UNIVERSITY OF THESSALY

DIPLOMA THESIS

Tissue Micro-Array Dearraying using Deep Learning & Image Registration

Author:
Christos NTONTIS

Supervisor:
Spyros LALIS

Examiners:
Nikolaos BELLAS
Gerasimos POTAMIANOS

A thesis submitted in fulfillment of the requirements for the degree of Diploma of Electrical and Computer Engineering

Volos, Greece

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Αναγνώριση στοιχείων ιστολογικών μικροπινάκων με τεχνικές βαθιάς μάθησης και εγγραφή εικόνας

Χρήστος Ντόντης

Οι Ιστολογικοί Μικροπίνακες (ΙΜ), οι οποίοι χρησιμοποιούνται στο πεδίο της ιστολογίας, συνδυάζουν τις έννοιες μικροστοιχείων πολλαπλών ιστών και DNA. Εμπεριέχουν μια συλλογή από αρκετά δείγματα ιστών τα οποία συγκεντρώνονται σε μία γυάλινη αντικειμενοφόρο πλάκα, σε συγκεκριμένες προκαθορισμένες θέσεις πάνω της, προκειμένου να επιτραπεί η πολυπλεκτική ανάλυση, μέσω επεξεργασίας πολλών δειγμάτων κάτω από τις ίδιες τυποποιημένες συνθήκες. Ωστόσο, κατά την διάρκεια της κατασκευής ενός ΙΜ, μιας και αυτή αποτελεί μια χειρωνακτική διαδικασία, τα δείγματα μπορούν να μετακινηθούν από τις θέσεις τους στη διάταξη του πλέγματος. Επιπρόσθετα, κατά την διαδικασία αυτή τα κεριά ενσωμάτωσης μπορούν να παραμορφωθούν. Κατά συνέπεια, αυτές οι στρεβλώσεις μπορούν να οδηγήσουν σε σοβαρά σφάλματα των αποτελεσμάτων της ανάλυσης όταν οι ταυτότητες των δειγμάτων δεν συμφωνούν μεταξύ των προκαθορισμένων και των παραγόμενων θέσεων. Στη παρούσα εργασία προσπαθούμε να αναπτύξουμε μια μέθοδο αποπινακοποίησης ΙΜ, η οποία εντοπίζει και ταιριάζει τα δείγματα ενός ΙΜ με το πλέγμα σχεδιασμού τους, παρουσιάζοντας στον παθολόγο κάθε δείγμα με την ταυτότητα του ώστε οι παθολόγοι να μπορούν να συσχετίζουν αβίαστα τα μεταδεδομένα τους στο αντίστοιχο δείγμα. Αρχικά, χρησιμοποιούμε τα συνελικτικά νευρωνικά δίκτυα για να ανιχνεύσουμε τους πυρήνες σε οποιαδήποτε αντικειμενοφόρο πλάκα. Επίσης, περιγράφουμε πως δημιουργήσαμε το σύνολο των δεδομένων για να τα εκπαιδεύσουμε. Στη συνέχεια, συνδυάζουμε την έξοδο του προηγούμενου βήματος και είσοδο από τον χρήστη (αριθμό αναμενόμενων πυρήνων σε μία γραμμή και μία στήλη) για να καθορίσουμε τη διάταξη του πλέγματος. Τέλος, χρησιμοποιούμε την εγγραφή εικόνας για να αντιστοιχήσουμε την παραγόμενη διάταξη πλέγματος στην αρχική εικόνα του ΙΜ, προκειμένου να καθορίσουμε την θέση του κάθε πυρήνα.

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Diploma Thesis

Tissue Micro-Array Dearraying using Deep Learning & Image Registration

by Christos NTONTIS

Tissue Microarray (TMA), which is used in the field of histology combines multi-tissue and DNA microarray concepts. It consists of a collection of several tissue samples that are assembled onto a single glass slide, according to a design grid layout, in order to allow multiplex analysis by treating numerous samples under identical standardized conditions. However, during the TMA manufacturing process the samples positions can be distorted, since this is a manual process, from the grid layout to imprecisions when assembling tissue samples and the deformation of the embedding waxes. Consequently, these distortions may lead to severe errors of assay results when the sample identities mismatched between the design and its manufactured output. In this Thesis, we try to develop a robust method for de-arraying TMA, which localizes and matches TMA samples with their design grid to present to the pathologist each core labeled, so they can associate effortlessly their metadata to the corresponding core. At first, we leverage convolutional neural networks to detect the cores on any TMA slide and how we created the dataset to train on. Then we combine the output of the previous step and input from the user (number of expected cores in a row and a column) to define the grid layout. Finally, we use Image Registration to register the produced grid layout to the original TMA slide in order to determine the position of each core.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Spyros Lalis and Christos D. Antonopoulos for giving me the chance to work on this project. Their guidance and availability were invaluable.

Additionally, part of this work was conducted at Philips Digital and Computational Pathology in Belfast. I am therefore grateful for sharing all their resources and tools. In particular, I would like to thank my mentor Ian Thompson for all the things that he taught me, Ahmed Serag for his immersive help and insight in Image Registration and Thomas Marshall for dealing with all the paperwork and allowing me to share this work.

Finally, I would like to thank my friends and family for always being there for me, their unconditional love and understanding. From the bottom of my hurt, thank you.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Paraffin blocks containing several tissues have become a major tool in surgical pathology as well as in research settings for many years. Multi-tissue blocks can be constructed by embedding several single tissue specimens in the same paraffin block.

The development of multi-tissue techniques was started at the mid-1980s in order to address the scarcity issue of diagnostic reagents and tissue samples. The pioneer work was contributed by Dr. Battifora who introduced in 1986, the multi-tumor "sausage" tissue block [1]. In this method several rods of tissue which were extracted from paraffin-embedded tissue blocks, deparaffinized and rehydrated, were put together and reparaffinized after being tightly wrapped in small intestine of small mammals like a sausage.

In 1987, Wan *et al.* conceived the punching technique [2] which used 16-gauge needle for retrieving cylinders of tissue from paraffin blocks and arraying them in a recognizable pattern. Although Wan's punching technique was a big step and is used in nearly all of today TMA techniques, its tissue pattern was not a grid one. The first multi-tissue grid pattern is described by Battifora and Mehta in their 1990's paper [3] in which tissue rods were manually aligned in a Cartesian coordinate system.

By combining the punching technique of Wan and the "checkerboard" concept of Battifora and Mehta, Kononen *et al.* invented in 1998 a machine for assembling efficiently and accurately tissue samples in grid pattern [4]. The proposed technique called "tissue microarray" (TMA) became therefore popular and widely used in most pathological laboratories. Since, in most TMA techniques, extracted tissue samples have cylindrical form, in the following we us the terms "tissue cores" or "TMA cores" to refer TMA samples.

In a TMA, assembled tissue cores are collected from different donor blocks. It is thus highly important to match them with their meta-data for further clinical or pathological analysis. To this end, a grid pattern is used to ease the localization of each TMA core. TMA manufacturing is subjected to the deformation of the designed tissue grid due to bad positioning of the cores with respect to the design. Another main source of deformation is the heat deformation of the paraffin waxes when embedding tissue cores into the recipient block. Sectioning paraffin-embedded tissue blocks with a microtome to produce multiple slides may also produce additional deformation. As a result, the design grid may suffer geometrical transformations such as translation, rotation and shearing (linear deformations) combined with dilation, distortion and random perturbations (nonlinear deformations). In addition, some cores may be lost or split into several fragmented parts. Figure 1 illustrates a typical TMA image.

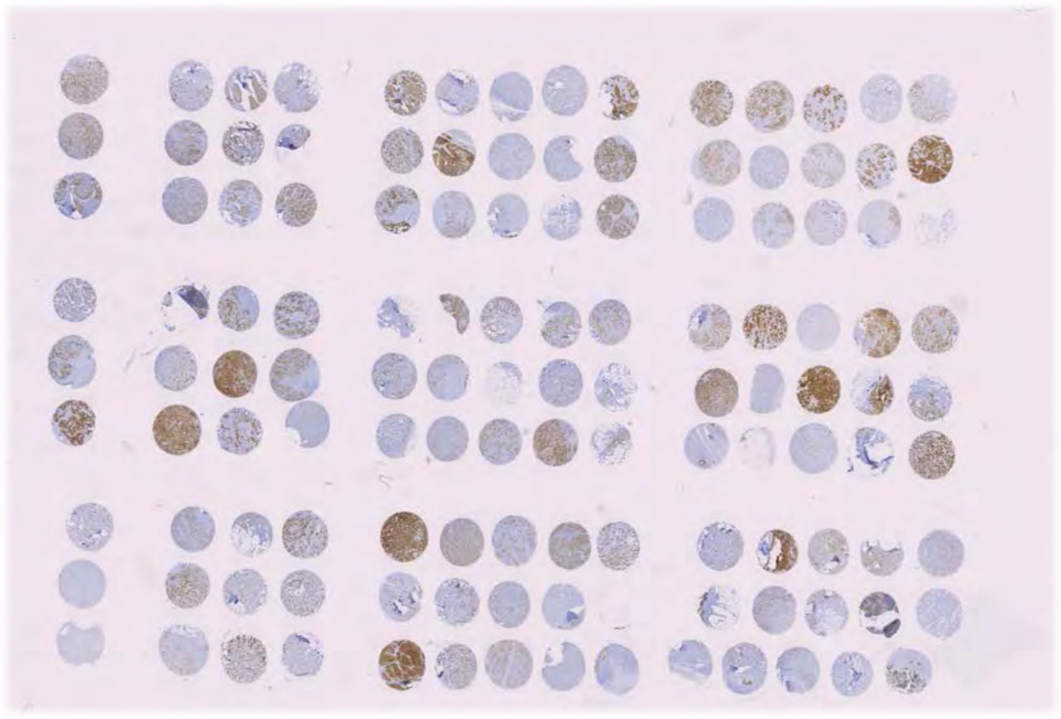

FIGURE 1.1: Typical TMA image. We can see that cores are not perfectly shaped, some of them are fragmented or even missing.

1.1 Contributions

This thesis focuses on introducing a robust de-arraying method. Our proposed method is based on CNNs and Image Registration. In particular, we employ convolutional neural networks (CNNs) to first address the task of detecting the TMA cores and then we use Image Registration techniques, both rigid and non-rigid, to match the grid layout to the CNN's output.

The contributions from this exploration and exploitation are the following:

- We create a dataset from scratch and train a CNN, while we measure its performance.
- After having some confidence in our CNN's performance, we combine information that we extract from the image and the user to create the grid layout.
- Finally, we use Image Registration to match the extracted grid layout and the output from our CNN in order to label each TMA core.

1.2 Thesis Structure

The rest of this Thesis is organized as follows:

Chapter 2 provides background on Deep Learning, specifically on convolutional neural networks that we are going to use and Image Registration, where we will describe the rigid and non-rigid registration.

Chapter 3 presents the data pre-processing and the neural network architecture used.

Chapter 4 describes the creation of the grid layout.

Chapter 5 introduces the use of Image Registration, and in particular rigid and non-rigid transformations, in order to match the output of the CNN and the grid layout.

Chapter 6 focuses on presenting the results of our implementation.

Chapter 7 concludes this thesis by discussing our key findings and by presenting some directions for future work.

Chapter 2

Background

2.1 Deep Learning

Deep learning is a subset of the field of machine learning, which is a subfield of Artificial Intelligence. In computer science, artificial intelligence, sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals. Colloquially, the term “artificial intelligence” is used to describe machines that mimic “cognitive” functions that humans associate with other human minds, such as “learning” and “problem solving” [5]

Deep Learning, which belongs to the field of Machine Learning or Statistical Learning, uses statistical methods in order to “learn” from data and generalize to new, unseen examples.

2.1.1 Artificial Neural Networks (ANN)

ANN are inspired by the brain, which contains roughly 85 billion neurons, each connected to many other. As in the brain, neurons are stimulated by inputs and pass on some, but not all, information they receive to other neurons, often after some transformation. Neurons can be trained to pass forward only signals that are useful in achieving the higher-level goals of the brain; we can train neural networks to do the same thing.

