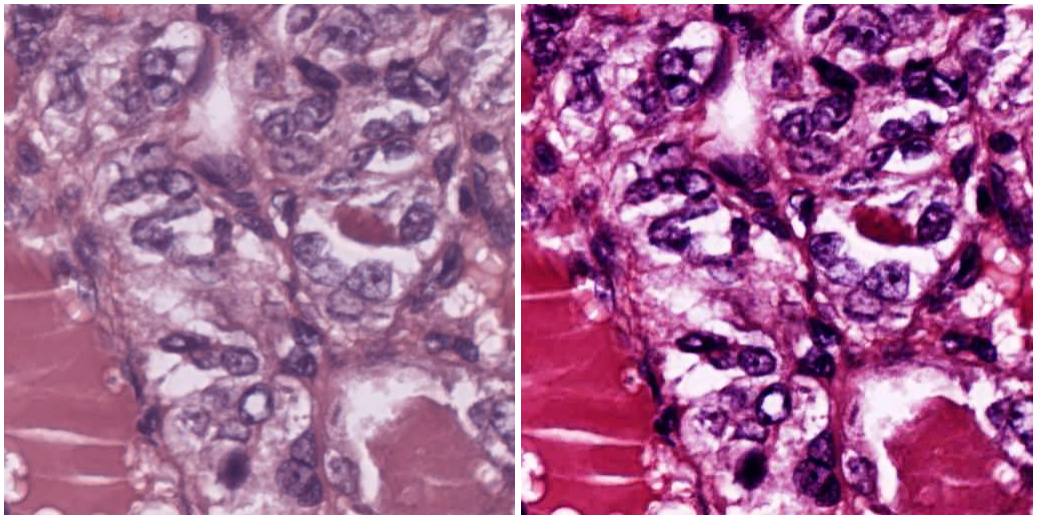
The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed. A key advantage of the method is that



(a) Original (b) Contrast Stretching Filter

Figure 2.54: Contrast Stretching Image Processing. The contrast in the original image is increased by stretching the image colors' intensity to a [0,255] range.

it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered. The calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal. (Figure 2.55).

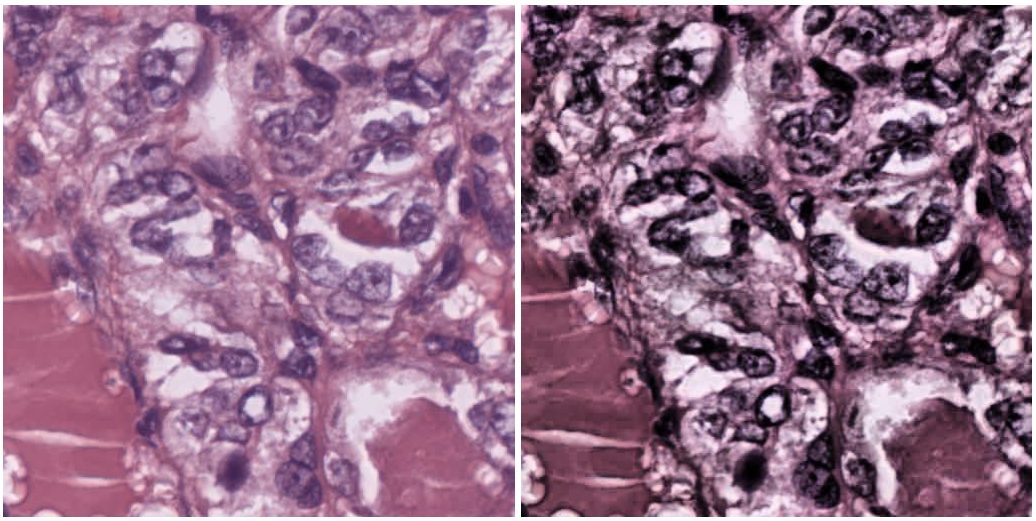


(a) Original (b) Histogram Equalization Filter

Figure 2.55: Histogram Equalization Image Processing. Areas with low image contrast in the original image gain an higher contrast and areas with similar contrast are highlighted from each other.

Adaptive Equalization

Rather than applying a single transformation to all pixels in an image, adaptive histogram equalization applies transformations to local regions in an image. As a result, adaptive equalization allows contrast to be enhanced to different extents in different regions based on the regions' intensity histograms. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image (Figure 2.56).



(a) Original (b) Adaptive Equalization Filter

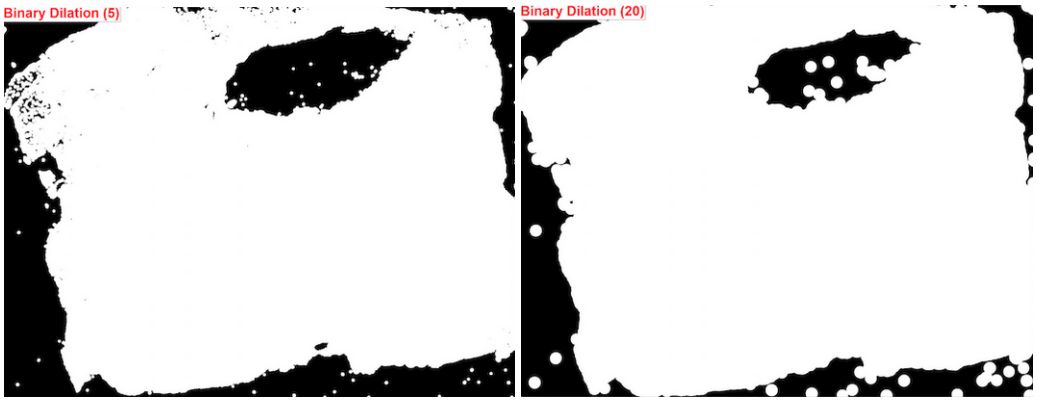
Figure 2.56: Adaptive Equalization Image Processing. Unlike Histogram Equalization, it enhances local regions by generating several histograms for each section of the image.

2.4.4 Morphology

Mathematical morphology (MM) is a theory and technique for the analysis and processing of geometrical structures, based on set theory, lattice theory, topology, and random functions. MM is most commonly applied to digital images, but it can be employed as well on graphs, surface meshes, solids, and many other spatial structures. The primary morphology operators are erosion, dilation, opening, and closing. With erosion, pixels along the edges of an object are removed. With dilation, pixels along the edges of an object are added. Opening is erosion followed by dilation. Closing is dilation followed by erosion. With morphology operators, a structuring element (such as a square, circle, cross, etc) is passed along the edges of the objects to perform the operations. Morphology operators are typically performed on binary and grayscale images.

Binary Dilation

- Dilation adds pixels to the boundaries of objects in an image or *grow* or *thickens* objects in a binary image.



(a) Binary dilation with *disk = 5* (b) Binary Dilation with *disk = 20*

Figure 2.57: Binary Dilation with a different disk size.

- The number of pixels added on the size & shape of the structuring element
- Let E be a Euclidean space or an integer grid, A a binary image in E, and B a structuring element regarded as a subset of R^d.

$$A \oplus B = \bigcup_{b \epsilon B} A_b$$

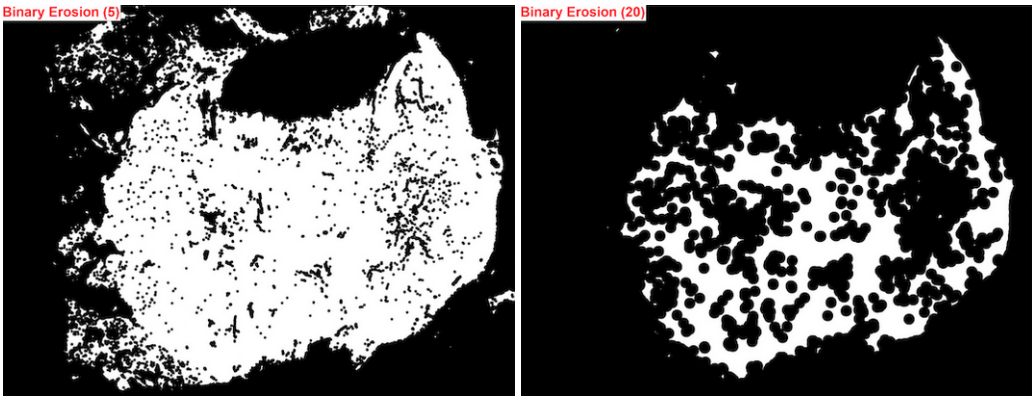
In our WSI image, binary dilation looks like Figure 2.57. The higher the *disk* the biggest is the dilation.

Binary Erosion

Erosion is one of the two basic operators in the area of mathematical morphology, the other being dilation. It is typically applied to binary images, but there are versions that work on grayscale images. The basic effect of the operator on a binary image is to erode away the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels shrink in size, and holes within those areas become larger. The erosion operator takes two pieces of data as inputs. The first is the image which is to be eroded. The second is a (usually small) set of coordinate points known as a structuring element (also known as a kernel). It is this structuring element that determines the precise effect of the erosion on the input image.

- Suppose that A is the set of Euclidean coordinates corresponding to the input binary image, and that B is the set of coordinates for the structuring element.
- Let B_z denote the translation of B so that its origin is at z.
- Then the erosion of A by B is simply the set of all points z such that B_z is a subset of A.

$$A \ominus B = \{z \in E | B_z \subseteq A\}$$



(a) Binary Erosion with *disk = 5* (b) Binary Erosion with *disk = 20*

Figure 2.58: Binary Erosion

Binary Opening

In mathematical morphology, opening is the dilation of the erosion of a set A by a structuring element B:

$$A \circ B = (A \ominus B) \oplus B$$

where $\ominus$ and $\oplus$ denote erosion and dilation, respectively.

Together with closing, the opening serves in computer vision and image processing as a basic workhorse of morphological noise removal. Opening removes small objects from the foreground (usually taken as the bright pixels) of an image, placing them in the background, while closing removes small holes in the foreground, changing small islands of background into foreground. These techniques can also be used to find specific shapes in an image. Opening can be used to find things into which a specific structuring element can fit (edges, corners, ...).

