



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**  
**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**  
**ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ**

**ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**  
**“ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ & ΥΠΗΡΕΣΙΕΣ”**

**Cancer Cell Metastasis Classification Using Image Processing and  
Convolutional Neural Networks**

by

Dimitrios Christos Kefalas

Submitted

in partial fulfilment of the requirements for the degree of  
Master of Advanced Information Systems

at the

UNIVERSITY OF PIRAEUS

July 2024

Thesis Supervisor: Professor Andriana Prentza

Πανεπιστήμιο Πειραιώς. Κάτοχος όλων των δικαιωμάτων  
University of Piraeus. All rights reserved.

Συγγραφέας / Author Δημήτριος-Χρήστος Κεφαλάς / Dimitrios-Christos Kefalas



## ΣΕΛΙΔΑ ΕΓΚΥΡΟΤΗΤΑΣ

**Όνοματεπώνυμο Φοιτητή/Φοιτήτριας:** Δημήτριος-Χρήστος Κεφαλάς

**Τίτλος Μεταπτυχιακής Διπλωματικής Εργασίας:** “Cancer Cell Metastasis Classification Using Image Processing and Convolutional Neural Networks”

*Η παρούσα Μεταπτυχιακή Διπλωματική Εργασία υποβάλλεται ως μερική εκπλήρωση των απαιτήσεων του Προγράμματος Μεταπτυχιακών Σπουδών “Πληροφοριακά Συστήματα & Υπηρεσίες” του Τμήματος Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς και εγκρίθηκε στις ..... [ημερομηνία έγκρισης] από τα μέλη της Εξεταστικής Επιτροπής.*

### **Εξεταστική Επιτροπή**

Επιβλέπων/ουσα (Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιώς) Ανδριάνα Πρέντζα Καθηγήτρια, υπογραφή

Μέλος Εξεταστικής Επιτροπής: Μιχαήλ Φιλιπτάκης Καθηγητής, υπογραφή

Μέλος Εξεταστικής Επιτροπής: Ανδρέας Μενύχτας Επίκουρος Καθηγητής, υπογραφή

### **ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΑΥΘΕΝΤΙΚΟΤΗΤΑΣ**

*Ο Δημήτριος-Χρήστος Κεφαλάς, γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα ότι η παρούσα εργασία με τίτλο «Cancer Cell Metastasis Classification Using Image Processing and Convolutional Neural Networks», αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές που έχω χρησιμοποιήσει, έχουν δηλωθεί κατάλληλα στις βιβλιογραφικές παραπομπές και αναφορές. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.*

*Επιπλέον δηλώνω υπεύθυνα ότι η συγκεκριμένη Μεταπτυχιακή Διπλωματική Εργασία έχει συγγραφεί από εμένα προσωπικά και δεν έχει υποβληθεί ούτε έχει αξιολογηθεί στο πλαίσιο κάποιου άλλου μεταπτυχιακού ή προπτυχιακού τίτλου σπουδών, στην Ελλάδα ή στο εξωτερικό.*

*Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου. Σε κάθε περίπτωση, αναληθούς ή ανακριβούς δηλώσεως, υπόκειμαι στις συνέπειες που προβλέπονται τις διατάξεις που προβλέπει η Ελληνική και Κοινοτική Νομοθεσία περί πνευματικής ιδιοκτησίας.*

### **Ο ΔΗΛΩΝ**

**Όνοματεπώνυμο:** Δημήτριος-Χρήστος Κεφαλάς

## **Acknowledgments**

I would like to thank my supervisor, Professor Andriana Prentza, for her guidance and recommendations regarding my thesis topic.

I express my gratitude to Dr. Zoe Kollia, a researcher at the National Hellenic Research Foundation, and Dr. Vassilios Gavriil, a research collaborator of the same Institute, for providing the image data files of metastatic/non-metastatic colorectal cancer tissues, for the scientific information concerning the Atomic Force Microscopy technique, and for clarifying the theoretical aspects on published results using variograms and theta statistics for the differentiation of metastatic/ non-metastatic colorectal cancer.

Finally, I want to thank my family for their support throughout this endeavor.