An ANN is composed of neurons organized in layers. They consist of input and output layers, as well as a number of hidden layers consisting of units (neurons) that transform the input into something that the output layer can use (Figure 2.1)

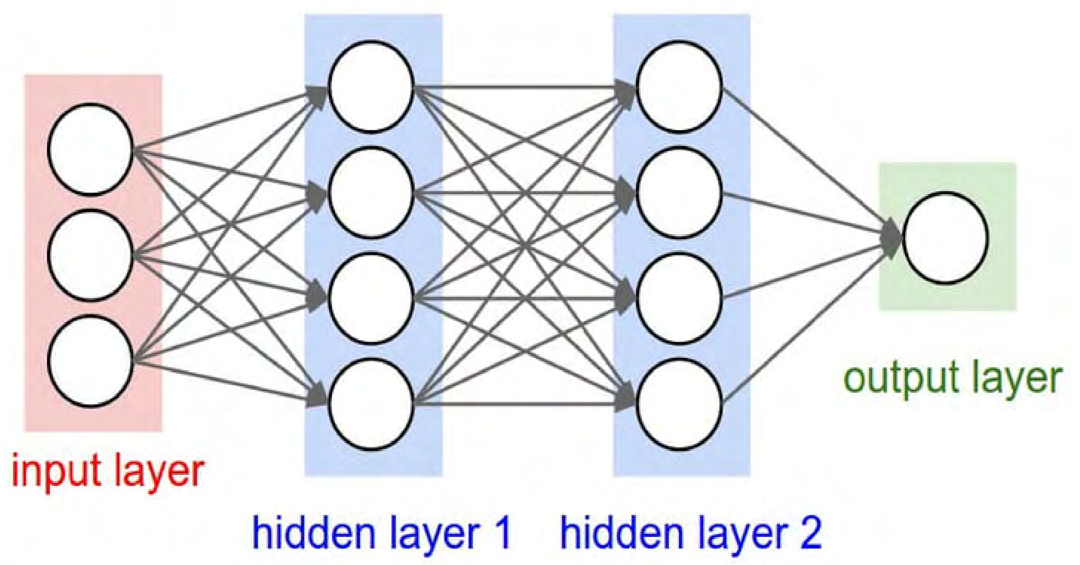

FIGURE 2.1: A Simple Neural Network [6]

The elements of each unit are:

- Inputs (x)
- Weights (w)
- Activation function (α). For example:

  Sigmoid: $y = \frac{1}{1+e^{x}}$

  ReLU: $y = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$

  Usually sigmoid is used in output units, while ReLU in all the hidden units due to the *Vanishing Gradient Problem* [7].

- Output (y) which is calculated as: $y = \alpha(w^{T}x)$

2.1.2 How an ANN Works During Training

During training, edge coefficients are optimized to minimize the prediction error. That said, ANNs use an optimization function, which aims at finding the optimal edge coefficient values with as few iterations as possible, a very common choice being stochastic gradient descent. For the optimization function to properly adjust the coefficients, it is crucial to know how good the predictions of the network are. For that purpose, a loss function is used, which quantifies the extent of a misprediction. The larger the loss function value, the larger the error between the correct value and the predicted one.

Specifically, the following iterative steps are followed:

- At first, a forward pass of the network is made, using a single sample to make the prediction. This means that an input vector is fed to the neural network and propagates through the neurons, layer by layer, until eventually reaching the output layer. There, the final prediction value of the network is computed. Finally, the error between the network prediction and the correct value is calculated, using the loss function.
- Once the prediction error is known, a backward pass of the network is performed, starting from the output layer. Then the error of each neuron is computed, which reflects its contributions to the final prediction value. These values are then used by the backpropagation algorithm to derive the gradient of the loss function. Finally, the gradient values are fed into the optimization algorithm, which adjusts the weight coefficients with the goal of a better prediction.

2.1.3 Convolutional Neural Networks (CNNs)

CNNs [8] combine three architectural ideas to ensure some degree of shift, scale and distortion invariance: local receptive fields, shared weights, and spatial or temporal subsampling. A typical convolution network is shown in Figure 3. The architecture of CNNs is analogous to that of the connectivity pattern of neurons in the human brain and was inspired by the organization of the Visual Cortex [9]. They are considered state-of-the-art solution in problems regarding images and videos.

In a CNN, each layer can be a Convolution or a Pooling one, except for the last few ones which are fully connected. (see the classification part in Figure 2.2).

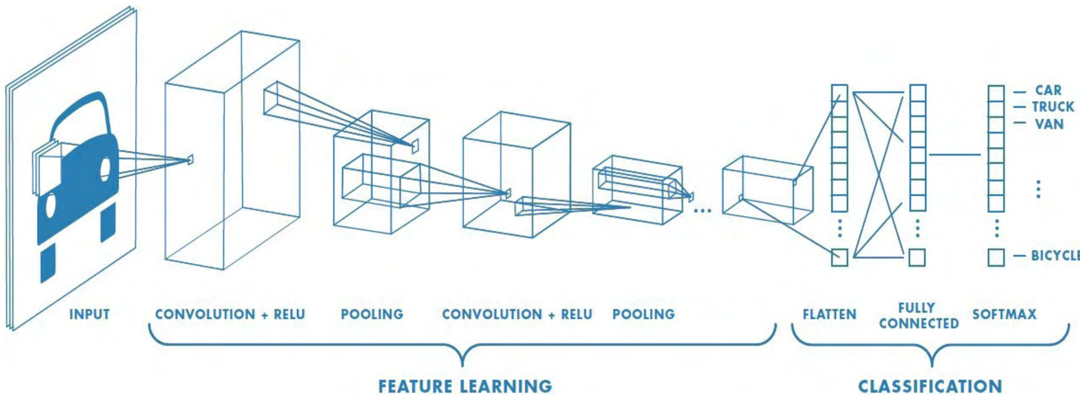

FIGURE 2.2: A Simple Convolutional Neural Network [10]

Convolution

The most important building block of a CNN is the convolutional layer. It is the element-wise product of the layer's kernel (or filter) with the input elements. The objective of the first convolutional layers is to extract the high-level features, such as edges, which are combined by the later layers creating the wholesome understanding of the input image. In Figure 2.3 we can observe an example of convolution.

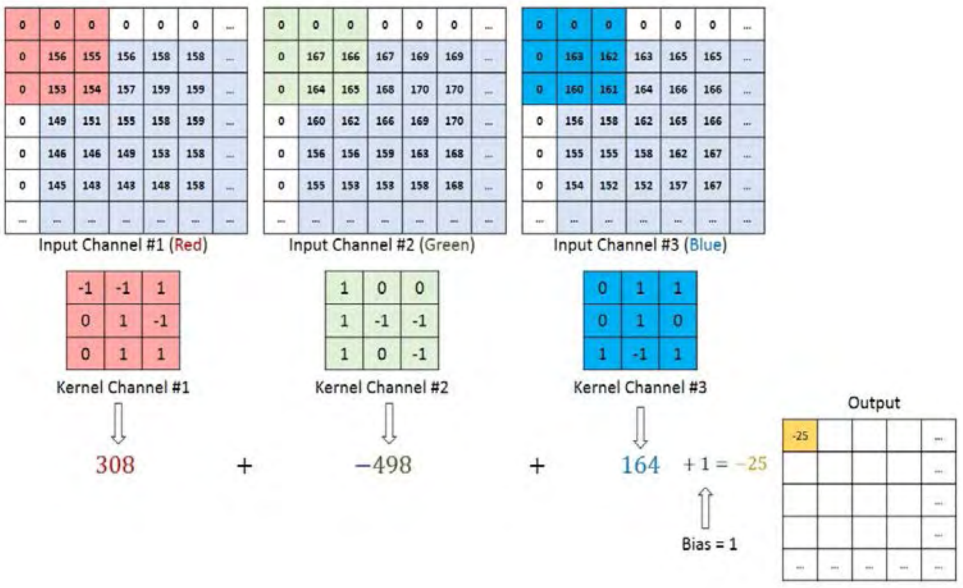

FIGURE 2.3: Matrix Convolution

Pooling

The Pooling layer is responsible for reducing the spatial size of the convolved Feature. This serves the purpose of reducing the computational power required to process the data through dimensionality reduction. Moreover, it is useful for extracting dominant features, which help to effectively train the model. There are two types of Pooling:

- **Max Pooling** returns the maximum value from the portion of the image covered by the kernel. It discards the noisy activations altogether and performs de-noising along with dimensionality reduction.

- **Average Pooling** returns the average value from the portion of the image covered by the kernel. Performs only dimensionality reduction.

Figure 2.4 shows an example of both types of Pooling.

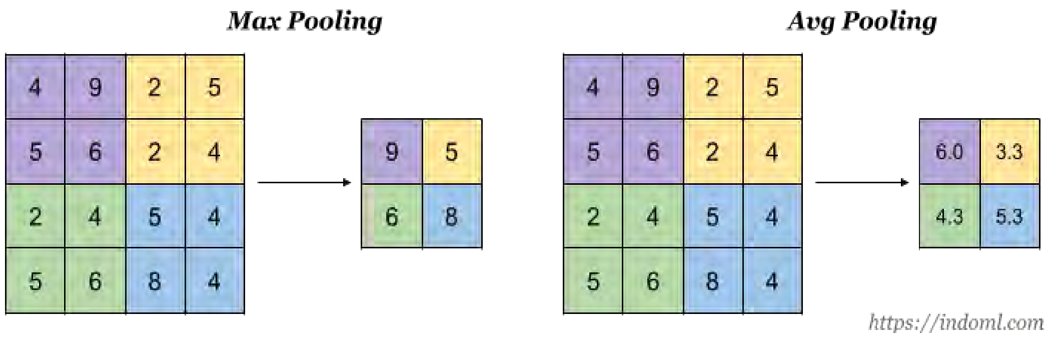

FIGURE 2.4: Max Pooling and Average Pooling with a 2x2 kernel [11]

Upsampling

The role of Upsampling is to bring back the resolution of an input to a previous resolution. There are several ways to perform such a task but for the purpose of this Thesis we will use the simplest one, which is copying an existing value to the neighboring ones. One can think of it as the reverse process of the max pooling operation. More sophisticated algorithms exist, but they introduce trainable weights which will make the model harder to train because of the added computations. For example:

$$\text{input} = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

After performing Upsampling the resulting array will be:

$$\text{output} = \begin{matrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{matrix}$$

2.2 Image Registration

Image Registration is the process of overlaying two or more images of the same scene taken at different times, from different viewpoints, and/or by different sensors. [12] Image Registration is widely used in remote sensing, medical imaging and computer vision. In general, its applications can be divided into four main groups:

- **Different Viewpoints,** images of the same scene are acquired from different viewpoints.
- **Different times,** images of the same scene were not acquired at the same time.
- **Different sensors,** images of the same scene are acquired by different sensors.
- **Scene to model registration,** images of a scene and a model of the scene are registered.

The last one is the one we will be examining and using in this thesis.

Due to different types of distortions in images it is impossible to design a universal approach to fit all registration problems. Nevertheless, most of the methods consist of the following four steps:

- **Feature detection:** salient and distinctive objects are manually or automatically detected. For further processing, these features can be represented by their center of gravity which are called control points (CP).
- **Feature matching:** the correspondence between the features detected in the sensed image and those detected in the reference image is established. A number of different feature descriptors and similarity measures along with spatial relationships among the features are used.
- **Transform model estimation:** the type and parameters of the mapping function, aligning the sensed image and the reference image are estimated. The parameters of the mapping function are computed by the means of the established feature correspondence.
- **Image resampling and transformation:** the sensed image is transformed by means of the mapping functions.

Transformation models

Image registration methods can be classified into two major categories.

- **Rigid (or linear):** including rotation, scaling and translation. They are global transformations, that means that they cannot tackle local differences between images. Figure 2.5 demonstrates some examples.
- **Non-rigid (or elastic):** transformations allow of locally wrapping the target image to align with the reference. They include radial basis functions (thin-plate or b splines, multiquadrics and compacity-supported transformations [13]), physical continuum models and large deformation models. Figure 2.6 illustrates an example.