One can think of B sweeping around the inside of the boundary of A, so that it does not extend beyond the boundary, and shaping the A boundary around the boundary of the element. On our images Binary Opening can be used to remove small foreground objects (Figure 2.59)

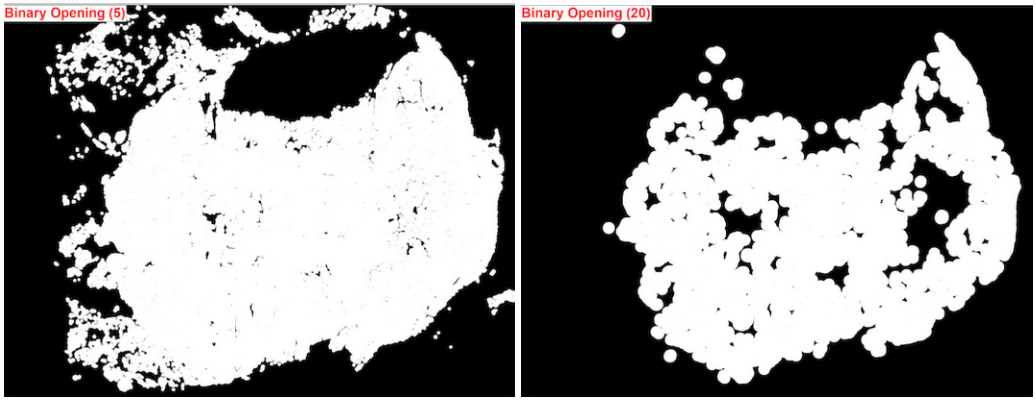
Binary Closing

In mathematical morphology, the closing of a set (binary image) A by a structuring element B is the erosion of the dilation of that set,

$$A \bullet B = (A \oplus B) \ominus B$$

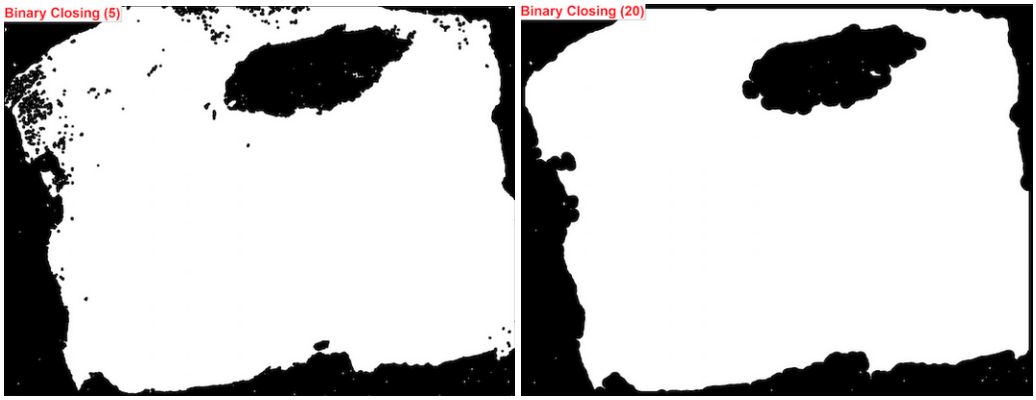
where $\ominus$ and $\oplus$ denote erosion and dilation, respectively.

In image processing, closing is, together with opening, the basic workhorse of morphological noise removal. Opening removes small objects, while closing removes small holes.



(a) Binary Opening with *disk = 5* (b) Binary Opening with *disk = 20*

Figure 2.59: Binary Opening



(a) Binary Closing with *disk = 5* (b) Binary Opening with *disk = 20*

Figure 2.60: Binary Closing

An example of binary closing with different disk sizes can be seen in Figure 2.60. We can that the with a bigger disk size (Figure 2.60b) small holes are completely removed while with a smaller disk size (Figure 2.60a) edges are not eroded enough to fill the small holes.

2.4.5 Watershed

A watershed is defined as a region of land that drains water into a river or a creek. It is an area of high ground through which water flows into the river (Figure 2.61).

Simply defined, watershed is a transformation on grayscale images. The aim of this technique is to segment the image, typically when two regions of interest are close to each other — i.e, their edges touch or they are overlapped. This technique of transformation treats the image as a topographic map, with the intensity of each pixel representing the height. For example, "darker areas can be intuitively considered to be 'lower' in height and can represent troughs. On the other hand, bright areas can be considered to be 'higher', acting as hills or as a mountain ridge"[15]. (Figure 2.62).

[15] http://thewatershedproject.org/what-is-a-watershed/

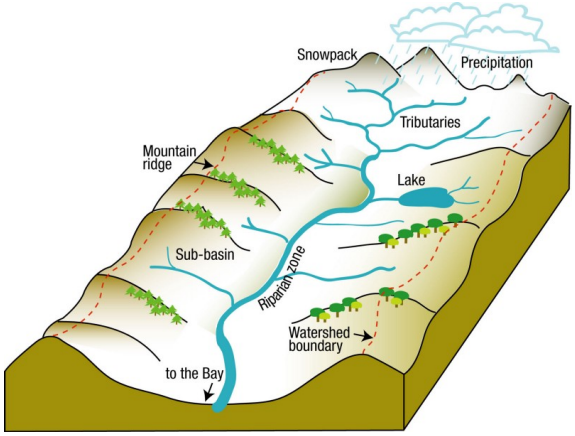


Figure 2.61: A geographical watershed.[15]

Various algorithms can be used to compute watersheds. One of the most popular algorithm is **watershed-by-flooding** that is described below.

Watershed-by-flooding

Consider that a source of water is placed in the catchment basins — the areas with low intensity. These basins are flooded and areas where the floodwater from different basins meet are identified. Barriers in the form of pixels are built in these areas. Consequently, these barriers act as partitions in the image, and the image is considered to be segmented.

An example of watershed can be seen in Figure 2.63.

2.4.6 Normalizing Staining

Inconsistencies in the preparation of histology slides make it difficult to perform quantitative analysis on their results. So, we followed a previous project [23] that purposes two different mechanisms to overcome many of the known inconsistencies. As we mentioned before these slides retrieved from digital scanner pass through a staining process. Staining is used to highlight important features of the tissue as well as to enhance the tissue contrast. Hematoxylin is a basic dye that is widely used in this process and stains the nuclei giving it a bluish color while eosin (another stain dye used in histology) stains the cell's nucleus giving it a pinkish stain [3]. The majority of stains only absorb light, and the stained slides are therefore viewed using a microscope with a light illuminating the sample from below. If no stain is present, all of the light will pass through, appearing bright white. Areas where the stain has adhered to a substance in the tissue will absorb some of the light. The amount of light absorbed depends on many factors. This paper purposed a solution to minimize the variations of the amount of light spread along the slide. This paper assumes that there is a specific stain vector corresponding to each of the two stains present in the image, and that the resulting color (in OD space) of every pixel is a linear combination of these stain vectors.

[16] https://www.mathworks.com/company/newsletters/articles/the-watershed-transform-strategies-for-image-segmentation.html

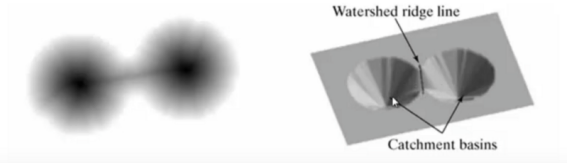
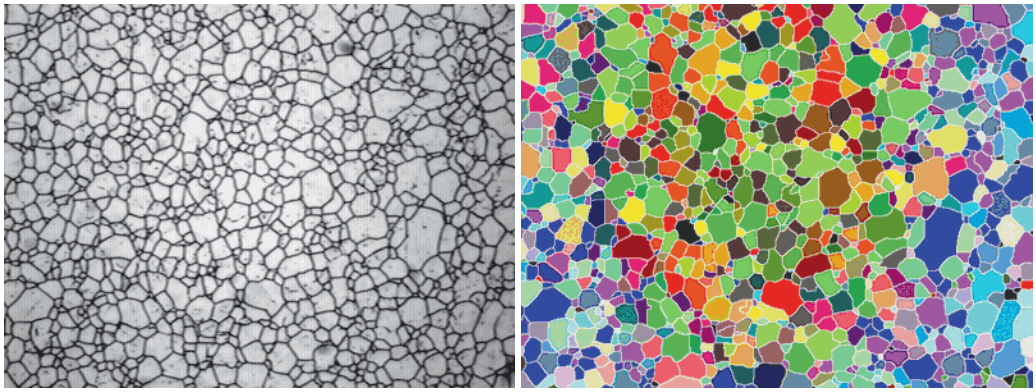


Figure 2.62: Visualizing the watershed: the image on the left can be topographically represented as the image on the right[15]



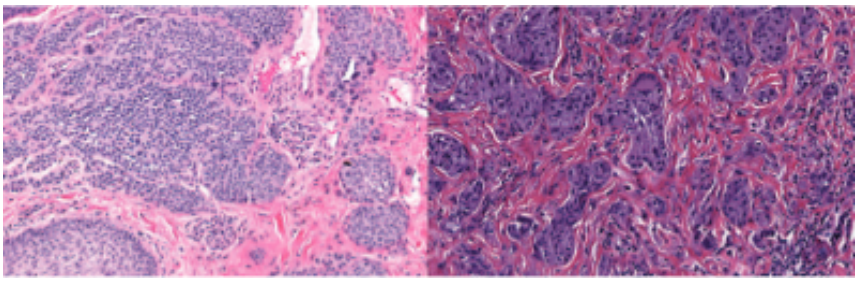
(a) Microscope image of steel grains. (b) Watershed Transformation

Figure 2.63: Example: Segmenting Steel Grains[16]

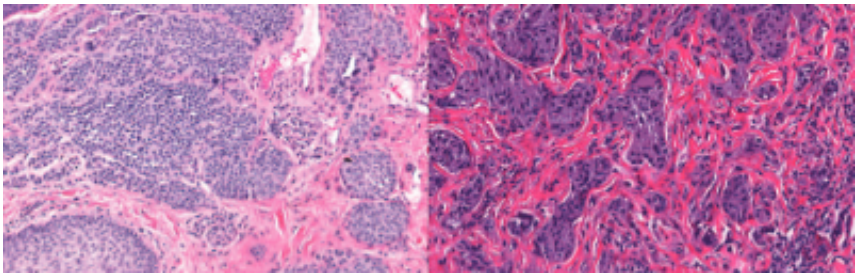
The presented algorithm has the following steps:

1. **Input:** RGB Slide
2. Convert RGB to Optical Density
3. Remove data with Optical Density less than β
4. Calculate singular value decomposition on the Optical Density tuples
5. Create plane from the SVD directions corresponding to the two largest singular values.
6. Project data onto the plane, and normalize to unit length.
7. Calculate angle of each point *wrt* the first SVD direction.
8. Find robust extremes(α^{th} and $(100-\alpha)^{th}$ percentiles) of the angle
9. Convert extreme values back to OD space
10. **Output:** Optimal Stain Vectors

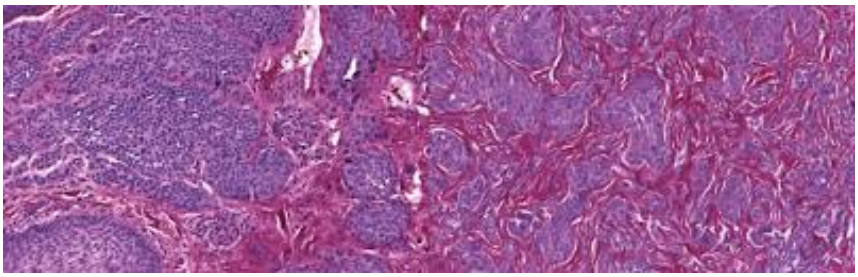
An example of the effect of this normalize staining algorithm can be seen in Figure 2.64 [23].



(a) Example of two histopathology slides of melanomas, both stained with hematoxylin and eosin, but with drastically different appearances. The images were obtained by scanning the slides at 20X using an Aperio Scanscope



(b) Same images as Figure 2.64a, but the second slide has been transformed into the same colorspace as the first slide by the method purposed by the paper.



(c) Same images as Figure 2.64a, but both are now at the same average intensity level by the method discussed by the project

Figure 2.64: Normalize Staining [23]

2.5 Chapter Summary

In this chapter we presented a biological and technological background. The biological background provides a brief explanation of the cell and its components so we can understand which characteristics are important for the feature extraction process.