# Table of Contents

Table of Figures.....	6
List of Tables .....	7
Abstract .....	8
Περίληψη .....	9
1. Introduction.....	10
2. Literature Review .....	14
2.1 <i>Limitations in the accurate and quantified prognosis of metastatic tumor from imaging methods.</i> .....	14
2.2 <i>Basic concepts of artificial Neural Networks</i> .....	16
2.3 <i>Convolutional Neural Networks</i> .....	19
2.4 <i>DL tools for AFM-based applications</i> .....	24
3. Methodology .....	30
3.1 <i>Tools</i> .....	31
3.1.1 AFM.....	31
3.1.2 Python.....	34
3.1.3 Google Colab.....	34
3.1.4 Keras Deep Learning API.....	35
3.1.5 TensorFlow.....	36
3.1.6 Keras Applications.....	37
3.2 <i>Pipeline</i> .....	38
4. Analysis of the Workflow: limitations and solutions.....	40
4.1 <i>Dataset optimization</i> .....	40
4.2 <i>Model selection</i> .....	41
4.3 <i>Training method</i> .....	42
4.4 <i>Evaluation method</i> .....	44
5. Results .....	48
6. Discussion .....	54
7. References.....	57
8. Appendix .....	61
8.1 <i>Python code</i> .....	61
8.2 <i>User Manual</i> .....	78

## Table of Figures

<i>Figure 1: Artificial intelligence methods. Deep learning, as a subgroup of machine learning techniques, is typically applied as a type of supervised learning. Reprinted from [6]</i> .....	11
<i>Figure 2: (a) Mechanisms of Biologic Neural Network and (b) Computer Neural Network. Reprinted from [17].</i> .....	16
<i>Figure 3 :(a) Sigmoid function; (b) Input and output signals from artificial neural and biological neurons; (c) (A) Biological neural network and (B) multi-layer perception in an artificial neural network. Reprinted partly from [19].</i> .....	17
<i>Figure 4: General artificial neural network model showing the way of generating the output of each neuron. Reprinted from [15].</i> .....	18
<i>Figure 5: A model or network that can produce automatically accurate results in new examples after efficient training. Reprinted from [15].</i> .....	19
<i>Figure 6: CNN Architecture. Reprinted from [24].</i> .....	21
<i>Figure 7: Convolutional Layer Operation. Reprinted from [25].</i> .....	21
<i>Figure 8: Pooling Layer Operation. Reprinted from [26].</i> .....	22
<i>Figure 9: Fully Connected Layer. Reprinted from [28].</i> .....	23
<i>Figure 10: Left: AFM image of human CRC histological section. Right: Variograms of metastatic and non-metastatic tissue images. Above the black threshold line, tissues are non-metastatic and metastatic below the line. Reprinted from [5].</i> .....	25
<i>Figure 11: Graphical abstract on cell recognition based on AFM and modified residual neural network. Reprinted from [2].</i> .....	26
<i>Figure 12: The neural network algorithm's architecture for classifying various tissue types and identifying MM malignant cells that have invaded healthy. Reprinted from [32].</i> .....	27
<i>Figure 13: Graphics of Deep Learning (DL) analysis. Reduction of dimensionality of the AFM metastatic and non-metastatic CRC images; splitting the data into the training and testing groups; developing a DL algorithm using just the training subgroup; using the testing subgroup to perform the statistical analysis of the developed algorithm while employing cross-validation.</i> .....	31
<i>Figure 14: Schematic layout of the AFM operation principle. Reprinted from [42].</i> . .....	32
<i>Figure 15: Examples of typical <math>20 \times 20 \mu\text{m}^2</math> AFM images of (m) metastatic and (nm) non-metastatic cancer tissue used in this study. Reprinted from [5].</i> .....	33
<i>Figure 16: Keras main layers. Reprinted from [48].</i> .....	35
<i>Figure 17: TensorFlow provides libraries for Python. It makes it easier for data collection, model training, prediction serving, and future result refinement. Reprinted from [50].</i> .....	37