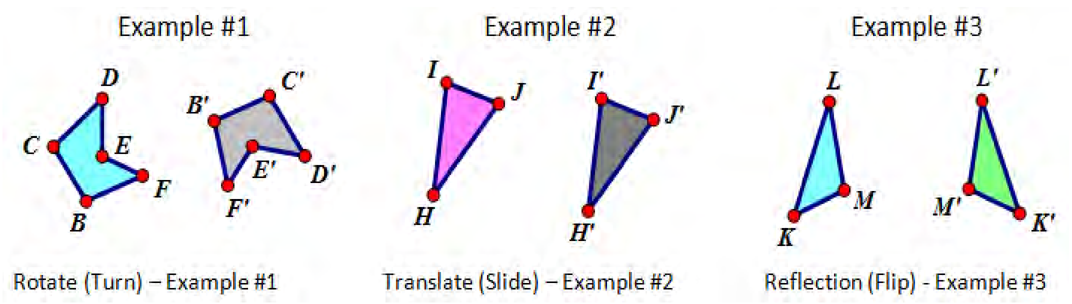
Example #1
Example #2
Example #3
D
C'
B'
C
E
E'
D'
F
F'
B
I
J
I'
J'
H
H'
L
L'
M
M'
K
K'
Rotate (Turn) – Example #1
Translate (Slide) – Example #2
Reflection (Flip) - Example #3

FIGURE 2.5: Examples of Rigid transformation [14].

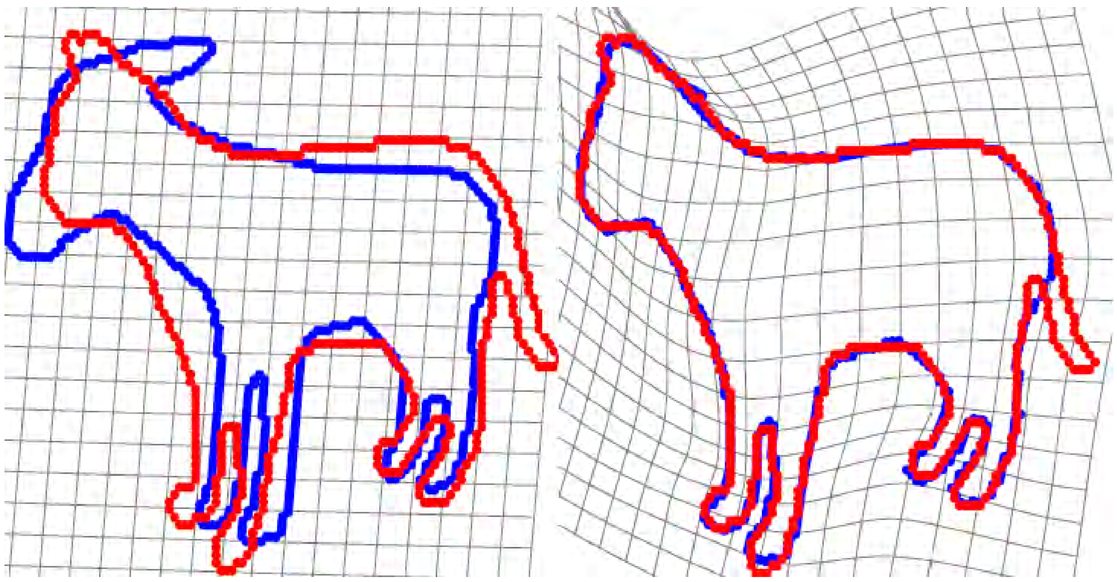

FIGURE 2.6: Example of non-rigid transformation of a donkey shape trying to be mapped to a cat shape [15].

Chapter 3

Data Pre-processing and CNN Architecture

3.1 Data Pre-processing

The size of a typical TMA slide is 1GB so a specialized software must be used in order to view and annotate them. In our case QuPath [19] was used. QuPath, is a complete solution for Pathology and is open source. A collection of 116 TMA slides was used, 106 as training set and 10 as test set. The training set slides were scanned with an Aperio [16] (svs files) and a Hamamatsu [17] (ndpi files) scanner. On the other hand, the test set was composed of slides scanned with a Philips [18] scanner (isyntax files).

The training set is composed of 13011 cores with different stains from several different laboratories, while the test set encompasses 1344 cores from the same laboratory. The size of the typical TMA core varies from 0.6 mm to 1.6 mm.

3.1.1 Annotations

Because of time constraints, not all the slides were annotated. Annotation boxes were used, with approximately, 10 TMA cores from each slide (Figure 3.1). In addition, artifacts, like stains and air bubbles were also annotated, after it was observed that the CNN was misclassifying a lot of them as cores due to their "roundish" shape (Figure 3.2), resulting in 165 annotation boxes of different size and context.

To reduce processing time, we down-sampled the resolution of the boxes by a factor of 16. To put things into perspective, the average original size of a slide is 118,000x84,520 pixels the down-sampled image will be 7,375x5,282 pixels and the extracted annotation boxes varied from 500x500 up to 2,000x2,000 pixels.

In conclusion, 3 different classes were used:

- **TMA Cores**
- **Artifacts**
- **Background**

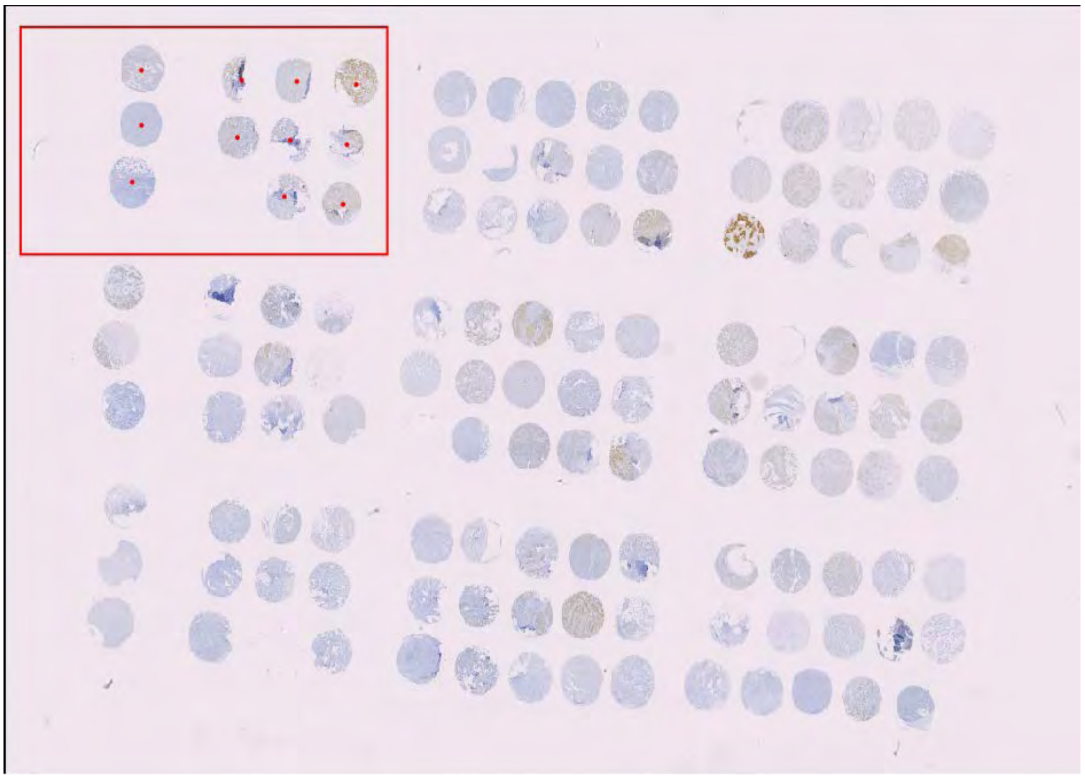

FIGURE 3.1: Example of Annotation Box in a slide.

Something to be noticed is that we did not segment the whole object (artifact or TMA core) but just the centroid. This was done because of the different sizes of the objects and, due to the fact that we wanted to provide to the CNN the bare minimum information, in order for it to be able to generalize.

Finally, the CNN architecture that we use, requires 128x128 input images, so we divide each annotation box in as many non-overlapping patches of 128x128 pixels as possible.

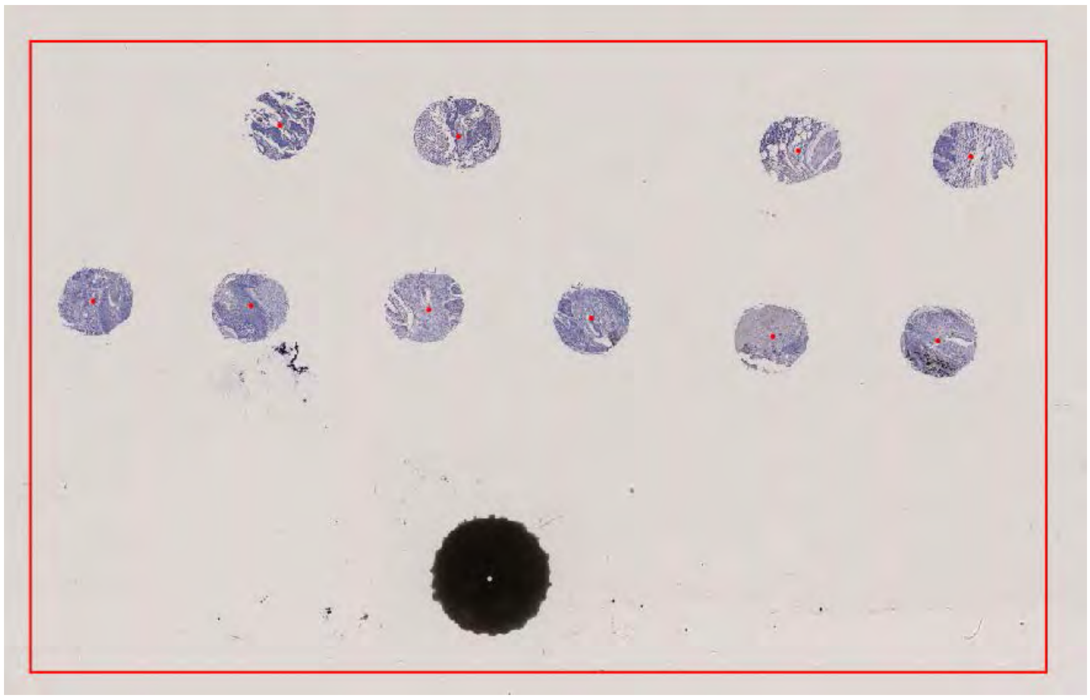

FIGURE 3.2: Annotation box including TMA Cores with red, Artifacts with green and everything else is classified as Background.

3.1.2 Augmentations

Several augmentations were used to increase the diversity of the data and help the CNN generalize better. More specifically we used:

- **Rotation** is a circular movement around the center point of the image.
- **Flipping** is mirroring an image vertically or horizontally.
- **Image Manipulation,** which includes manipulation of brightness, contrast, sharpness, saturation and gamma correction [21].

The library that we use to apply these transformations can be found here [20].

We use a function to randomly choose values for the above augmentations as well as to select how many times each augmentation will be applied (with a different value) to an image (can be from one up to six times).

3.1.3 Normalization

Data normalization is a very common and necessary data pre-processing step. Many machine learning algorithms behave significantly better if all input features are scaled to the same range, as the absolute amplitude of feature values does not affect the feature effect on the model. Two methods are well known for scaling data:

- **Normalization**, which scales all feature values to range [0,1].
- **Standardization**, which transforms all feature values to have **zero mean** and unit variance.

In our experiments, normalization proved to give better results. Given a pixel value x, then the normalized x_{new} value is:

$$x_{new} = \frac{x}{255}$$

Where 255 is the maximum pixel value in the RGB color space.