We also presented the state of the art methods for image recognition and classification. We started by presenting an overview of how neural networks work and why they are appropriated to image processing. Then, we presented different CNN models that are utterly used in image analysis. For each model, we presented the overall architecture, advantages and drawbacks and possible performances.

We can conclude that Mask R-CNN implementation and training is harder since it employs a two-stage learning approach, where you first optimize for an RPN (Region Proposal Network) and then predict bounding boxes, classes and masks simultaneously. On the other hand U-Net is

a very popular end-to-end encoder-decoder network for semantic segmentation and it was originally invented and first used for biomedical image segmentation. In this project we used both architectures but our segmentation was done using U-Net since it presented better results.

Capsule Networks was not used in this dissertation but we think it is relevant for this kind of problems and could be tested in further progress. A recent study [18] used CapsNet for classification of breast cancer histology images and the results were very promising. One of the reasons why CapsNet presents good results in histology images is because of the hierarchical representation of data. For example, to identify an *oncocyte* the inter-nuclei distance between cells is bigger due to the high amount of mithocondria in the cytoplasm. CapsNet uses this relationships and geometrical representations as features.

Chapter 3

Methodology

As mentioned before the data was downloaded from the *National Cancer Institute*(NCI). NCI provides a Genomic Data Commons Data Portal, a robust data-driven platform that allows cancer researchers and bioinformaticians to search and download cancer data for analysis. The project from which we downloaded our *Whole Slide Images* is named THCA (Thyroid Carcinoma). These images have a gigapixel resolution which makes them computationally hard and laborious to process. In the next sections we will explain the preprocessing steps applied to reduce our images size by cropping and applying traditional image processing techniques.

3.1 Data Loading and Preprocessing

Since the gigabyte size of a WSI poses serious challenges for scalable storage and fast retrieval, which is essential for next-generation image analytics, we propose a system for scalable storage of WSIs and fast retrieval of image tiles using Apache Spark.

Apache Spark is an open-source distributed general-purpose cluster-computing framework (see Figure 3.1). Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It has as its architectural foundation the Resilient Distributed Dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. We could also describe Spark as a distributed, data processing engine for batch and streaming modes featuring SQL queries, graph processing, and machine learning. In our case we did not take full advantage of this cluster computation since our application is standalone - we only used one machine. However we could have used parallel computing to increase our performance.

Spark Key Concepts:

- Application: This may be a single job, a sequence of jobs, a long-running service issuing new commands as needed or an interactive exploration session.

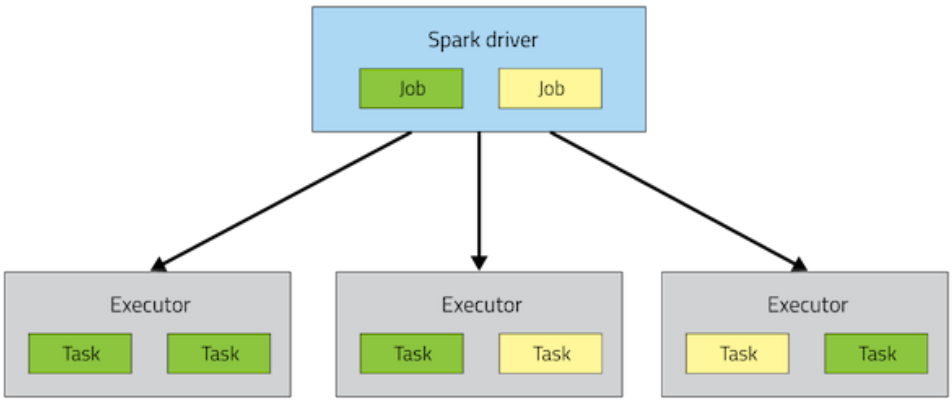


Figure 3.1: Spark Application architecture. A **driver** to manage the job flow and schedule tasks and **executors** that run concurrently in a single process.[17]

- Spark Driver: The Spark driver is the process running the spark context (which represents the application session). This driver is responsible for converting the application to a directed graph of individual steps to execute on the cluster. There is one driver per application.
- Spark Application Master: The Spark Application Master is responsible for negotiating resource requests made by the driver and finding a suitable set of hosts/containers in which to run the Spark applications. There is one Application Master per application.
- Spark Executor: A single JVM instance on a node that serves a single Spark application. An executor runs multiple tasks over its lifetime, and multiple tasks concurrently. A node may have several Spark executors and there are many nodes running Spark Executors for each client application.
- Spark Task: A Spark Task represents a unit of work on a partition of a distributed dataset.

Our Apache Spark system used the following configuration:

- Driver-Cores: 8
- Driver Memory 16GB
- Number of executors 8
- Executor Memory 8GB
- Executor Cores 1

An optimal system would have multiple workers, with multiple executors preferably with more memory. Even though we do not have an optimal system we managed to reduce the preprocessing time from 4,5 hour/image to 1 hour/image.

[17] https://blog.cloudera.com/blog/2014/05/apache-spark-resource-management-and-yarn-app-models/

RDD Operations

RDDs support two types of operations:

- transformations - which create a new dataset from an existing one.
- actions - which return a value to the driver program after running a computation on the dataset.

For example, *map* is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel *reduceByKey* that returns a distributed dataset).

All transformations in Spark are **lazy**, in the sense that they do not compute their results right away.

Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently – for example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.

By default, each transformed RDD may be recomputed each time we run an action on it. However, we can also keep an RDD in memory using the persist (or cache) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

Transformations

In our project we have applied some transformations, shown in Table 3.1, that can be seen as a sequence of steps shown below.

- **open_slide** - Open a whole-slide image, given an image number and returns an OpenSlide object representing a whole-slide image.
- **process_slide**- Generate all possible tile indices for a whole-slide image. Given a slide number, tile size, and overlap, generate all possible *(slide_num, tile_size, overlap, zoom_level, col, row)* indices. The zoom level is the maximum magnification or the closest possible.
- **process_tile_index**- Generate a tile from a tile index. Given a *(slide_num, tile_size, overlap, zoom_level, col, row) tile index*, generate a *(slide_num, tile)* tuple. Returns a *(slide_num, tile)* **tuple**, where **slide_num** is an integer, and **tile** is a 3D NumPy array of shape *(tile_size, tile_size, channels)* in RGB format.
- **keep_tile** - Determine if a tile should be kept. This filters out tiles based on size and a tissue percentage threshold, using a custom algorithm. If a tile has height & width equal to

Action	Function	Description	Result
filter	open_slide	Reads a Whole Slide Image	Return an OpenSlide object
flatMap	process_slide	Generate all possible tile indices for a WSI	All possible tile indices
Map	process_tile_index	Generate a tile from a tile index.	A (slide_num, tile) tuple
filter	keep_tile	Determine if a tile should be kept.	A Boolean indicating whether or not a tile should be kept.
flatMap	process_tile	Generates all the kept indices	Kept tile indices
map	normalize_staining	Normalize the staining of H&E histology slides.	Normalized slides.

Table 3.1: Transformations applied to our WSI dataset

(tile_size, tile_size), and contains greater than or equal to the given percentage, then it will be kept; otherwise it will be filtered out. The algorithm consists in a set of image processing techniques to verify if the percentage of tissue present in a tile is higher than the **threshold**. The steps are the following ones:

1. The image is converted to **grayscale**
2. **8 bit 2's complement**, from 1 (dense tissue) to 0 (plain background).
3. **Canny edge detection** with hysteresis thresholding. This returns a binary map of edges, with 1 equal to an edge. The idea is that tissue would be full of edges, while background would not.
4. **Binary closing**, which is a *dilation* followed by an *erosion*. This removes small dark spots, which helps remove noise in the background.
5. **Binary dilation**, which enlarges bright areas, and shrinks dark areas. This helps fill in holes within regions of tissue.
6. Fill remaining holes within regions of tissue.
7. Calculate percentage of tissue coverage.

- **process_tile** - Process a tile into a group of smaller samples. Cut up a tile into smaller blocks of *sample_size* x *sample_size* pixels, change the shape of each sample from (H, W, channels) to (channels, H, W), then flatten each into a vector of length *channels*H*W.*

- **normalize_staining** - Normalize the staining of H&E histology slides. This function normalizes the staining of H&E histology slides. See Figure 2.4.6 for the implementation details and Figure 3.2 for an example.

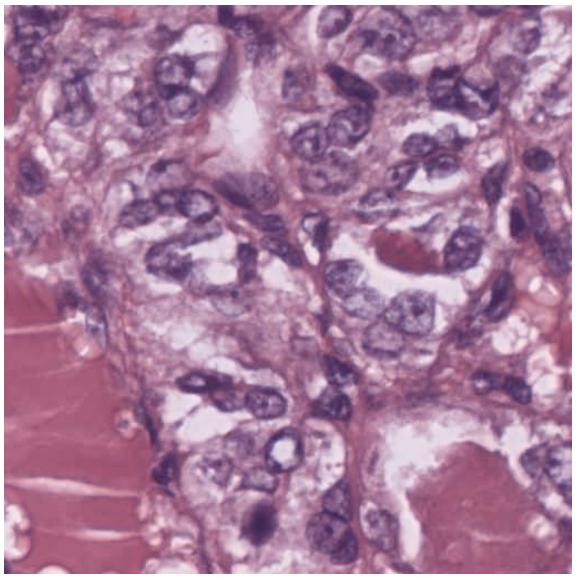


Figure 3.2: Stain normalized image

- **RGB to HED** - Filter RGB channels to HED (Hematoxylin - Eosin - Diaminobenzidine) channels.

- **HED to Eosin** - Obtain Eosin channel from HED Image and rescale it for increased contrast (Figure 3.3)

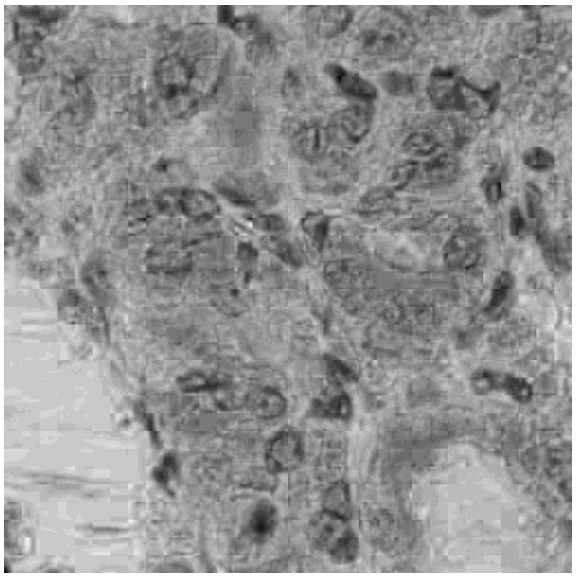


Figure 3.3: HED to Eosin Channel

- Contrast with **Histogram Equalization** (Figure 3.4)

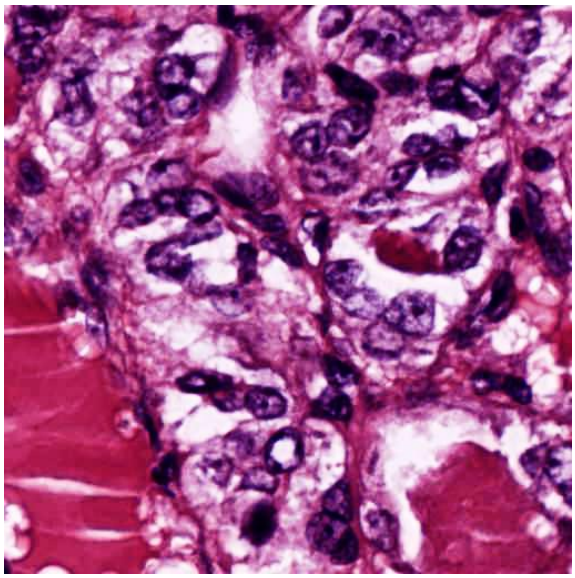


Figure 3.4: Histogram Equalization Contrast

This image processing steps proved to be very helpful when performing our segmentation since foreground extraction (FE) decreased the images size by almost 40% (see Table 3.2) and when applying normalize staining we removed the noise caused by light and other external factors. Furthermore, this preprocessing module also increases the classification of tumor cells. In Section 3.3 we compare the results of several models performance metrics between a dataset with preprocessing and a dataset without it.