<i>Figure 18: InceptionV3 Architecture. Reprinted from [51].</i>	37
<i>Figure 19: A typical pipeline workflow (modified reprint from [54]).</i>	38
<i>Figure 20: Methodology Activity Diagram.</i>	39
<i>Figure 21: Preprocessed Data Sample.</i>	41
<i>Figure 22: K-Fold validation. Reprinted from [54].</i>	42
<i>Figure 23: Resulting Confusion matrix.</i>	52
<i>Figure 24: Resulting ROC Curve.</i>	53
<i>Figure 25: Generated Prediction result on the prediction dataset.</i>	53
<i>Figure 26: Uploaded data.</i>	61
<i>Figure 27: Extracted data.</i>	63
<i>Figure 28: train1.csv.</i>	64
<i>Figure 29: Console result, which provides the details of a train1_ds dataset.</i>	68
<i>Figure 30: Preprocessed data batch.</i>	71
<i>Figure 31: Model description.</i>	72
<i>Figure 32: Entire dataset sample.</i>	78
<i>Figure 33: Entire dataset sample cropped.</i>	79
<i>Figure 34: Entire dataset sample cropped, resized and grayscaled.</i>	80
<i>Figure 35: MET nomenclature.</i>	80
<i>Figure 36: Add to archive.</i>	82
<i>Figure 37: zip format.</i>	83
<i>Figure 38 : Uploading the data.</i>	84
<i>Figure 39: Run Before command.</i>	90
<i>Figure 40: h5 file location in the log.</i>	91
<i>Figure 41: Disconnecting and deleting runtime.</i>	93

## List of Tables

<i>Table 1: Machine learning techniques and their bio-applications with AFM.</i>	29
<i>Table 2: Confusion Matrix Template.</i>	45
<i>Table 3: Results under different parameter settings.</i>	49
<i>Table 4: Training parameters.</i>	51
<i>Table 5: Evaluation results.</i>	51

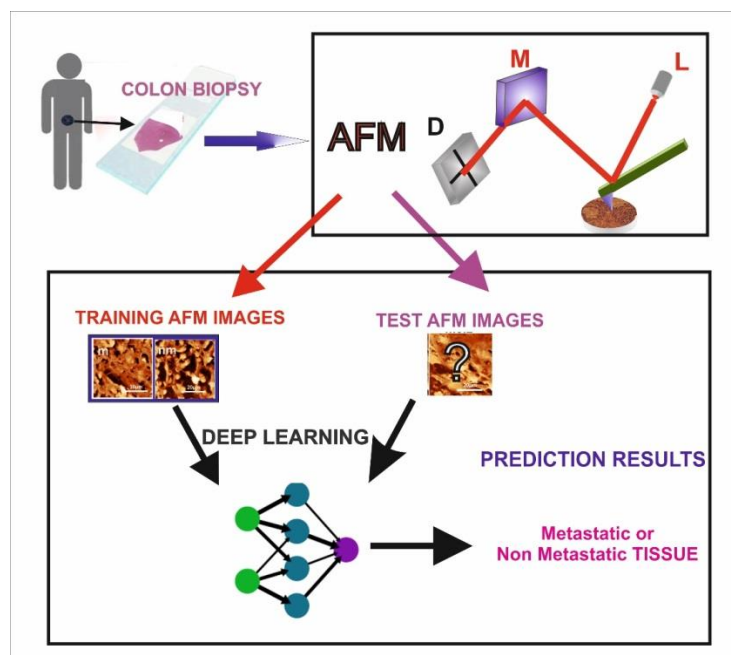
## Abstract

Cancer metastasis occurs when a certain form of cancer, known as proteogenic cancer, is spread from an original source to a different part of the body. Once spread, cancer is much harder to treat; therefore, having a reliable method to predict cancer metastasis can greatly enhance treatment strategies and help produce positive outcomes.

Deep Learning (DL) algorithms and, more specifically, the Convolutional Neural Network (CNN) architecture was employed since, at its core, this is an image classification problem. Images depicting cancer cell tissue captured by an Atomic Force Microscope (AFM) at the nanoscale are processed by the model, the result being the generation of an accurate prediction as to whether said cancer is metastatic or non-metastatic. The results mentioned above are compared to existing detection and prediction methods.