In conclusion, the final training dataset consists of 13,000 images of 128x128 pixels, which was split into 80% for training and 20% for testing.

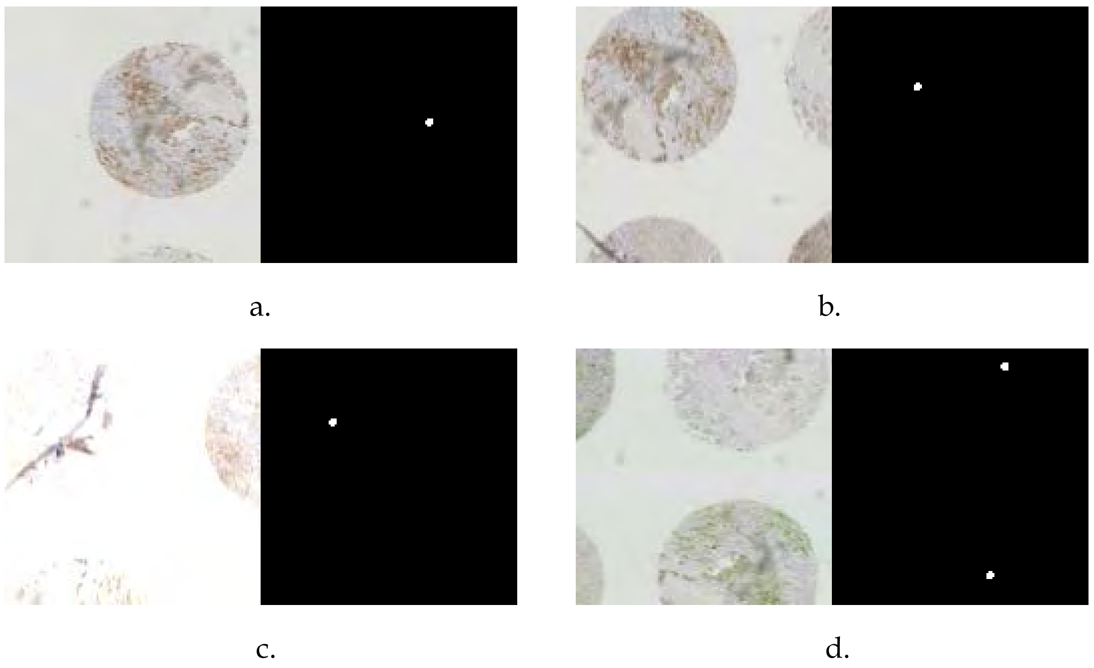

FIGURE 3.3: Examples of input patches (left) and their corresponding ground truth (right).

Figure 3.3 contains some examples of the final input that we will feed to the CNN during the process discussed in the next paragraphs. We have the input patches on the left (all of them are from the same slide) where we can also observe all the augmentations applied. On the right we can see the ground truth of each patch. This is what our CNN will try to predict, and it will use theses patches converge during training.

3.2 SegNet

SegNet is a deep encoder-decoder architecture for multi-class pixelwise segmentation. The architecture consists of a sequence of non-linear processing layers (encoders) and a corresponding set of decoders followed by a pixelwise classifier [22].

Typically, each encoder consists of a convolution layer with batch normalization [23] and a ReLU non-linearity, followed by a max pooling. The decoder has a similar architecture as the encoder with the only difference that an extra layer of upsampling exists before each convolution. For example, the SegNet, Figure 3.4, contains 5 encoders and 5 decoders and we can say that is *a depth 5 SegNet*.

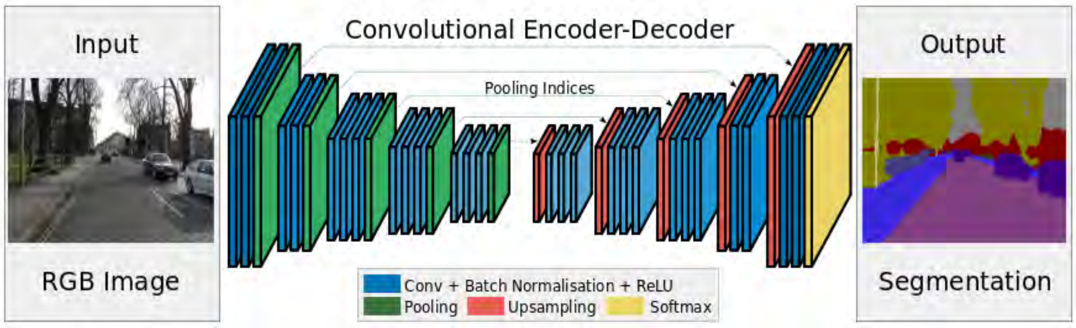

FIGURE 3.4: SegNet Architecture [22].

The novelty of this architecture is in the Subsampling stage. Max pooling is used to achieve translation invariance over small spatial shifts in the image. Subsampling results in a large input image context (spatial window) for each pixel in the feature map. . These methods achieve better classification accuracy but reduce the feature map size. This leads to lossy image representation with blurred boundaries which is not ideal for segmentation purposes. It is desired that output image resolution is same as input image. That is where the decoders and especially Upsampling comes into picture. It is necessary to capture and preserve boundary information in the encoder feature maps before sub-sampling. In order to achieve this, SegNet stores only the max pooling indices for each encoder map. The advantages of this approach are that we have less parameters and the boundary delineation is improved.

3.2.1 Optimization Method

We choose the RMSprop optimization algorithm [24], which is similar to the gradient descent algorithm with momentum [25]. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase the learning rate and the algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. Specifically, the equations for RMSprop are:

$$u(w,t) = \gamma * u(w,t-1) + (1+\gamma)\left(\nabla \mathrm{Q_i}(w)\right)^2$$
$$w = w - \frac{\eta}{\sqrt{u(w,t)}} * \nabla Q_i(w)$$

Where γ is the forgetting factor.

And for gradient descent with momentum:

$$\Delta w = a * \Delta w - \eta \nabla Q_i(w)$$
$$w = w + \Delta w$$

Where η is the step size (or equivalently learning rate). $Q(w)$ for both methods is defined as:

$$Q(w) = \sum_{i=0}^{n} (w_1 + w_2 * x_i - y_i)^2$$

and corresponds to a minimization function for a straight line $y = w_1 + w_2 * x_i$, where x_i are the training observations and y_i the corresponding responses. In our experiments we set $\eta = 0.01 \; and \; \gamma = 0.9$. Grid search was used to find the best values.

3.2.2 Loss Function

Because our dataset is unbalanced, 99% of the pixels are Background, 0.8% are Cores and the rest 0.2% are Artifacts, we use pixel-weighted cross entropy [26], where uncommon classes get a higher weight, and can be defined as follows:

$$L = -\frac{1}{N} \sum_{i} \sum_{j} \frac{1}{\sqrt{f_j}} p_{ij} log p'_{ij}$$

Where *i* is the pixel index, *j* is the class index, f_j is the frequency for class *j*, p_{ij} is the true probability for class *j* for pixel *i*, p'_{ij} is the predicted probability for class *j* for pixel *i* and *N* is the total number of pixels. In other words, we tell the model that the Core and Artifact pixels are more important, and that more emphasis should be placed on them when updating the weights.

3.2.3 Proposed Model Architecture

The SegNet that we used is a depth 4 one, with 128x128 input size. Several depths were tried but the one that performed best was this one. Table 3.2 provides a detailed description of the architecture, including the encoder – decoder pattern. Specifically, we can see that from the 128x128 input size we reach to a feature map of 8x8 which the decoders scale back to the input resolution. The total number of parameters is 21,992,707. Table 3.1 outlines the hardware used to train this model.

Component	Specification
OS Version	Windows 10
Processor Threads	6 Cores, 2-way hyperthreading, 12 threads
Processor Clock	3.6 GHz+
Memory	32 GB
GPU	NVIDIA GFORCE GTX 1080 TI, 11 GB GDDR5

TABLE 3.1: Hardware specification used in training

Layer type	Output Shape	Number of Parameters
Input	128, 128, 3	0

Convolution	128, 128, 64	1792
Batch Normalization	128, 128, 64	256
Max Pooling	64, 64, 64	0
Dropout	64, 64, 64	0
Convolution	64, 64, 128	73856
Batch Normalization	64, 64, 128	512
Max Pooling	32, 32, 128	0
Dropout	32, 32, 128	0
Convolution	32, 32, 256	295168
Batch Normalization	32, 32, 256	1024
Max Pooling	16, 16, 256	0
Dropout	16, 16, 256	0
Convolution	16, 16, 512	1180160
Batch Normalization	16, 16, 512	2048
Max Pooling	8, 8, 512	0
Dropout	8, 8, 512	0
Convolution	8, 8, 1024	4719616
Batch Normalization	8, 8, 1024	4096
Dropout	8, 8, 1024	0
Convolution	8, 8, 1024	9438208
Batch Normalization	8, 8, 1024	4096
Dropout	8, 8, 1024	0
Upsampling	16, 16, 1024	0
Convolution	16, 16, 512	4719104
Batch Normalization	16, 16, 512	2048
Dropout	16, 16, 512	0
Upsampling	32, 32, 512	0
Convolution	32, 32, 256	1179904
Batch Normalization	32, 32, 256	1024
Dropout	32, 32, 256	0
Upsampling	64, 64, 256	0
Convolution	64, 64, 128	295040
Batch Normalization	64, 64, 128	512
Dropout	64, 64, 128	0
Upsampling	128, 128, 128	0
Convolution	128, 128, 64	73792
Batch	128, 128, 64	256

Normalization		
Convolution	128, 128, 3	195
Activation	128, 128, 3	0

TABLE 3.2: SegNet Architecture Summary.

Chapter 4

Grid Creation

4.2 Grid Layout Creation

As we discussed in the previous paragraph, the probability map extraction process uses a list, that the *findContours()* function returns, to extract information about the core placement. We extract the minimum and maximum x and y from that list, with which we can construct a bounding box including all core centroids. Also, the user must provide the number of cores that lie on each row and column. Combining all this information we can find the distance between each core on x and y, using the code shown in Listing 4.1.

```
import numpy as np
def find_grid__layout(min_x,min_y,max_x,max_y,slide_height,slide_width,num_rows,num_columns):
  where_x = np.linspace(min_x, max_x, num=num_cols)
  where_y = np.linspace(min_y, max_y, num=num_rows)
  grid_core_locations = np.meshgrid(where_x, where_y)
```

LISTING 4.0.1: Grid Layout Core location python script.

Since we know where the cores should have been, we create another image, similar to the one we have in Figure 4.5.b, only this time it contains the ideal location of the core centroids. Figure 4.1 demonstrates a result.

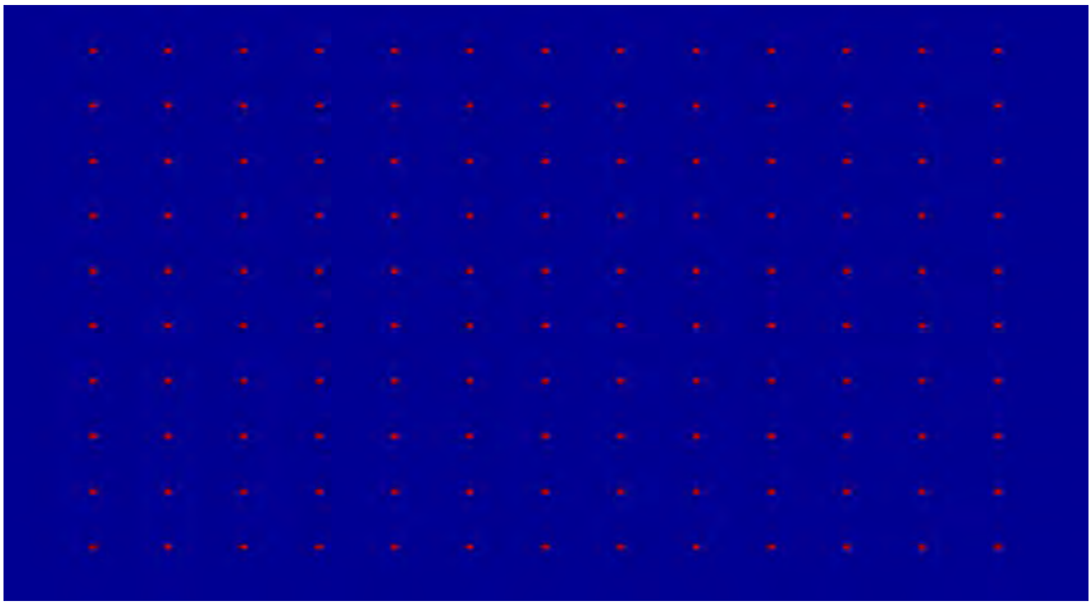

FIGURE 4.1: Grid Layout for the TMA slide in Figure 6.5.a.