3.2 Nuclei identification

As we mentioned before, nucleus are a very small structure, with a very specific morphology, i.e. small components with a specific phenotype and its recognition is dependant on many factors, such as lightning, staining, equipment used, etc. This deep and detailed processing of WSI will be handful later, in the segmentation process. The aim of this project is to identify oncocytic cells on thyroid tissue slides which is a phenomenon that can be identified on the nucleus shape and phenotype. Therefore, it is important to have a clear image of the nucleus that is going to be annotated. To ease the pathologist task of annotating every single oncocyte, we implemented a

WSI	**Original size**	**After FE**
001	3686.4 MB	2334.7 MB
002	2578.8	1578.3 MB
003	3256.1	1953.9 MB

Table 3.2: Size comparison between three WSIs before and after Foreground Extraction.

pre-segmentation process that identifies all the nuclei in an image. Therefore, instead of annotating every single structure on a tissue slide, the pathologist only has to annotate every *segmented* nuclei as *oncocyte* or not.

To detect the nucleus in patch images we used a model developed by Selim Seferbekov and his team to the *Data Science Bowl 2018* challenge. *DSB2018* is a Kaggle competition[18] and the proposed challenge was to identify nucleus in divergent images. The challenge ended up by April 2018 and the winners shared their solution. They won the competition by a significant difference and their solution proved to be state of the art in nuclei identification using object detection algorithms. For our pre-segmentation step used the winner U-Net Neural Network (Section 2.3.9) with a encoder-decoder architecture that we will be described in the next section.

Segmentation

On the constructed model the authors used a **UNet** like encoder-decoder architectures with encoders pretrained on ImageNet. The encoders they used were: *DPN-92, Resnet-152, Inception-ResnetV2, Resnet101* and ensembled the results. However, the only encoder-decoder that we used in this project was *ResNet-101* since it presented better results. The approach followed by Selim and his team proved to be the best among the other models used on *DSB1018* and there are a few reasons for that. The main constributions are the following:

- **Targets** - touching borders were predicted along with the other two channels. The results are a three channel mask composed by *nuclei*, *background*, *boundaries*. Figure 3.5.
- The problem is an **instance segmentation**. Labels were generated from the given masks.
- **Loss function** - combines CrossEntropy and soft dice loss so that pixel imbalance doesn not affect the results.
- Very deep encoder-decoder architectures that also achieve state-of-the-art results in other binary segmentation problems (SpaceNet, Inria and others)
- Exhaustive postprocessing that combines watershed, morphological feature and second-level model with Gradient Boosted Trees. Figure 3.6.
- Task specific data **augmentations**

Target Masks

Initially, the authors tried the simplest approach and added watershed line for binary masks. This helped with the labelling of the given masks since it detects boundaries but it was not enough for overlapping nucleus. Therefore, it was added a second channel with contours, where width of contours depended on the nucleus size. However, the networks were easily predicting contours in non ambiguous places but having difficulties in places where that was actually needed contours to

[18] https://www.kaggle.com/c/data-science-bowl-2018

separate the nuclei. Hence the authors decided to predict only the borders between the cells. This approach improved their solution since that they could finally separate the nucleus masks and get a label for each training image.

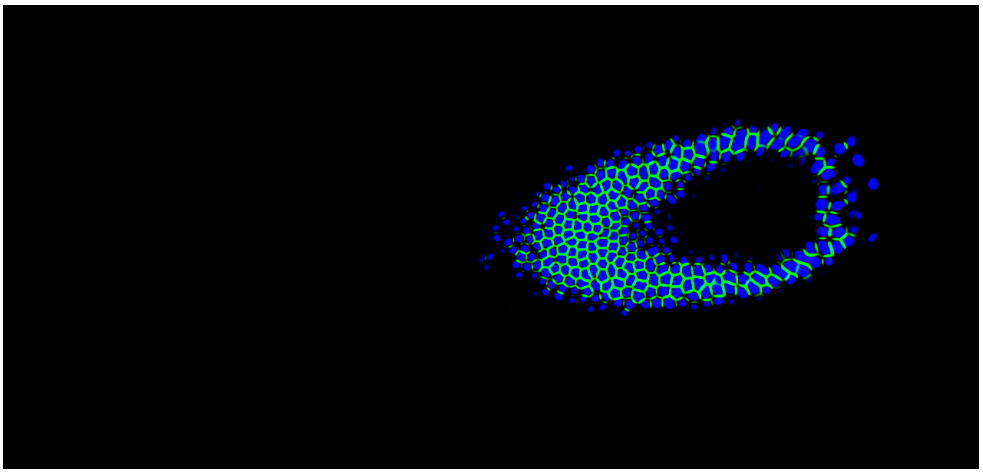


Figure 3.5: Resulting Masks from the model. Three visible targets: nucleus (blue), boudaries (light blue) and background (black)

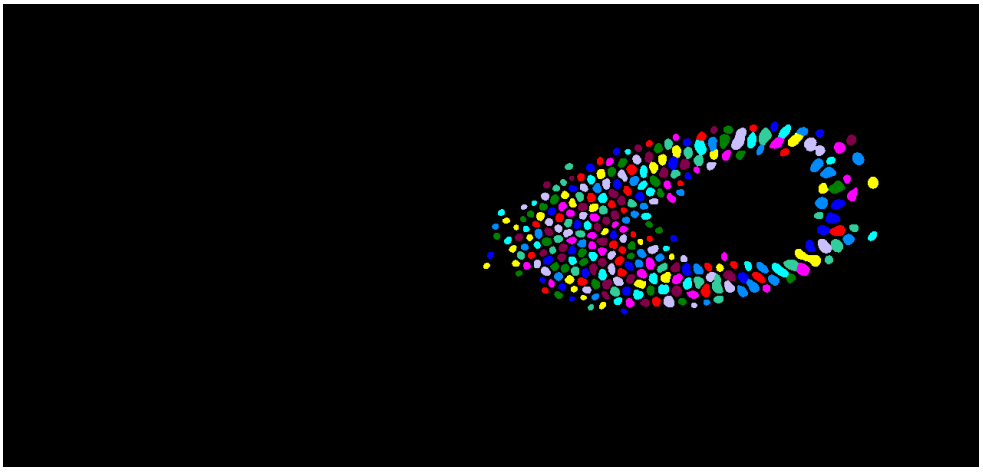


Figure 3.6: Masks after Watershed postprocessing

Augmentations

Since the training data used for the segmentation problem did not have enough images, specific augmentations were used to prevent models from overfitting and made them more or less generalizable. For that, they used a lot of heavy augmentations.

- CLAHE, Sharpen, Emboss
- Gaussian Noise

- Color to Gray
- Remapping grayscale images to random color images
- Blur, Median Blur, Motion Blur
- Contrast and brightness
- Random scale, rotates and flips
- Heavy geometric transformations: Elastic Transform, Perspective
- Transform, Piecewise Affine transforms, pincushion distortion
- Random HSV
- Channel shuffle - divergent data
- Nucleus copying on images. That created a lot of overlapping nuclei which helps networks to learn better borders for overlapping nuclei.

Training parameters

As we mentioned before, U-Net architecture was used with a deep encoder-decoder. The model was trained with the following hyper-parameters:

- Random Crops: 256x256
- Batch Size: 16
- Optimizer: Adam
- Learning rate: initial 1e-4 with decay (we had different learning rate (LR) policies, but mostly small LR no more than 1e-4)

Loss function

For networks with sigmoid activation and 2 channel masks they used a combination of *binary crossentropy* with *softdice* per channel. For networks with softmax activation and 3 channel masks they a used combination of *categorical crossentropy* with *soft dice* per channel (soft dice was applied only to mask and border channels).

Postprocessing

Now that we have all nucleus identified, we need to import the masks onto our application. As we mentioned before, the output from the segmentation process are 3-channel masks, *nuclei*, *background*, boundaries. To get the nuclei contour, we need to apply some image processing techniques to separate the nucleus from the predicted mask.

An example of the result from the segmentation process is shown on Figure 3.7.

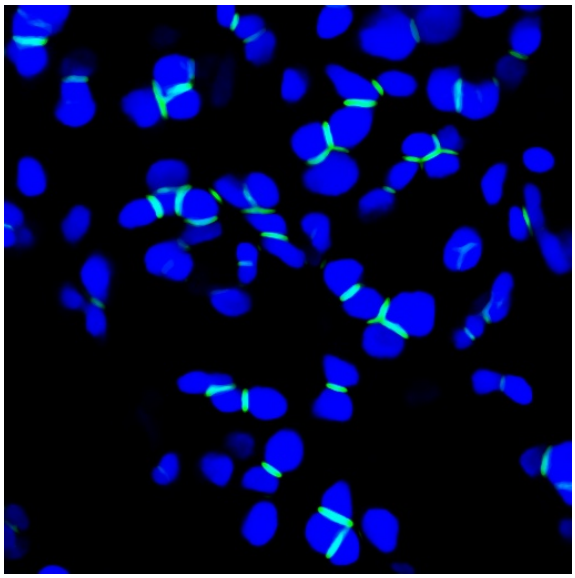


Figure 3.7: Resulting Masks of a single patch

- The *black* color represents the background
- The *blue* color represents the nuclei
- The *light blue/green* color represents the boundaries between two overlapped nucleus

Applying watershed (Section 2.4.5) to the image we can separate adjacent nucleus and get the boundary line (Figure 3.8).