The convergence of AFM and DL represents a promising multidisciplinary approach that leverages both technologies' strengths. As research progresses, adopting these advanced methodologies in clinical practice could revolutionize early cancer detection and treatment, ultimately contributing to better patient outcomes.

*Index Terms-* Atomic Force Microscopy (AFM), Cancer Metastasis, Convolutional Neural Network (CNN), Deep Learning (DL)





## Περίληψη

Η καρκινική μετάσταση εμφανίζεται όταν μια οποιαδήποτε μορφή καρκίνου, ονομαζόμενη πρωτογενής καρκίνος, εξαπλώνεται από το αρχικό σημείο του σώματος από όπου προήλθε σε κάποιο νέο. Κύρια επίπτωση της καρκινικής μετάστασης είναι το γεγονός πως δημιουργεί επιπλοκές στη θεραπεία. Ως επακόλουθο, η εύρεση μεθόδου ικανής να προβλέψει με ακρίβεια εάν ο διαγνωσμένος καρκίνος ασθενών θα προβεί σε μετάσταση, καθίσταται ως ένα πλέον περιζήτητο εργαλείο κατά της νόσου.

Προς τη διερεύνηση τέτοιας δυνατότητας, στην παρούσα εργασία, έγινε χρήση των δυνατοτήτων των Deep Learning (DL) αλγορίθμων και συγκεκριμένα της Convolutional Neural Network (CNN) αρχιτεκτονικής.

Η λογική πίσω από την επιλογή αυτήν έγκειται στο γεγονός πως προκειμένου να εξαχθεί μια πρόβλεψη (μετάσταση ή μη), γίνεται επεξεργασία των εικόνων καρκινικού ιστού που έχουν ληφθεί με μικροσκόπιο ατομικών δυνάμεων (Atomic Force Microscope, AFM) στη νανοκλίμακα και συνεπώς το πρόβλημα ανάγεται σε πρόβλημα δυαδικής κατηγοριοποίησης εικόνων, πρόβλημα δηλαδή που βρίσκει απάντηση στα νευρωνικά δίκτυα. Τα αποτελέσματα της προσέγγισης αυτής συγκρίνονται με εκείνα από ήδη υπάρχουσες μεθοδολογίες, με την ελπίδα πως θα αποτελέσουν βοήθεια στη μάχη κατά του καρκίνου.

*Κύριοι όροι/ Keywords* - Μικροσκοπία Ατομικών Δυνάμεων/Atomic Force Microscopy (AFM), Καρκινική μετάσταση/ Cancer Metastasis, Συνελικτικό Νευρωνικό Δίκτυο/Convolutional Neural Network (CNN), Βαθιά Μάθηση/Deep Learning (DL)

## 1. Introduction

Globally, cancer metastasis is expected annually to be the cause of 19.3 million deaths [1]. Metastasis transfers cancer cells to tissues, lymph nodes, or distant organs from the primary tumor cores. For a patient to survive, metastasis must be diagnosed early. Cancer cells link themselves to other cells and the extracellular matrix during the intricate process of metastasis. It represents a crucial feature [2] of advancing malignancy into a more severe clinical condition. Developing therapeutic schemes, enhancing early cancer prognosis, and illuminating the course of cancer depend on indexing assessments of the metastatic condition and its early prediction [3]. Variable percentages of metastasis can occur at any stage, meaning that shared histology and cytological findings are essential but insufficient to recognize high-risk traits and forecast the stages of metastasis. Similarly, to increase the chances of patient survival, it is essential to determine the stage of the tumor precisely, create a general prognosis regarding the course of the disease, and discover the metastatic phase and heterogeneity of primary cancer cells as soon as possible [4].

Today, metastasis is identified mainly with optical microscopy magnetic resonance imaging (MRI) imaging techniques, with a resolution of 0.1 mm, too late for patients' survival. Atomic Force Microscopy (AFM) allows for non-destructive nanometer-scale investigation of biological samples, including cells and tissues. Recent advances in mathematical image analysis of cancer human tissues have enabled its application in cancer metastasis diagnosis and prognosis [5]. Nevertheless, analyzing data can be tedious and time-consuming, prone to user bias, and complex data analysis requires skilled personnel.