We can also see that, for example, in the first row except the 2 cores that are present on the slide, the grid layout, also, contains the missing ones. This happens because the grid layouts are created blindly. Most of the times TMA slides come with an excel spreadsheet where they specify where cores exist and where they are intentionally absent. Unfortunately, we did not have in our possession this kind of information, but it is something that can easily be adopted and could improve the Image Registration algorithm.

Chapter 5

Probability Map and Grid Layout Matching

In the previous chapters we have achieved the detection of the TMA core centroids using a CNN and the grid layout creation by combining information from a user and core detection. In this chapter we will present the last step, which is the actual labeling of the cores. To achieve this, we are proposing the use of Image Registration which will help us match the grid layout to the actual image (probability map). We will use two types of transformations. In the beginning, we are applying a rigid transformation and more specifically rotation, followed by a free-form deformation (FFD) based on B-splines which is a non-rigid transformation. The goal of image registration in our problem is to relate any point in the grid layout to the probability map, i.e., to find the optimal transformation T: $(x, y) \rightarrow (x', y')$ which maps any point in the moving image $I(x, y, t)$ at time t into the corresponding point in the probability map. In general, the probability maps are non-rigid so that rigid transformations alone are not sufficient for the deformation correction of the grid layout. Therefore, we develop a combined transformation T which consists of both global (rigid) and a local (non-rigid) transformations.

5.1 Rigid Transformation

The global transformation describes the overall motion of the fixed image, and is parameterized by 6 degrees of freedom, describing the rotation and translation of the image.

$$T_{rigid}(x, y) = R * \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tr_{11} \\ tr_{21} \end{pmatrix}$$

Where the coefficients θ parameterize the 6 degrees of freedom of the transformation,
tr_{11} and tr_{21} correspond to the translation and R is the rotation matrix and is equal to:

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

In order to apply the rotation to the grid layout (moving image) we need to compute it on the fixed image.

Slide Rotation

Let core$_1$ have (x_1, y_1) coordinates and core$_2$ (x_2, y_2). For core$_2$ to be the right hand-side neighbor of core$_1$, all following statements must apply:

$$x_2 > x_1$$
$$x_2 - x_1 < 2 * d$$
$$\|y_2 - y_1\| < d$$

where d is the core diameter. If all the statements apply, then we compute the angle between the two cores:

$$angle = \tan^{-1}(y_2 - y_1, x_2 - x_1)$$

We go through this process for all the detected cores and we finally select the median value of the computed angles because of the skewed distribution of the data (cores might be missing or misplaced).

Core size

In order to compute the slide rotation, we also need the size of the core. To compute it, we are going to use OpenCV and mainstream computer vision techniques to achieve it. Listing 5.1, contains the code used. Given a slide, we convert it to grayscale and then smooth it using a Gaussian filter. We then perform edge detection along with a dilation and erosion to close any gaps in between edges in the edge map. Then we call find contours (i.e., the outlines) that correspond to the objects in our edge map. These contours are then sorted from left-to-right. We start looping over each of the individual contours. If the contour is not sufficiently large, we discard the region, presuming it to be noise left over from the edge detection process. Provided that the contour region is large enough, we compute the rotated bounding boxes of the objects. Then we unpack our ordered bounding box, then compute the midpoint between the top-left and top-right points, followed by the midpoint between the bottom-right points.
We will also compute the midpoints between the top-left + bottom-left and top-right + bottom-right, respectively. Finally, we compute the Euclidean distance between the sets of midpoints and choose the maximum between them. We return the median distance between all of the points to avoid outliers.

```
def find_core_diameter(image):
    core_diameter = []
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (7, 7), 0)
    # perform edge detection, then perform a dilation + erosion to
    # close gaps in between object edges
    edged = cv2.Canny(gray, 30, 80)
    edged = cv2.dilate(edged, None, iterations=1)
    edged = cv2.erode(edged, None, iterations=1)
    cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    (cnts, _) = contours.sort_contours(cnts)
    for c in cnts:
        if cv2.contourArea(c) < 300:
            continue
        # compute the rotated bounding box of the contour
        orig = image.copy()
        box = cv2.minAreaRect(c)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")
        box = perspective.order_points(box)
        # unpack the ordered bounding box, then compute the midpoint
        # between the top-left and top-right coordinates, followed by
        # the midpoint between bottom-left and bottom-right coordinates
        (tl, tr, br, bl) = box
        (tltrX, tltrY) = midpoint(tl, tr)
        (blbrX, blbrY) = midpoint(bl, br)
        (tlblX, tlblY) = midpoint(tl, bl)
        (trbrX, trbrY) = midpoint(tr, br)
        dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
        dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
        core_diameter.append(max(dA, dB))
    return math.ceil(np.median(core_diameter))
```

LISTING 5.0.1: Python code to compute the core diameter in a slide.

5.2 Non-Rigid Transformation

The rigid transformation captures only the global motion of the image. An additional transformation is required to capture the local deformation of the slide. The local deformation of each slide may vary significantly. Therefore, it is difficult to describe the local deformation via parameterized transformations. Instead, we use a Free-form deformation (FFD) model based on B-splines [28]. The basic idea of FFDs is to deform an object by manipulating an underlying mesh of control points. The resulting deformation controls the shape of the object and produces a smooth and continuous transformation.

To define a B-spline FFD, we denote the domain of the image as $\Omega = \{(x, y)|0 \leq X, 0 \leq y < Y\}$, where X,Y are the width and height of the slide. Let Φ denote a $n_x \times n_y$ mesh of control points $\varphi_{i,j}$ with uniform spacing δ. Then, the FFD can be written as the 2-D tensor product of the familiar 1-D cubic B-spline:

1-D Cubic B-spline basis matrix: $f(t) = [t^3 \quad t^2 \quad t \quad 1] * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} * \frac{1}{6}$

If we perform the matrix multiplication the result will be:

$$f(t) = \frac{1}{6} * \begin{bmatrix} 1 - t^3 \\ 3t^3 - 6t^2 + 4 \\ -3t^3 + 3t^2 + 3t + 1 \\ t^3 \end{bmatrix} \qquad (1)$$

Then the 2-D Cubic will be:

$$T_{non\_rigid} = \sum_{l=0}^{3} \sum_{m=0}^{3} f_l(u) f_m(w) \varphi_{i+l,j+m}, \text{ where}$$

$$u = \frac{x}{n_x} - \left\lfloor \frac{x}{n_x} \right\rfloor,$$

$$w = \frac{y}{n_y} - \left\lfloor \frac{y}{\mathrm{n_y}} \right\rfloor$$

$$i = \left\lfloor \frac{\mathrm{x}}{\mathrm{n_x}} \right\rfloor - 1$$

$$j = \left\lfloor \frac{\mathrm{y}}{\mathrm{n_y}} \right\rfloor - 1$$

As f_l we refer to the lth element of the vector in *(1)*.

In contrast to thin-plate splines or elastic-body splines, B-splines are locally controlled, which makes them computationally efficient even for a large number of control points. In particular, the basis matrix of cubic B-splines has a limited support, meaning that changing a control point $\varphi_{i,j}$ affects the transformation only in the local neighborhood of that control point.

The control points act as parameters of the B-spline FFD and the degree of non-rigid deformation which can be modeled depends essentially on the resolution of the mesh of control points. A large spacing of control points allows modeling of global non-rigid deformation, while a small spacing, allows local non-rigid deformations. At the same time, the resolution of the control point mesh defines the number of degrees of freedom and, consequently, the computational complexity. For example, a 5 x 5 mesh of control points corresponds to a transformation with 20 degrees of freedom. There is a tradeoff between model flexibility and computational complexity and should be taken into account. It depends on the accuracy of the deformation we want to achieve versus the associated execution time overhead. In our case the mesh of control points that gave the best results was 6 x 6.

Correlation

To compute the similarity of the images we use correlation between pixels in the fixed image and pixels in the moving image. The correlation is normalized by the autocorrelations of both the fixed and moving images.

A more negative metric value indicates a greater degree of correlation between the fixed and moving image. This makes the metric simpler to use with optimizers that try to minimize their cost function by default.

Optimization

To find the optimal transformation we minimize a cost function associated with the parameters Φ. There are several methods that could be used but the one that worked best for our case is Limited-memory BFGS [29]. It belongs to the family of quasi-Newton methods. It is an algorithm for finding local extrema of functions, based on

Newton's method of finding stationary points of functions.

```
import SimpleITK as sitk
def ffd(fixed_image, moving_image):
    control_pts = 3
    transformDomainMeshSize = [control_pts] * moving_image.GetDimension()
    initial_transform = sitk.CenteredTransformInitializer(fixed_image,
                                                          moving_image,
                                                          sitk.Euler3DTransform(),
                                                          sitk.CenteredTransformInitializerFilter.GEOMETRY)
    registration_method = sitk.ImageRegistrationMethod()
    registration_method.SetMetricSamplingStrategy(registration_method.REGULAR)
    registration_method.SetMetricSamplingPercentage(0.1)
    registration_method.SetMetricAsCorrelation()
    registration_method.SetOptimizerAsLBFGSB(gradientConvergenceTolerance=1e-5,
                                             numberOfIterations=100,
                                             maximumNumberOfCorrections=5,
                                             maximumNumberOfFunctionEvaluations=1000,
                                             costFunctionConvergenceFactor=1e+7)
    registration_method.SetInitialTransform(initial_transform, inPlace=False)
    registration_method.SetInterpolator(sitk.sitkLinear)
    registration_method.SetOptimizerScalesFromPhysicalShift()
    final_transform = registration_method.Execute(fixed_image, moving_image)
    return final_transform
```

LISTING 5.0.2: Python function that applies the FFD transformation to a set of images.

Listing 5.2 demonstrates the implemented FFD transformation using the SimpleITK framework [30]. Given two images (fixed and moving) a mesh will be created. Next, we use the CenteredTransformInitializer to align the centers of the two images and set the center of rotation to the center of the fixed image. Sampling strategy defines how many points will be used to evaluate our method. The higher the percentage the slower the algorithm will be. We set it to 0.1 which gave the best results. We define correlation as the similarity metric and we set the Limited Memory LBFGSB as the optimizer and a linear interpolator, which we described in the previous paragraphs. It is worth mentioning that all values were chosen using grid search.

Chapter 6

Results

In Chapter 3, we implemented a method to detect the cores using a CNN, while in Chapter 5, we discussed image registration and how we used to localize and label the centroids. Now having a complete method to detect and label our cores, we evaluate its performance and discuss our observations.

6.1 TMA Core Centroid Prediction Results

Because most of the pixels in the dataset are Background and only the centroid of each core is classified as TMA core, the loss during the training is low (less than 0.1%). In order to validate that our algorithm is, indeed, working we had to go through a manual process of counting the cores on each slide and then counting the detected cores. This was a long procedure which resulted in going back and forth, tuning the algorithm and extracting more annotations to enrich and add variety to the dataset.