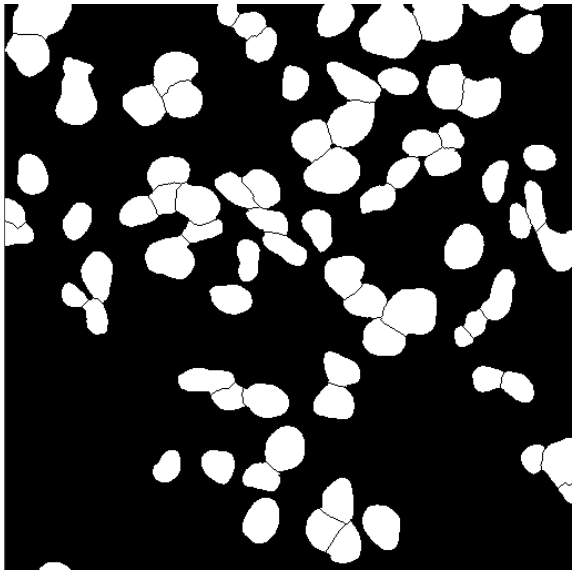


Figure 3.8: Resulting mask of a single patch with watershed

```
{
        "filename": "",
        "size": -1,
        "regions": [
        {
                "shape_attributes": {
                    "name": "polyline",
                    "all_points_x": [],
                    "all_points_y": []
                }
            }
            ],
        "file_attributes": {}
    }
```

Listing 3.1: JSON file object

However, with this *whatershed mask* the nucleus are not differentiated since that adjacent cells share the boundary. To have a nuclei individually segmented, we need to fill the contours by eroding the boundaries. As we mentioned in Section 2.4.4, binary erosion uses a disk as the structuring element that erodes the edges, usually denominated *disk*. The result of binary erosion on our masks can be seen in the Figure 3.9.

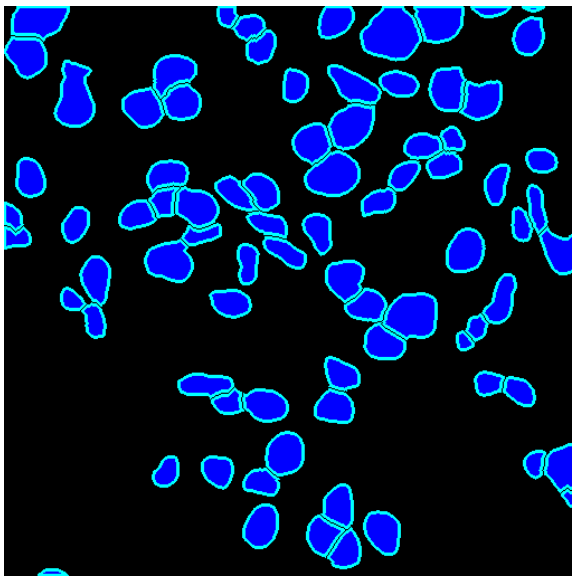


Figure 3.9: Resulting mask of a single patch with watershed and binary erosion

With this approach, cells that were segmented as overlapping (some may have escape) are correctly separated, with its own boundary. To get the coordinates of those boundaries we use an *OpenCV* module to find the object contours that returns the x and y points of the whole polygon. Finally, we created a *JSON* file with the format presented in Listing 3.1.

3.3 Experimental Procedure

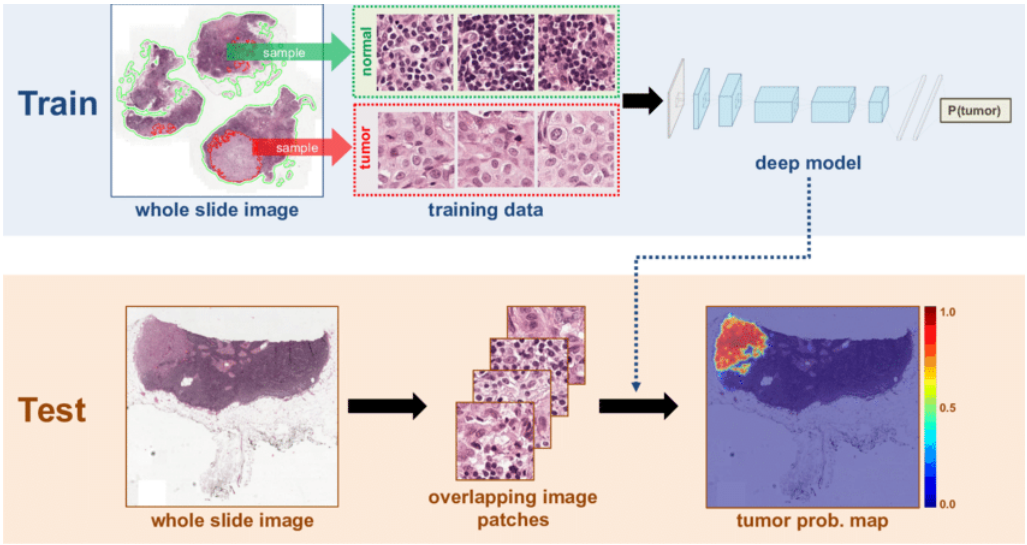


Figure 3.10: Framework pipeline.[19]

In this section, we briefly describe the details about the dataset, experiment setting, and demonstrate the performance of our framework for tumor cell classification. After the preprocessing our dataset is divided in two labels - *tumor* and *normal*. The framework pipeline can be seen in Figure 3.10 and can be divided in the following steps:

- Extract patches from ROI (Regions of Interest)
- Divide patches in two classes - tumor and normal
- Feed the neural network with 256x256 patches.
- Use the best model to test a WSI
- Generate an heatmap with tumor cells identified

3.3.1 Dataset

Having our images correctly annotated by the pathologist, we divide them into labels and build our dataset. Our dataset has a total of 100000 images cropped in a 256x256 resolution that are splitted by the *train*, *validation* and *test* set. On the Table 3.3 we can see the dataset distribution.

3.3.2 Data Augmentation

Data augmentation adds diversity to the dataset by adding information that is derived from the actual data. This synthetically modified data, can represent a variety of conditions in which our

[19] https://www.researchgate.net/publication/304163398_Deep_Learning_for_Identifying_Metastatic_Breast_Cancer

	Tumor	Normal
Train	31596	30874
Validation	5321	5112
Test	5783	5823

Table 3.3: Train, valid and test set samples distribution

data can be presented. For example, if our images are always centered and have the same scale, the model will have difficulty to train when those kind of images are computed. In our case, images can have a different zoom, a different brightness and a different staining. Using augmentation will help enhancing features that are very important in the classification process. In this project we used the following data augmentation:

- Rotations
- Random Flips (Horizontal and Vertical)
- Brightness
- Random Grayscale

Adding more diversity contributes for a more robust dataset and helps to avoid overfitting.

3.3.3 Hardware Specification

The experimental setup was conducted on a single machine with the following specifications:

- Processor. Intel Core i7-6700HQ (Intel Core i7)
- Graphics adapter: NVIDIA GeForce GTX 1060 (Laptop) - 6144 MB, GDDR5
- Memory: 16384 MB DDR4

3.3.4 Deep Learning Software

Keras, TensorFlow and PyTorch are among the most known frameworks that are used for Deep Learning. The advantages and drawbacks of each framework depends on the problem that is being addressed and on the approach that is followed. A brief description of these popular frameworks is presented below.

Keras

Keras is an open source high-level API for neural network frameworks. Its main features include user friendliness, modularity, and ease of extensibility. Keras is written in Python which makes for an easy to understand source code. It is one of the most used high-level wrappers for Tensorflow. Developing neural networks is quicker when using it for many commonly used layer declarations.

The wrapper provides means to build networks, 27 load pre-trained weights, preprocess data, augment data, asynchronously feed training data and other features.

Tensorflow

Keras is a high-level neural networks API, written in Python and capable of running on top of different backend frameworks for computation-heavy tasks such as Tensorflow, CNTK or Theano. TensorFlow is a framework that provides both high and low level APIs and is used for high-performance numerical computations across various computing hardware. To achieve high performance levels, it uses optimized code for specific hardware. For instance, on NVIDIA GPUs can use cuDNN[1]library for computation-heavy tasks.

PyTorch

PyTorch[2]is a lower-level API focused on direct work with array expressions. It is primarily developed by Facebook's artificial-intelligence research group and Uber's "Pyro" software for probabilistic programming is built on it. Pytorch is

PyTorch provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autodiff system

In terms of programming, Tensors can be considered multidimensional arrays. Tensors in PyTorch are similar to *NumPy* arrays, with the addition being that Tensors can also be used on a GPU that supports CUDA.

In our project, Deep learning experiments were conducted using Keras with Tensorflow backend. We also used *Transfer Learning*, i.e. pre-trained **PyTorch** models, for our classification process.

3.3.5 Experimental Setup

During training and inference, we extracted 304x304 patches from WSIs at the highest magnification. We trained our data using the following models:

- ResNet - Residual Networks (Section 2.3.2)
  - ResNet101
  - Resnet50
  - Resnet34
  - Resnet18

[1]https://developer.nvidia.com/cudnn
[2]https://www.analyticsvidhya.com/blog/2018/02/pytorch-tutorial/

- DenseNet - Densely Connected Networks (Section 2.3.3)
  - DenseNet101
- InceptionV3 (Section 2.3.4)

Each of these models have a different architecture. For example, ResNet models can have different architectures as we mention in Section 2.3.2.1. The more layers a model have, the deeper it is, meaning that it will help to extract more complex features, but we can only do that up to a certain extent. There is a limit and after that, instead of extracting features, we tend to *overfit* the data. Although we have a significant amount of data, histology slides are very rich in information and the thyroid cytology contains many structures that can be hard to identify and can differ from slide to slide. To understand what we need to classify, we have to know exactly what kinds of features we are looking for and the learning growth we want for our model.

Since we have some hardware limitations we have to take special care with our parameters and tune our model to process fewer images at a time, which can lead to overfitting. In the Figure 3.11, it is shown the loss of a *ResNet101* model. ResNet has 101 layers and it is one of the deepest models we experimented. We can see that after the 8th epoch the model starts overfitting, meaning that the loss on the validation set is diverging from the training loss. The model is not learning anymore but memorizing each label. For this kind of issues we implemented few mechanisms to control *overfitting*. The first was already mentioned, it is data augmentation which adds diversity to the original patches. The other mechanisms are presented below.

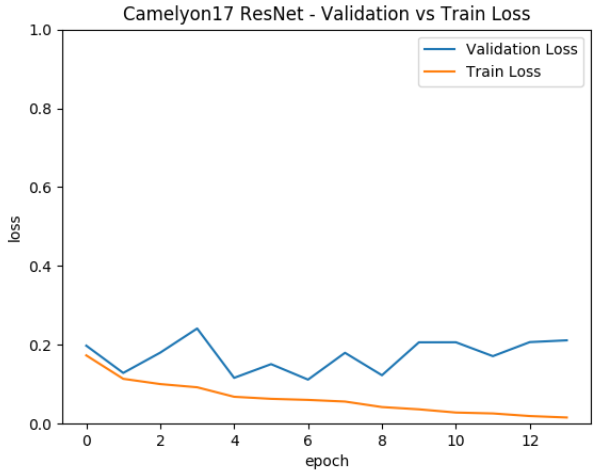


Figure 3.11: Training versus Validation loss on *ResNet101*

3.3.5.1 Early Stopping

Our model has a saturation point. This saturation point shows the moment when the validation and training loss are converging. When the training/validation loss starts to diverge, the model is

overfitting. Therefore, we stopped our model when it does not learn anything relevant. Our **early stopping** method works as follows:

1. The best loss starts at infinity
2. We give our model a *patience* variable. When the patience runs out, we stop the training process.
3. Compare the validation loss at each epoch with the best loss.
4. If the current validation loss is better than the actual *best loss*, it becomes the new *best loss* and we continue.
5. If the validation loss is worse than the *best loss* we decrease our *patience* by one.
6. When *patience* reaches 0, the model stops.