True Number of Cores	**Predicted Number of Cores**	**Artifacts Predicted as Cores**	**Missed Cores**
394	391	0	3
389	388	0	1
111	111	0	0
111	111	0	0
182	180	0	2
179	179	0	0
118	116	0	2
94	94	0	0
102	101	2	3
110	109	0	1
110	107	0	3
91	87	0	4

True Number of Cores	**Predicted Number of Cores**	**Artifacts Predicted as Cores**	**Missed Cores**
89	84	0	5
87	78	0	9
66	61	0	5
96	92	0	4
92	94	2	0
92	92	1	1
92	95	3	0
75	74	1	2
56	57	5	4
77	78	1	0
119	121	2	0
67	61	0	6
98	88	1	11
394	387	0	7
55	54	0	1
121	115	0	6
123	122	0	1
126	123	2	5
124	123	0	1
125	125	1	1
124	122	0	2
125	124	0	1
124	124	0	0
125	125	0	0
125	125	0	0
125	125	0	0
122	122	0	0
122	122	0	0
122	122	0	0

True Number of Cores	**Predicted Number of Cores**	**Artifacts Predicted as Cores**	**Missed Cores**
122	123	1	0
120	117	0	3
121	121	0	0
123	123	0	0
124	124	0	0
124	124	0	0
124	123	0	1
125	124	0	1
123	122	0	1
125	123	0	2
124	124	0	0
123	123	0	0
126	126	0	0
126	126	1	1
126	126	0	0
126	126	0	0
126	126	0	0
126	125	0	1
124	124	0	0
125	123	0	2
126	126	0	0
126	126	0	0
126	127	1	0
125	125	0	1
126	126	1	1
126	125	0	1
126	124	0	2
126	124	0	2
125	124	0	1

True Number of Cores	**Predicted Number of Cores**	**Artifacts Predicted as Cores**	**Missed Cores**
117	114	0	3
118	116	0	2
111	103	0	8
112	114	2	0
130	122	0	8
137	135	0	2
149	150	1	0
150	150	1	1
105	105	0	0
121	110	0	11
157	154	0	3
185	183	0	2
131	127	1	5
22	21	1	3
15	15	0	0
24	26	2	0
21	18	0	3
229	227	0	2
113	113	0	0
141	140	0	1
130	129	0	1
129	129	0	0
136	136	0	0
126	126	0	0
130	126	0	4
125	119	0	6
142	141	0	1
132	130	0	2
117	115	0	2

	True Number of Cores	Predicted Number of Cores	Artifacts Predicted as Cores	Missed Cores
	117	117	0	0
	139	139	0	0
	129	129	4	4
	138	134	0	4
	117	115	0	2
	11	10	0	1
	11	11	0	0
Summary	**13011**	**12858**	**37**	**192**

TABLE 6.1: TMA Core Detection Results for the Training set.

6.1.1 Training Dataset Results

Table 6.1 summarizes the final results of the TMA detection algorithm. Each row corresponds to a TMA slide. From the results, 98.53% of the cores where classified correctly 0.014% were missed and we had 37 artifacts that were misclassified as cores, corresponding to 0.002% Most of the missed ones where fractions of cores which even a human would be skeptical about classifying them as cores. For example, in Figure 6.1 someone would argue that these cores are highly distorted and shouldn't be included or that only one core exists in the middle or even someone would be stricter and say that there are three cores. In our annotations we consider every bit of tissue as a core and we try to detect it, sometimes successfully and sometimes not. On this particular example our algorithm detects two cores.

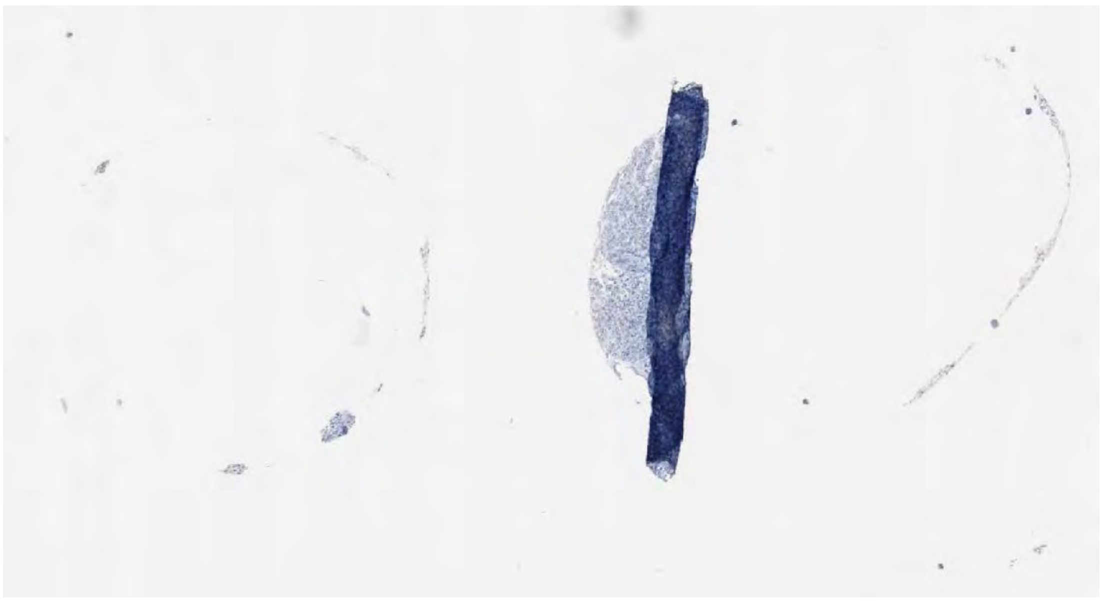
FIGURE 6.1: Example of fractions and distorted cores.

Someone would argue that the data in Table 6.1 don not add any value and we should not take them under consideration when evaluating the algorithm, since they were used in training the algorithm. We should keep in mind that only subsets of each of these slides was used in our training dataset and that the cores, even on the same slide, vary.

Accuracy	98.53%
Precision	99.71%
Recall	98.52%
F1 score	99.1%

Table 6.2: Performance metrics for the training dataset

In Table 6.2 We can see some Performance metrics for the training set.
The equations for the metrics are the following:

$$\boldsymbol{Accuracy} = \frac{true\ positives}{total\ number\ of\ elements}$$

$$\boldsymbol{Precision} = \frac{true\ positives}{true\ positives + false\ positives}$$

$$\boldsymbol{Recall} = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$\boldsymbol{F1\ score} = 2 * \frac{precision * recall}{precision + recall}$$

While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant. In our case we prefer a precision close to 1 over recall since false positive detections will affect our work in the next chapters.

6.1.2 Validation Dataset Results

Table 6.3 presents the classification results for the Validation dataset. 99.1% of the cores were classified correctly by the algorithm, 0.8% of the cores were missed and 4 artifacts were classified as cores, corresponding to 0.003%. We would like to focus more on the results in the first row, where 3 artifacts were classified as cores. The artifacts were red marker stains of round shape, which are very similar to some actual cores that we have in our dataset and that is why our algorithm misinterpreted them. Figure 6.2 contains an example slide. This could be fixed by retraining our model with examples of marker stains. Table 6.4 summarizes the accuracy, precision, recall and F1 score achieved on the validation dataset. The results are close to the training set ones, which means that our algorithm can generalize.

	True Number of Cores	**Predicted Number of Cores**	**Artifacts Predicted as Cores**	**Missed Cores**
	97	100	3	0
	96	95	0	1
	101	99	0	2
	101	101	0	2
	132	132	0	0
	112	112	0	0
	118	116	0	2
	112	111	0	1
	102	101	0	1
	110	109	0	1
Summary	**1081**	**1076**	**4**	**9**

TABLE 6.3 TMA Core Detection Results for the Validation set.

The results look very promising but further testing with unseen data is needed. Unfortunately, at the time of this writing we did not have more data for further testing. Figures 6.3 and 6.4 show some TMA core detection results.

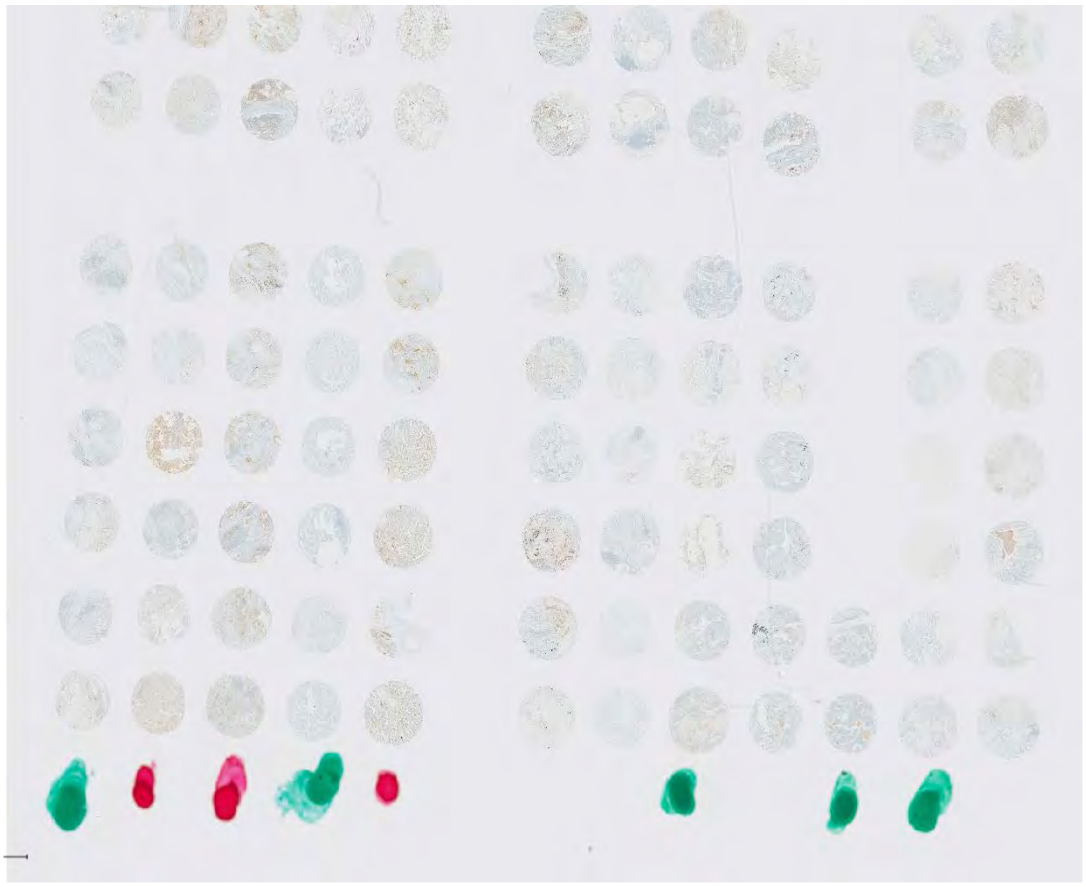
FIGURE 6.2: Slide with red and green stain.

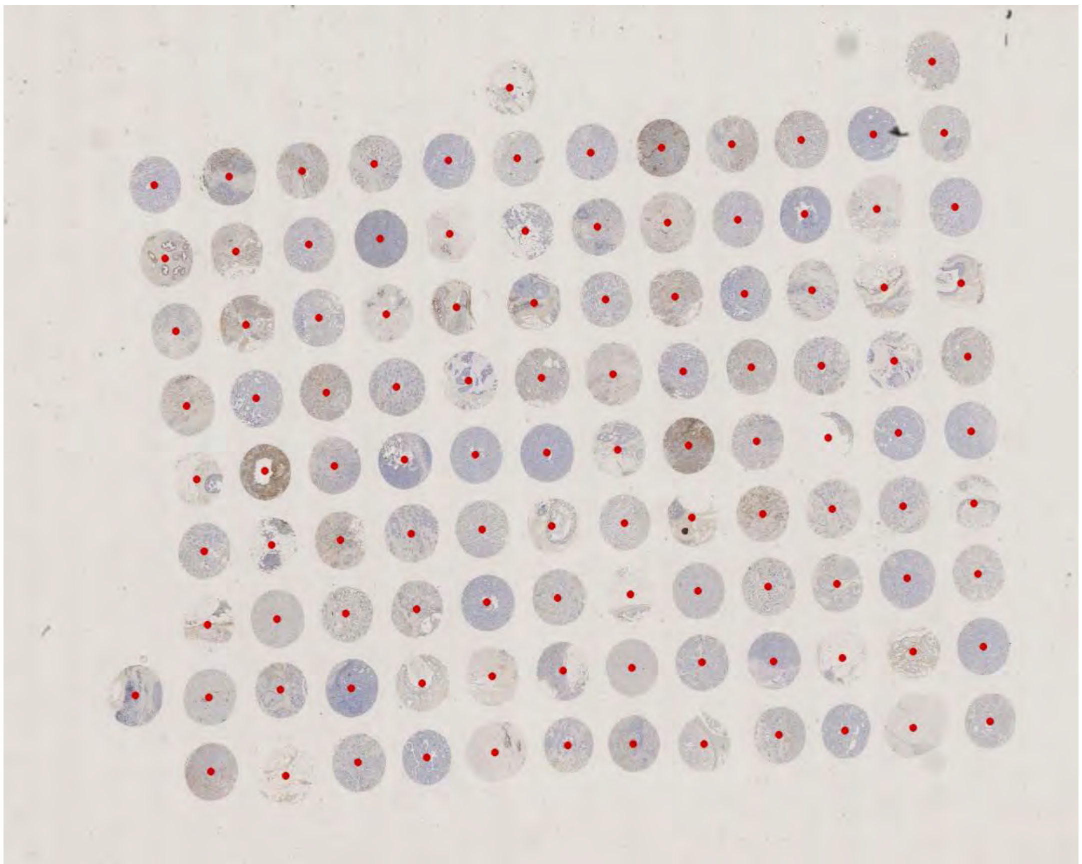
FIGURE 6.3: Example of TMA slide were all the cores were detected.

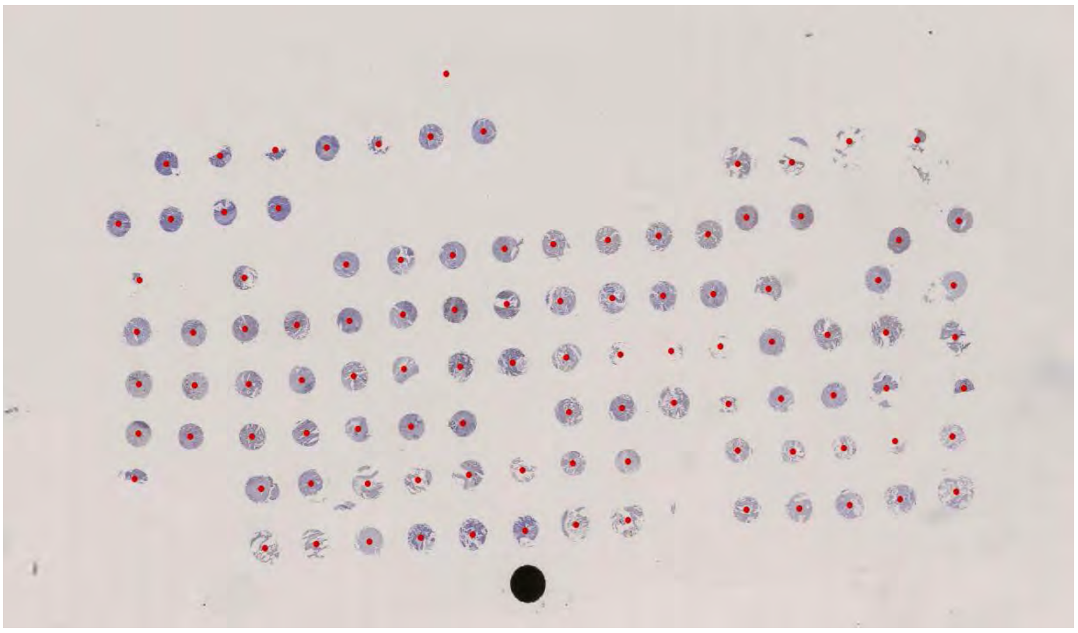
FIGURE 6.4: Example of TMA slide were an artifact was classified as a TMA core.

Accuracy	99.1%
Precision	99.62%
Recall	99.16%
F1 score	99.38%

Table 6.4: Performance metrics for the validation dataset

6.1.3 Probability Map Extraction

Before we discuss how we manufactured the grids, we should mention what we are going to use for the rest of the thesis from the TMA detection algorithm. Because the next major step is the actual labeling of the cores, as the grid layout expects, the idea is that the actual slide contains a lot of information, thus making it difficult to map the cores to the layout. What we propose instead, is to extract the probability maps which contain only the centroids of the cores and proceed with them. This does not come without a cost, since misclassifications of artifacts as cores, that are outside the "box" containing all the cores, can have a huge impact to the result. The grid layout that will be created will not align with the actual map and the Image Registration algorithms will not be able to converge. Figure 6.5 contains a slide and its corresponding output.

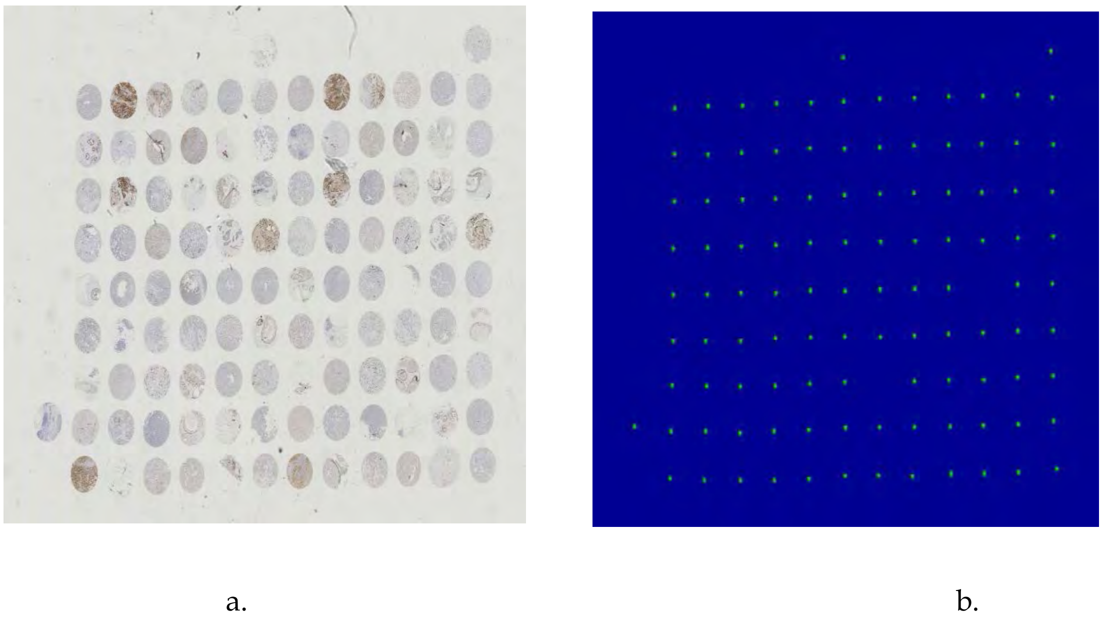

FIGURE 6.5: TMA slide (a) and its corresponding core detection output (b).

Specifically, we apply the following steps:

- Pass each slide in the TMA core prediction algorithm, which returns a probability map.
- Apply thresholding to the result, which discards the probabilities that are less than 0.6 (after manually testing on the training set, and validating on the test set), helping discard cases that the algorithm is not confident about.
- Use the *findContours()* function, provided by the OpenCV library [27], which detects the centroid of each contour and returns the x, y coordinates of each one in a list.
- Extract the height and width of the TMA slide to create a new image containing only the centroids of each core and save it on disk.

6.2 Registration Performance and Results

Since this process is in a loop involving the pathologist, it is necessary to not only be accurate but also minimize its latency. First, we need to understand, as discussed in the previous chapter, that the images we try to register represent the core centroids. This has a great impact on how we measure the accuracy of our model and what we can accept as correct. In an ideal solution we would like the cores in the moving image to be overlapping 100% with the cores in the fixed image. If the registration brings the centroids in a close distance, then the matching becomes a trivial task. Timewise, our solution has a mean time of 14.34 seconds per slide which is acceptable. We should mention that there are cases for which the registration will not work. FFD transformations assume that the images are relatively well aligned in a global sense. Image registration in general is mainly used in matching CT scans and images of the exact same object but from different angles. In Figure 6.6.a we can see an unsuccessful registration example. A lot of centroids in the fixed image (green dots) are missing or are misplaced making it impossible for our algorithm to converge. In the rest of examples in Figure 6.6 we can observe both of the registration methods that we have applied and were discussed in Chapter 5.

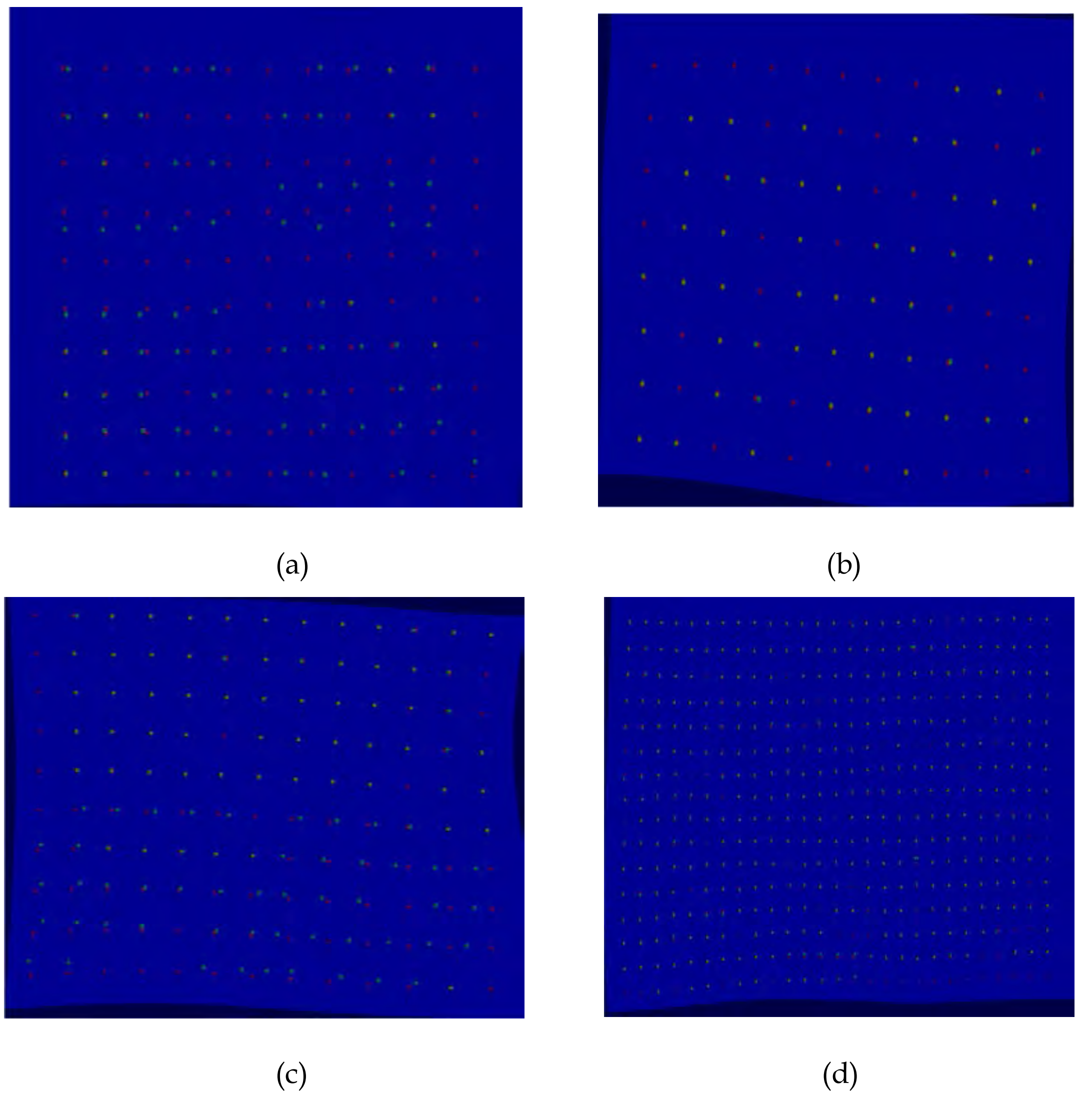

FIGURE 6.6: Examples of image registration output, when a centroid is of color khaki it means that the fixed and moving images overlap.

```
for 1 to total number of grid cores
    Calculate the distance between predicted core and each grid core. using Euclidean distance
Sort the calculated distances in ascending order based on distance values
Get top 1 row from the sorted array
Return that core
```

LISTING 6.1 Nearest Neighbor algorithm.