With this mechanism we can keep track of our best validation loss and we can check if we are overfitting and wait for our curve to be steep. As soon as the model starts overfitting, the results are not relevant and we can stop learning and save the model with the best loss ratio. In Figure 3.12 we can see the **early stopping** mechanism on the *ResNet101* model. The validation curve stops converging at *epoch* 6 and given a *patience=7* it will give up on the model after seven epochs without learning or improving its loss.

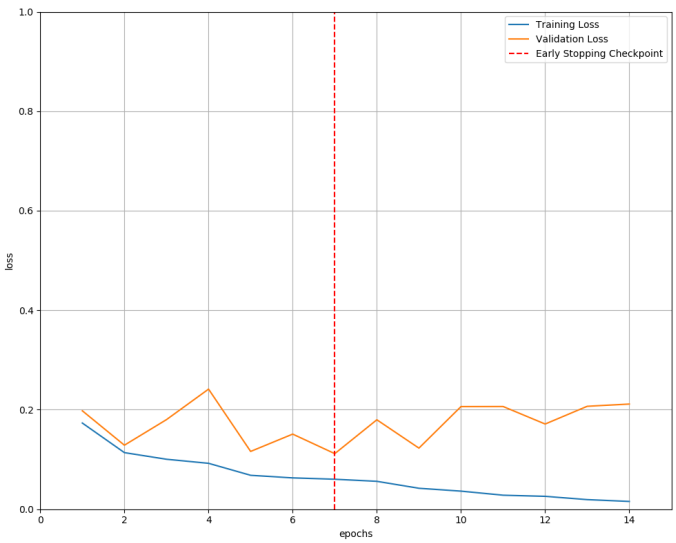


Figure 3.12: Early Stopping on *ResNet101*

3.3.5.2 Learning Rate Adjustment

The learning rate is how quickly a network abandons old beliefs for new ones. If our model sees 10 examples of tumor cells and all of them have a pink nuclei, it will assume that tumor cells have

pink nuclei. However, if it sees a tumor cell with violet nuclei, with a large *learning rate*, it will quickly realize that the *violet nuclei* is not the most important feature of a tumor cell. With a small *learning rate*, it will assume that the violet nuclei is an outlier and that tumors are identified by pink nucleus. The higher the *learning rate* means that the network *changes its mind* more quickly. If the *learning rate* is too high it might start to think that all tumor cells have violet nuclei even though it has *seen* more tumor cells with pink nuclei that violet ones.

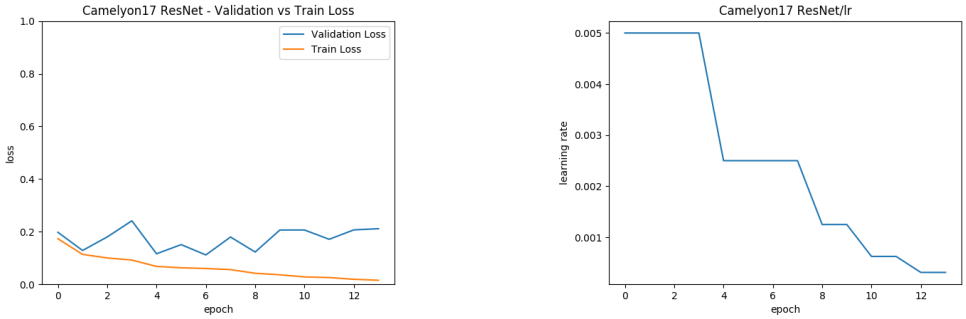
In general, we want to find a learning rate that is low enough that the network converges to something useful, but high enough that we do not have to spend too much time training it.

Normally, when the dataset has exhaustive data from both labels the learning rate can be static since it will have all the diversity it needs to converge to an output. However, when the dataset is limited, we need to take the features that we already extracted and build our model around it. So, we implemented a *learning rate adjustement* that works like this:

1. The *best loss* is initialized with infinity.
2. We initialize the *LR_DECAY* and *LR_CHANCE* variables with 0 and 2, respectively. The *LR_DECAY* is the how much the factor of how much the learning rate drops. *LR_CHANCE* works like *patience* mentioned on **early stopping** and gives the *learning rate* a chance to perform.
3. At each epoch, we check the validation loss.
4. If the validation loss is better than the *best validation loss* it becomes the new *best validation loss*.
5. If the validation loss is worse than the *best validation loss* the *LR_CHANCE* is decreased by one.
6. When the *LR_CHANCE* reaches 0, *LR_DECAY*is increased by one and *LR_CHANCE* is reseted.
7. The *actual learning rate* is calculated by the following equation:

$$actual\_learning\_rate = initial\_rate * (decay\_factor^{LR\_DECAY})$$

With this adjustment, we can make our model to converge to something useful. When the *actual learning rate* is becoming obsolete, we decrease the learning rate by a factor. Doing this, the learning rate will become smaller and smaller which means that it will become more strict when it learns, building a model around the beliefs of non-overfitting epochs. In Figure 3.13, we can see the learning rate being adjusted for the *ResNet101* model. Initially, the model reaches its optimal solution at epoch 1. Then, it gives the model 2 epochs as a chance to improve its best result. In epoch 4, the chances run out and the learning rate decreases from 0.005 to 0.0025. The model reaches its best solution at epoch 6, replacing the *best loss* from epoch 4. After that, the model starts *overfitting* and the training process stops.



(a) Validation vs Training Loss

(b) Learning Rate per Epoch

Figure 3.13: Learning Rate and Loss for *ResNet101* model

3.3.5.3 Results

As explained on the previous sections, we trained a set of Deep ConvNets with the most state-of-art models in order to determine which one performs better on our data and which limitations we have and what we can do to overcome them. We used mini-batch Stochastic Gradient Descent (SGD) as the optimizer, Binary Cross Entropy as the loss function and exponential decay mechanism for learning rate. We set the initial learning rate to 0.01 and decreased the value by half every three epochs without an improvement of the model in order to reduce oscillation and avoid divergence.

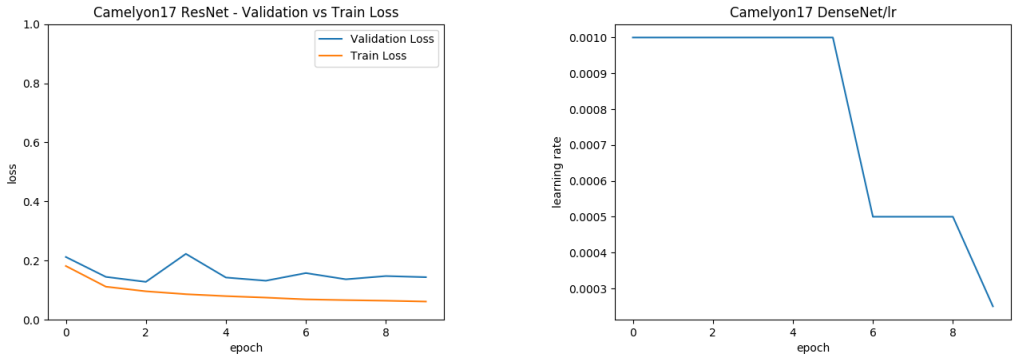
The model is trained continually using 1 GPU and the batch size depends on the model architecture due to memory limitations.

In Figure 3.14 we can see that our model reaches its solution in the second *epoch*. In the loss plot (Figure 3.14a) we can see that the validation and training loss curves converge until *epoch* 2 reaching almost the same loss, after that the model diverges on *epoch* 3 and that is when the *learning rate* is adjusted (Figure 3.14b) keeping the loss curve steep until the end of the training process.

To evaluate our image processing module impact in the classification process, we tested our models on two datasets: a dataset without any preprocessing and a dataset with the preprocessing methodology presented in Section 3.1. The results of each model performance on non-processed images can be seen in Table 3.4.

Model	Accuracy	Recall	Precision	Specificity	F1	Threshold	AUC
ResNet101	0.848	0.851	0.843	0.874	0.843	0.150	0.887
DenseNet101	0.836	0.813	0.878	0.830	0.805	0.273	0.870
ResNet50	0.835	0.847	0.815	0.817	0.852	0.370	0.883
ResNet34	0.853	0.845	0.854	0.817	0.840	0.51	0.892
ResNet18	0.889	0.898	0.848	0.835	0.872	0.38	0.887
InceptionV3	0.857	0.855	0.878	0.858	0.880	0.150	0.891

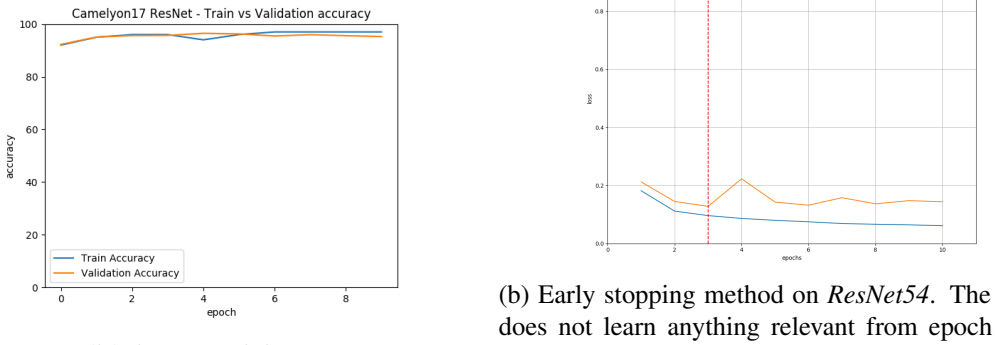
Table 3.4: Results without image processing



(a) Training versus Validation accuracy

(b) Learning Rate per epoch for *ResNet54* model

Figure 3.14: Accuracy and Learning Rate decay for *ResNet54* model



(a) Validation vs Training Accuracy

(b) Early stopping method on *ResNet54*. The model does not learn anything relevant from epoch 5 and stops. The saved model is the one with lowest loss (epoch 3)

Figure 3.15: Learning Rate and Loss for *ResNet54* model

Model	Accuracy	Recall	Precision	Specificity	F1	Threshold	AUC
ResNet101	0.884	0.892	0.880	**0.904**	0.885	0.140	0.933
Densenet101	0.873	0.841	0.901	0.870	0.830	0.090	0.920
ResNet50	0.868	0.881	0.859	0.854	0.870	0.120	0.927
ResNet34	0.897	0.888	0.897	0.898	0.887	0.340	0.935
ResNet18	0.885	0.877	0.897	0.878	0.875	0.170	0.930
InceptionV3	**0.898**	**0.895**	**0.903**	0.898	0.903	0.150	**0.941**

Table 3.5: Test Results

In Table 3.5 we can see the results of each model used on our project with the preprocessing module applied. InceptionV3 has the best results except for *specificity*, i.e. the true positive rate, meaning that ResNet101 is better on classifying normal patches and InceptionV3, with 90.1% *precision* does a better classification on *negative patches*, tumor patches. As a comparative metric, we used *Area Under Curve* (AUC). AUC combines the False Positive Rate (FPR) and the True Positive Rate (TPR) into one single metric; we first compute the two former metrics with many different threshold, then plot them on a single graph, with the FPR values on the abscissa and the

TPR values on the ordinate. The resulting curve is called ROC curve, and the metric we consider is the AUC of this curve, which we call AUROC. Analyzing the AUC for each model, we can conclude that InceptionV3 performs better on our dataset with an AUC of 94.1%.

ResNet101 results are very close to InceptionV3, having an higher *specificity*. DenseNet101 has the highest *precision* together with InceptionV3, with 90.1%.

The false positives are due to incomprehensible training data as we have extracted training patches randomly to reduce number of training samples. Because of this randomness, some difficult negatives patches from the histological mimics of cancer were missed in training data, which results in producing false positives.

We can see an improvement of 0.5% in AUC metric comparing InceptionV3 performance on pre-processed images with the results in Table 3.4. These results mean that Image processing has an impact on the model accuracy and data preparation is very important when analyzing histology slides. As we mentioned before, WSI are very rich in information and have fine-grained details. However, the quality of these images is dependant on many factors, such as lightning, tissue damage and staining. The image processing module implemented in this project helps to reduce the noise caused by external factors, normalizes images and enhances their features. Improving data quality we are improving our feature extraction process leading to a better classification.

Overall, deeper models presented a better performance among other architectures. However, due to lack of data, different mechanisms needed to be implemented to regulate our learning rate to avoid *overfitting*. Comparing the training/validation loss of ResNet54 (Figure 3.15) with ResNet101 (Figure 3.11), we can see that the curve of ResNet54 model is steeper than ResNet101.

Higher capacity models have a tendency to overfit unless we use some sort of regularizer. One way that very deep networks *overfitting* can hurt performance is that they will rapidly approach very low training error in a small number of training epochs, i.e. we cannot train the network for a large number of passes (epochs) through the dataset. A technique like Dropout, a stochastic regularization technique, allows us to train very deep nets for longer periods of time. This in effect allows us to learn better features and improve our classification accuracy because we get more passes through the training data. Using a smaller training set size, may result in learning a smaller distributed feature representation, and this may hurt the generalization ability. Lastly, if we want to be able to generalize well we would need a larger training that will allow us to learn a more diverse distributed feature hierarchy.

3.3.5.4 Heatmaps

To have a clear visualization of our results we implemented a feature that aggregates the patch-level predictions to create tumor probability heatmaps and perform post-processing over these heatmaps to make predictions for the slide-based classification task and the tumor-localization task. In the heatmap based post-processing approach, we use the tumor probability heatmap to compute the slide-based evaluation scores for each WSI.

An example of a probability heatmap can be seen in Figure 3.16.

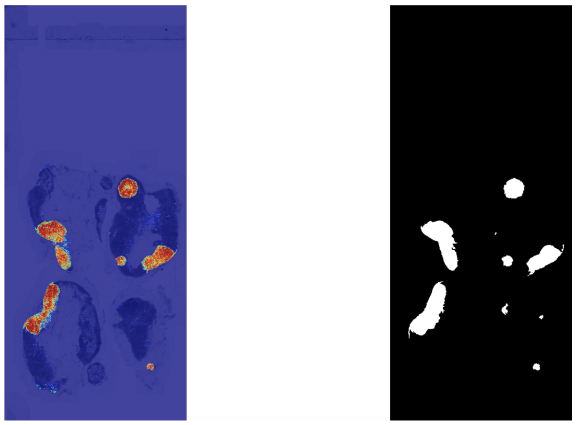


Figure 3.16: On the right image we can see a tumor binary mask and on left image the heatmap image based on the tumor probability.

This heatmap feature helps the pathologist to visualize regions of interest and can be used to improve the framework by identifying bad classifications. Furthermore, this feature can also be used to fix wrong annotations. Analyzing the heatmaps, the pathologist can identify wrong labelled zones and help to improve the segmentation process which leads to a more accurate dataset and consequently, a better classification.

Chapter 4

Oncofinder

Oncofinder is the name of the annotation tool developed to ease the pathologist task of annotate every single oncocytic cell. Oncofinder is a very light *Javascript* application and its main functionality is to label the objects that we want to identify. The web application contains a built-in segmentation model that highlights the nucleus present in the image.

Figure 4.1: Oncofinder Use Case Diagram

In Figure 4.1 we present a Use Case Diagram to illustrate the main functionalities of our web

application. In the following sections we provide a description of each of these features and of the overall process of the application.

4.1 Overall Interface

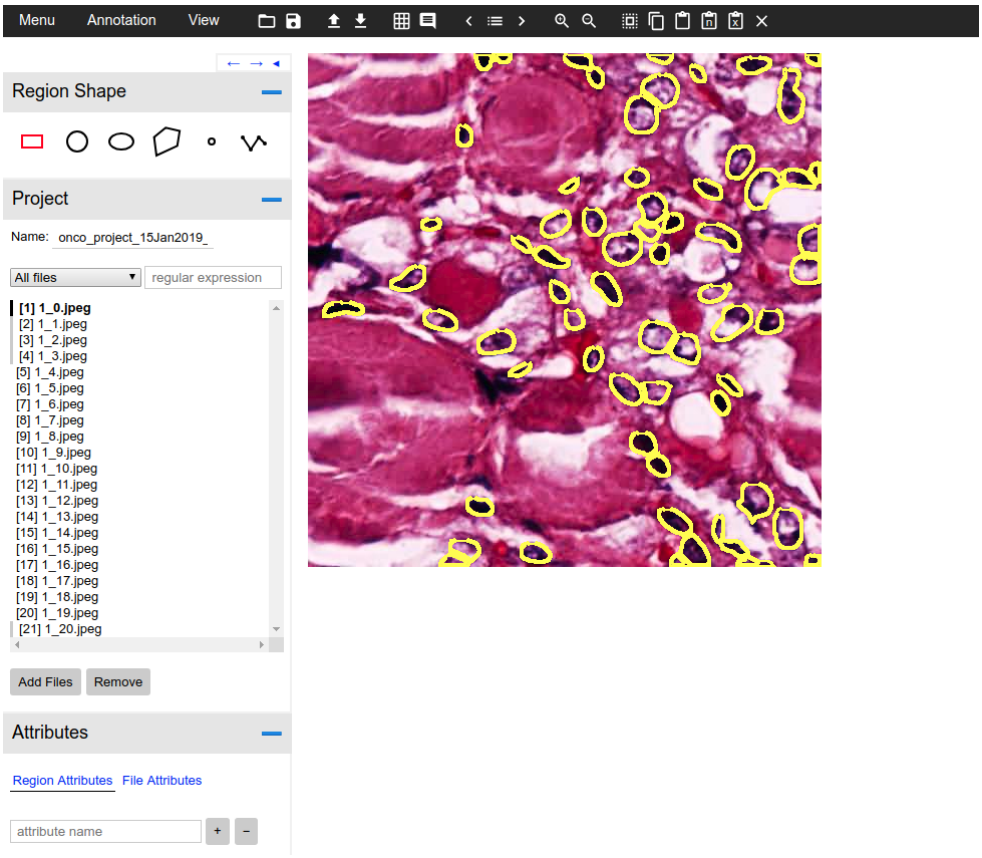


Figure 4.2: Oncofinder main page

In Figure 4.2 we can see our main page. This web application contains a single page where the User has access to all the features. The navigation bar provides a shortcut for the main actions, such as *Open/Save Project*, *Load/Import Annotations*, *Grid View*, *Next/Previous Image*, *Zoom In/Out*. The left side bar provides the submenus for file management, drawing tools and attributes creation.

4.1.1 Open Project

Open project lets the user load a previous saved project. Each project file is in JSON format and contains information regarding the project structure and appearance. The saved settings are presented below.

- UI settings

  - Annotation editor height
  - Annotation editor font size
  - Left sidebar width
  - Image height
  - Region shape fill
  - Region shape fill opacity
  - Region shape color
  - Region shape stroke width
  - Region label font
  - Region label placement
- Core
  - Buffer size
  - File path
  - Default File path
- Project
  - Project name
  - Last update

These settings are specific for each saved project and are loaded every time the User opens its respective JSON file. An example of **Open Project** can be seen in Figure 4.3.

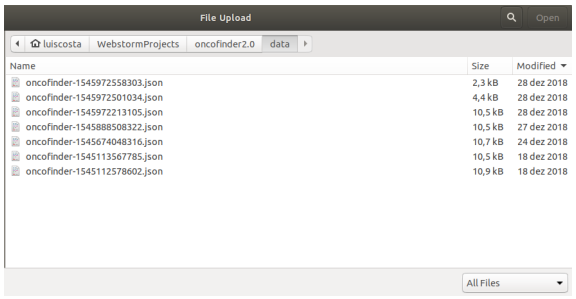


Figure 4.3: Project file upload window

4.1.2 Manage Images

Manage Images feature is an add/remove option that lets the User upload a new image to annotate or remove an image from the current workspace. When an image is removed its annotations are also deleted. In Figure 4.4 it is shown the **Files** submenu with both options and a list of current project images. The User can also search for an image or use the scroll menu to filter images using a given **option**. Options include the following filters:

- Show images without regions
- Show images missing region annotations
- Images that could not be loaded
- Regular expression

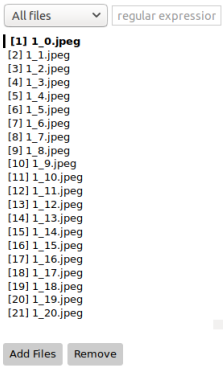


Figure 4.4: Oncofinder main page

4.1.3 Load Annotations

Load Annotations is a feature that lets the user load annotations for the current set of images in the workspace or for one single image. Consider that the user has been working on a different project and wants to load an image annotation to the current project. To address that, we created an option to load a JSON or CSV file of a specific image or project. In Figure 4.5 is shown an example of an annotation file upload.

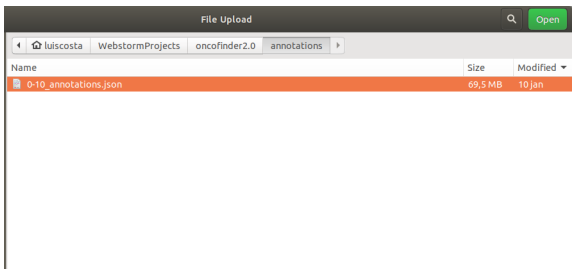


Figure 4.5: Annotations file upload window

4.1.4 Draw Section

The User has access to several draw tools present in the left upper corner of our web application (see Figure 4.2). When choosing one of the tools, the User can draw annotations in the current

image. We provide different draw shapes in order to be able to annotate different objects and make the application suitable for different problems. In Figure 4.7 we can see the Region Shape menu.

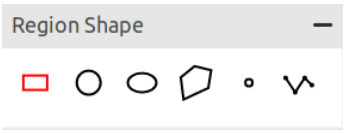


Figure 4.6: Region Shape menu. From left to right, the user can draw the in the following shapes: rectangle, circle, elliptical, polygon, points, lines.

4.1.5 Save Annotations

Save Annotations lets the User save the current image annotations or all the annotations drawn in the project. When the User is done drawing the regions, the shape coordinates are saved in a JSON file (see Section 3.1). This file is used in our framework in order to create our dataset with tumor/normal patches or binary masks of oncocytic cells.