Finally, to label the cores, we use a nearest neighbor algorithm (Listing 6.1) to correlate each detected core to a grid core and we return a json file with the coordinates of each core with a label. Listing 6.2 demonstrates an example.

```
{
  "cores": [
    {
      "X": 12,
      "Y": 14,
      "label": "A1"
    },
    {
      "X": 20,
      "Y": 15,
      "label": "A2"
    },
    {
      "X": 26,
      "Y": 16,
      "label": "A3"
    },
    {
      "X": 10,
      "Y": 21,
      "label": "B1"
    },
    {
      "X": 21,
      "Y": 18,
      "label": "B2"
    },
    {
      "X": 30,
      "Y": 17,
      "label": "B3"
    }
  ]
}
```

LISTING 6.2: Json example of our algorithm output.

Chapter 7

Related Work

Closely similar to TMAs, DNA microarrays (also known as bio-chips) are constructed by spotting DNA probes by robots, with high precision, according to a grid pattern. Numerous gridding methods for microarrays were used to localize each DNA probes and find its row and column coordinates with respect to the designed grid. This procedure is called "de-arraying". Despite the similarity to these microarray concepts, existing "de-arraying" methods for microarrays have not been adapted for TMAs because the grids are more highly deformed. Along with the commercialization of digital imaging devices for TMA analysis over the last decade, several methods for TMA "de-arraying" have been developed [32–37]. In general terms, a "de-arraying" approach consists of two steps: (i) segmentation and localization of assembled tissue cores; (ii) array coordinate (row and column coordinates) estimation of each core.

Firstly, for segmenting tissues, existing de-arraying methods usually assume that the histogram of a TMA image is bimodal. Under this assumption, these methods perform in general a thresholding by taking the local minimum between two highest peaks corresponding to the background and the foreground, of the image intensity histogram as global threshold. Various thresholding techniques were proposed from a simple thresholding as in [32] to more sophisticated methods such as the moment-preserving thresholding in [34], the automatic thresholding based on Savitsky-Golay filtered histogram in [35] or Otsu's method used in [36, 37]. To improve the segmentation result, pre-processing like contrast enhancement transform [37] or template matching [34] was applied. Morphological operators were also used as post-processing for removing outliers in the thresholded map as in [32, 37]. However, this underlying assumption is not satisfied in case of images acquired from novel fluorescence devices because of their complex background. Due to the nature of fluorescence imaging, pixels corresponding to irrelevant objects – such as dust, glue and washing stains – in the background have often high intensities, resulting in a high peak in the intensity histogram; in contrast, the intensities of pixels corresponding to tissue cores could be relatively lower. Hence, most of cores fail to be detected with a high threshold and there is a number of outliers corresponding to a low threshold value. During the writing of this Thesis there were no other approaches that use Deep learning to detect the cores.

Secondly, for estimating row and column coordinates of each TMA core, the methods mentioned above were generally based on distance and angle criteria to define the average spacing between the cores and the orientation of the observed grid. These criteria were derived simply from the distance between neighbor tissue cores [34], or from sophisticated measures such as the histogram of distance and angle [32] or from coefficients of the Hough transform [33], or even from Delaunay triangulation [37]. To deal with the case of missing tissue cores or the design of TMA grid in which some positions are left empty [38], linear or local bilinear interpolation were used as in [32, 37] for completing the grid. Whereas these methods yield satisfactory results for further pathological analysis, they cannot produce quantitative information about the deformation of the TMA grid, which is an indicator for evaluating the quality of the manufactured input TMA. The above, in our case, are enabled by the usage of Image Registration.

Chapter 8

Conclusions

In this Thesis, we used a deep convolutional neural network to detect core centroids on a Hamamatsu, Aperio or Philips scanned slide. This is possible even when we introduce slides from vendors that the network was not trained on. However further testing from more vendors is needed to increase the confidence in our approach. Furthermore, the different artifacts that may be introduced in a slide could affect the algorithm's accuracy. Different labs have different practices. As we saw, some may use markers on the slide and the colour of them can be close to the colour of the cores, affecting the networks judgement. With the introduction of respective examples from different labs in the training set this can be addressed.

We explored Image Registration, and how we could apply it to our problem. We found some encouraging results but as we discussed it requires images that are relatively well aligned. Non-rigid transformations are mainly used for brain CT scans and images of the exact same object but from different angles. In our problem this is not the case. The images may significantly vary from each other (a lot of cores might be missing, which the current layout ignores). Maybe this can be fixed by using a more sophisticated way to produce the grid, but a mathematical approach could be a better idea for a production environment. Since we already have methods to compute core size, rotation of the slide (5.1) and we also take as input the number of expected cores in each row and column, we can combine all this information and try to localize the cores. We are currently in the process of developing such an algorithm, which will hopefully allow us to eliminate the Image Registration step altogether, resulting in lower computation time and higher quality results.

Bibliography

[1] Battifora H. “The multitumor (sausage) tissue block: novel method for immunohistochemical antibody testing.” Lab Invest. 1986;55:244-8.

[2] Wan WH, Fortuna MB, Furmanski P. “A rapid and efficient method for testing immunohistochemical reactivity of monoclonal antibodies against multiple tissue samples simultaneously.” J Immunol Methods. 1987;103:121-9.

[3] Battifora H, Mehta P. “The checkerboard tissue block. An improved multi-tissue control block.” Lab Invest. 1990;63:722-4.

[4] Kononen J, Bubendolf L, Kallionimeni A, Barlund M, Schraml P, Leighton S, Torhorst J, MIhatsch MJ, Sauter G, Kallionimeni OP, “Tissue microarrays for high-throughput molecular profiling of tumor specimens”, Nat Med. 1998;4:844-7.

[5] Russell S. and Norvig P. “Artificial Intelligence: A Modern Approach”, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[6] https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/, accessed on 20 January 2020.

[7] Hochreiter S., Bengio Y., Frasconi P., and Schmidhuber J., “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies in A Field Guide to Dynamical Recurrent Neural Networks”, Kolen J. F. and S. C. Kremer, Eds. IEEE, 2003;237-43

[8] LeCun Y. and Bengio Y. “Convolutional networks for images, speech, and time series in The Handbook of Brain Theory and Neural Networks”, M. A. Arbib, Ed. MIT Press, 1998;255-58.

[9] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional -neural-networks-the-eli5-way-3bd2b1164a53, accessed on 16 November 2019.

[10] Hubel D. H. and Wiesel T. N., “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex” Journal of Physiology (London), 1960;160:121-36.

[11] https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/, accessed on 7 November 2019.

[12] Zitova Barbara, Flusser Jan “Image Registration methods: a survey”, Institute of Information Theory and Automation Image and Vision Computing 2003;21;977-1000

[13] Goshtasby A. Ardeshir “2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications”, J.Willey & Sons,Wiley Press, 2005.

[14] http://www.geometrycommoncore.com/content/unit1/gco2/teachernotes 1.html, accessed on 16 February 2020.

[15] https://www.mathworks.com/matlabcentral/fileexchange/58987-non-rigid-registration-between-2d-shapes), accessed on February 4, 2020.

[16] https://www.leicabiosystems.com/digital-pathology/scan/, accessed on 16 March 2020.

[17] https://www.hamamatsu.com/jp/en/product/life-science-and-medical-systems/digital-slide-scanner/index.html, accessed on 16 March 2020.

[18] https://thepathologist.com/fileadmin/issues/App_Notes/0016-024-app-note-Philips__iSyntax_for_Digital_Pathology.pdf, accessed on 16 March

2020.
[19] Bankhead, P. “QuPath: Open source software for digital pathology image analysis.”, Scientific Reports, 2017;1.
[20] https://pillow.readthedocs.io/en/3.1.x/reference/, accessed on 20 November 2019.
[21] https://www.cambridgeincolour.com/tutorials/gamma-correction.htm, accessed on 16 December 2019.
[22] Badrinarayanan Vijay, Kendall Alex, Cipolla Roberto, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 2017;39:2481-495.
[23] Ioffe Sergey, Szegedy Christian, “Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, 2015, arXiv: 1502.03167.
[24] Tieleman, Tijmen and Hinton, Geoffrey, Lecture 6.5-rmsprop: “Divide the gradient by a running average of its recent magnitude”, COURSERA: “Neural Networks for Machine Learning”, 2012, accessed on February 2020.
[25] Runelhart, Hinton David E., Williams Goeffrey E., Ronald J., “Learning representations by back-propagating errors”, Nature. 1986;323: 533-536.
[26] Cui Yin, Menglin Jia Menglin, Tsung-Yi, Song Yang, Belongie Serge, “Class-Balanced Loss Based on Effective Number of Samples”, 2019, arXiv: 1901.05555.
[27] Bradski, G., “The OpenCV Library”, Dr.Dobb’s Journal of Software Tools, 2000;2236121.
[28] Lee S., Wolberg G., and Shin S. Y. “Scattered data interpolation with multilevel B-splines”, IEEE Transactions on Visualization and Computer Graphics ., 1997;3:228-44.
[29] Liu, D.C., Nocedal, J. “On the limited memory BFGS method for large scale optimization.” Mathematical Programming 1989;45:503–28.
[30] Yaniv Z., Lowekamp B. C., Johnson H. J., Beare R., "SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research", Journal of Digital Imaging, 2017;31:290-303.
[31] Rosebrock Adrian, “Measuring size of objects in an image with OpenCV”, PyImageSearch, ttps://www.pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/, accessed on 16 April 2019.
[32] Vrolijk H, Sloos W, Mesker W, Franken P, Fodde R, Morreau H, Tanke H. “Automated acquisition of stained tissue microarrays for high-throughput evaluation of molecular targets.” The Journal of Molecular Diagnostics, 2003; 5:160–7.
[33] Chen W, Reiss M, Foran DJ. “A prototype for unsupervised analysis of tissue microarrays for cancer research and diagnostics.” IEEE Trans Inform Technol Biomed. 2004;8:89–96.
[34] Dell’Anna R, Demichelis F, Barbareschi M, Sboner A. “An automated procedure to properly handle digital images in large scale tissue microarray experiments.” Comput Methods Programs Biomed. 2005;79:197–208.
[35] Rabinovich A, Krajewski S, Krajewska M, Shabaik A, Hewitt SM, Belongie S, Reed JC, Price JH. “Framework for parsing, visualizing and scoring tissue microarray images.” IEEE Transactions on Information Technology in Biomedicine . 2006;10:209–19.
[36] Lahrmann B, Halama N, Westphal K, Ernst C, Elsawaf Z, Sinn P, Bosch FX, Dickhaus H, Jäger D, Schirmacher P, Grabe N. “Robust gridding of TMAs after whole-slide imaging using template matching.” Cytometry A. 2010; 77:1169–76.
[37] Wang Y, Savage K, Grills C, McCavigan A, James JA, Fennell DA, Hamilton PW. “A TMA de-arraying method for high throughput biomarker discovery

in tissue research.” PLoS ONE. 2011;6:26007.

[38] Pilla D, Bosisio FM, Marotta R, Faggi S, Forlani P, Falavigna M, Biunno I, Martella E, De Blasio P, Borghesi S, Cattoretti G. “Tissue microarray design and construction for scientific, industrial and diagnostic use." J Pathol Inform. 2012;3:42.