4.1.6 Manage Attributes

Manage Attributes features are designed to create and edit Region Attributes. A region attribute is a designation for the segmentation target we are currently labelling. Each target contains an option that is the class of the object we are identifying. Consider we have a project and we want two different datasets using the same images, we can add a region attribute named *Nuceoli* where we annotate all the nucleoli in the image and we can add an attribute named *Nucleus* where we annotate the whole nucleus. This feature was also developed regarding different problems since we only have one attribute, *Oncocyte*.

For each attribute we add the labels that will be used for annotation. In this project, even though we only want to identify oncocytic cells, we can create several labels that can be used for further improvements in our framework, such as *Bad Segmentation*. Using this label we can mark wrong segmentations and improve the segmentation model presented in Section 3.2. However, we only used *Oncocyte* and *Non-Oncocyte* for this project purposes.

4.1.7 Label Image

Once the labels are created for our Region Attributes, the User can start the labelling processing and press on the objects that are already annotated and classify them or draw its own annotations. An example of the labelling feature can be seen in Figure 4.8.

4.1.8 Save Project

Save Project is a feature that lets the user save the current settings of the project creating a new local file or updating the current project JSON file.

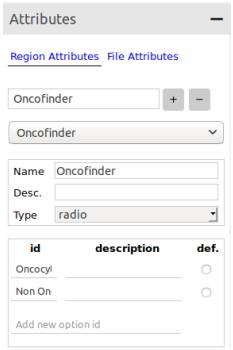


Figure 4.7: Manage Attributes submenu

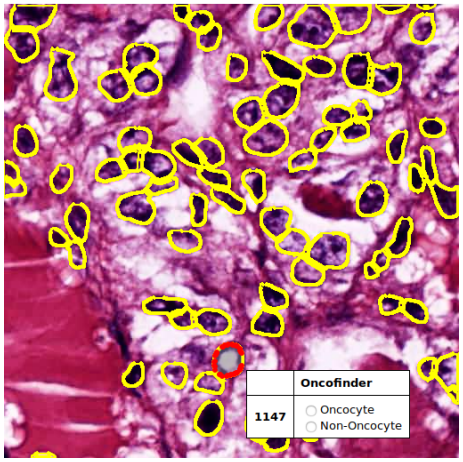


Figure 4.8: Label image interface

4.2 Chapter Summary

This web application was designed to make the annotation process easier to the pathologist. Basic operations such as *Open/Save Project* and *Import/Export Annotations* were implemented in order to keep track of the changes made in the project and use the annotations to create our framework dataset. Since we wanted to maximize our time with the pathologist, we tried to avoid a complex interface with too many tools and make the application *ready-to-use* providing a pre-created project with all the images already loaded and annotated, displaying only the necessary features for quick annotation. We concluded that the task of annotating each image with one of the drawing tools could be too exhaustive and would not be as effective as a *point by point* segmentation. With this approach, the pathologist only needs to access the web site, click on the nucleus, label and save.

Chapter 5

Conclusions and Future Work

The importance of acquiring and processing information from WSI is increasing every day, especially in the medical and biological fields. Recent scanners are able to digitize slides from biopsies at a very high magnification level which increases the amount of information present in the output images. The analysis made by pathologists is based on a set of patterns and details that are not always visible or can be damaged. These morphological characteristics can be of uttermost importance in the analysis of a slide since they can change the perception of basic concepts needed to classify the tissue. In this dissertation we addressed the problem of identifying oncocytic cells in thyroid tumors. The identification and quantification of oncocytes can help to understand the severity of the disease and improve the diagnosis [24, 10].

We now draw our conclusions, compare our results with the initial objectives determined for this project and point out future work needed to improve further the presented framework.

5.1 Objectives Fulfillment

During the thesis work we have implemented a framework that allows the efficient identification of the cell nucleus on a slide image, using state of art algorithms for object detection. We expect the tool to be a quite valuate instrument for pathologits since it eases the examination of oncocytes which is a fairly time-consuming and error-prone process. The tool provides an interface where the expert classifies each nuclei as oncocytic or not. The result of the annotation is used to build a model that outputs an heatmap with the regions of interest identified.

The automation of the segmentation step, an important step in the whole process is a very important tool since it can speed up the annotation process and its efficiency.

The image processing module developed for this project increased our solution AUC metric for 0.4% proving that using several techniques that increase image quality and enhances the attributes are of uttermost importance in the feature extraction process.

The annotation tool provides an easy and friendly interface to label microscopic images and was developed to be applied to any histologic slides. The only task required to the pathologist is to label the nucleus by clicking on the correct option provided by the interface.

The image processing and the annotation tool can be seen as part of an exhaustive preprocessing module which brings several benefits for classification part. Having an end-to-end image processing helps to normalize the features that are extracted and creates a closed system where classified images can be used for segmentation again. This is very helpful for rough patches, bad annotations and faulty segmentations.

5.2 Future Work

Future work will focus on improvements utilizing larger datasets and higher computational power. Deeper models could be used in order to extract more complex features and a lesser execution time would make the model tuning more accurate.

Staining normalization was an addition that proved to be very helpful technique in the image processing phase. Recent experiments on histopathological H&E images with high staining variations, collected from different laboratories, show that some neural networks models outperform quantitatively traditional methods, i.e. standard image processing techniques, in the measure of color constancy with at least 10-15%, while the converted images are visually in agreement with this performance improvement [43]. It would be an improvement to our framework since image degradation caused by external factors can substantially affect the feature extraction quality.

Data Augmentation is also used in this project in order to add diversity to our dataset. This feature becomes handful in the classification process since we have data limitations. Adding operations such as channel filtering, random rotations and random cropping adds new and different patches that increase the number of features and consequently, the model does not generalize so fast, helping to avoid *overfitting*.

Oncofinder, can also be improved in terms of usability. The feedback of the experts, of what they would change/add to the tool, is fundamental to improve the tool. The experience of the expert pathologists is also important to select data to train the learning classifiers, in the early stages of the tool usage, where the available manually classified data may be little or nonexistent. Working together with pathologists would also be a major advantage since there are different approaches that can help to improve our results. A constant interaction and a better understanding of the features that are more relevant can help to improve the image processing step, improving the image quality and subsequently, the segmentation.

References

[1] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep Convolutional Neural Networks. *Entropy*, 19(6), 2017.

[2] Felix Altenberger and Claus Lenz. A Non-Technical Survey on Deep Convolutional Neural Network Architectures. 2018.

[3] Tashkandi F. M. & Mohammedsaleh Z. M. Alturkistani, H. A. Histological Stains: A Literature Review and Case Study. *Global journal of health science*, 2015.

[4] Christopher M Bishop, C. M. (2006). Pattern Recognition and Machine Learning. (M. Jordan, J. Kleinberg, & B. Schölkopf, Eds.)Pattern Recognition (Vol. 4). Springer. doi:10.1117/1.2819119Bishop. *Pattern Recognition and Machine Learning*, volume 4. 2006.

[5] Liat Clark. Google's artificial brain learns to find cat videos. Available at https://www.wired.com/2012/06/google-x-neural-network/.

[6] Tiago Marques Dias da Mota. Identificação e Quantificação de Células Oncocíticas em Imagens Microscópicas. 2014.

[7] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y Ng. Large Scale Distributed Deep Networks. *NIPS 2012: Neural Information Processing Systems*, pages 1–11, 2012.

[8] Yoram Duchi, John and Hazan, Elad and Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12(1532-4435):2121–2159, 2011.

[9] L Pereira et al. "somatic mitochondrial dna mutations in cancer escape purify- ing selection and high pathogenicity mutations lead to the oncocytic phenotype: pathogenicity analysis of reported somatic mtdna mutations in tumors.". *BMC Cancer 12*, page 53, 2012.

[10] Giuseppe Gasparre, Elena Bonora, Giovanni Tallini, and Giovanni Romeo. Molecular features of thyroid oncocytic tumors. *Molecular and Cellular Endocrinology*, 321(1):67–76, may 2010.

[11] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015:1440–1448, 2015.

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:2980–2988, 2017.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 International Conference on Computer Vision, ICCV 2015, pages 1026–1034, 2015.

[16] Le Hou, Dimitris Samaras, Tahsin M. Kurc, Yi Gao, James E. Davis, and Joel H. Saltz. Patch-based Convolutional Neural Network for Whole Slide Tissue Image Classification. 2015.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:2261–2269, 2017.

[18] Tomas Iesmantas and Robertas Alzbutas. Convolutional capsule network for classification of breast cancer histology images. pages 1–8.

[19] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. feb 2015.

[20] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.

[21] Alex KrizhKrizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems, 1–9.evsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[23] et al. Macenko, Marc. A method for normalizing histology slides for quantitative analysis. *ISBI'09. IEEE International Symposium on. IEEE, 2009*, 2009.

[24] Valdemar Máximo and Manuel Sobrinho-Simões. Hurthle cell tumours of the thyroid. A review with emphasis on mitochondrial abnormalities with clinical relevance. *Virchows Archiv*, 437(2):107–115, 2000.

[25] Yurii Nesterov. A Method of Solving A Convex Programming Problem With Convergence rate O(1/kˆ2). *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[26] NIH. Cancer statistics. Available at https://www.cancer.gov/about-cancer/understanding/statistics.

[27] Shengxiao Niu, Jingjing Yang, Sheng Wang, and Gengsheng Chen. Improvement and parallel implementation of canny edge detection algorithm based on GPU. *Proceedings of International Conference on ASIC*, (6):641–644, 2011.

[28] Liron Pantanowitz, Navid Farahani, and Anil Parwani. Whole slide imaging in pathology: advantages, limitations, and emerging perspectives. *Pathology and Laboratory Medicine International*, page 23, 2015.

[29] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[30] Ankit Rathi. Capsule neural networks (capsnets). Available at https://medium.com/@rathi.ankit/capsule-neural-networks-capsnets-6fc5d8071671.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. pages 1–8, 2015.

[33] Sebastian Ruder. An overview of gradient descent optimization algorithms. pages 1–14, 2016.

[34] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. sep 2014.

[35] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. (April), 2015.

[36] Richard S Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks, 1986.

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. sep 2014.

[38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015.

[39] SH Tsang. Review: Densenet - dense convolutional network (image classification). Available at https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803.

[40] SH Tsang. Review resnet - winner of ilsvrc 2015 (image classification, localization, detection). Available at https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detect

[41] J R R Uijlings, K E A Van De Sande, T Gevers, and A W M Smeulders. Selective Search for Object Recognition. 2012.

[42] Wikipedia. Oncocyte. Available at https://en.wikipedia.org/wiki/Cancer.

[43] Farhad Ghazvinian Zanjani. Histopathology Stain-Color Normalization Using Deep Generative Models. *Medical Imaging with Deep Learning*, (Midl):1–11, 2018.

[44] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833, 2014.

[45] Guangyong Zeng, Yi He, Zongxue Yu, Xi Yang, Ranran Yang, and Lei Zhang. Preparation of novel high copper ions removal membranes by embedding organosilane-functionalized multi-walled carbon nanotube. *Journal of Chemical Technology and Biotechnology*, 91(8):2322–2330, 2016.