Many of these problems can be resolved with Deep Learning (DL) methods, an area of artificial intelligence (AI) and machine learning (ML), which produces quick, automated, and operator-independent analyses, *Figure 1*, [6]. DL is considered a vital technology of the Fourth Industrial Revolution. Due to its capability for data-driven learning, DL technology has roots in artificial neural networks, has gained popularity in the computer community, and finds widespread use across various fields. However, creating a reliable DL model in real-world applications is challenging because real-world conditions and data are dynamic and complex.

This thesis aims to integrate AFM of cancer tissues with DL methods to identify metastasis at a very early stage with a resolution of  $\sim 80$  nm.

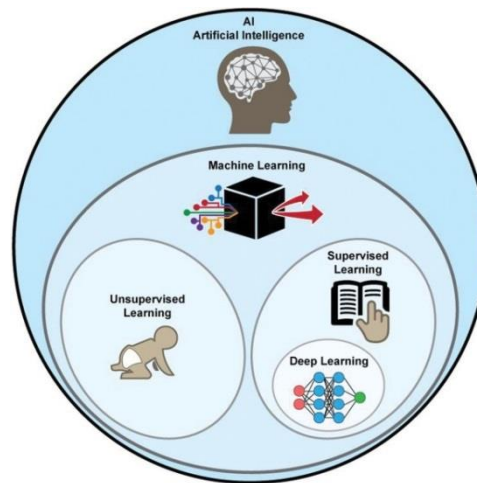


Figure 1: Artificial intelligence methods. Deep learning, as a subgroup of machine learning techniques, is typically applied as a type of supervised learning. Reprinted from [6]

Convolutional Neural Network (CNN) is a class of DL methods primarily designed to process and classify images and automatically identify pertinent structures without human supervision [7]. Similar to a traditional neural network, the structure of CNNs was modeled by neurons found in the brains of humans and other animals. In a well-known neuroscience experiment by Hubel and Wiesel, the CNN simulates the complicated cell sequence of a cat's visual cortex neuron cells [8],[9].

CNN uses the convolution process (principles from linear algebra) to extract features and identify image patterns. For example, a specific group of pixels may imply a structure like a protein in an AFM image. A convolution involves multiplying a matrix of pixels with a filter matrix or 'kernel' and then sums up the multiplication values. Next, the convolution moves to the next pixel and reiterates the same process until all the image pixels have been parsed. A CNN may include dozens, hundreds, or even thousands of layers, depending on the complexity of its intended purpose. Each layer builds on the outputs of the previous ones to identify detailed patterns. CNN has become popular in various computer vision applications, attracting interest for data in several forms, such as audio and other signal data.

This thesis used an operator-independent (unsupervised) CNN approach to analyze AFM images of colorectal tumor metastatic and non-metastatic histological sections. Specifically, to solve this problem, CNN was built and trained in Keras, a powerful and simple-to-use deep-learning library for Python. The effectiveness of training a CNN depends on the model's structure, which is decomposed into several interconnected steps combined into a workflow referred to as Pipeline (not to be confused with the Pipeline section of this thesis). These steps mainly include collecting and grouping the available data into three datasets: training, testing, and validation. Those datasets are used for training and, finally, for evaluating the model.

Training CNN is not a trivial process and should be evaluated via the testing data to determine whether it accurately predicts the status of an image outside the training dataset. Likewise, training cannot be efficient without the systematic collection of high-quality labeled data to be used for model training. The weight-sharing characteristic of CNN is the primary reason for considering it. Nevertheless, one benefit of using CNNs over other traditional neural networks is that it lowers the number of trainable network parameters, improving generalization and preventing overfitting. In the end, while adjusting the model's parameters, the validation data set objectively assesses the model's accuracy on the training data set.

Our study also compares the classification results with those of recently published work identified in higher moments of variograms on the same AFM images. The current method enables the accurate and entirely automated differentiation of metastatic cancer from non-metastatic tissues. The cell and tissue recognition method combining AFM and CNN is expected to bring new progress to cancer studies, accelerating the implementation of treatment protocols and thus having a significant societal impact.

The thesis is organized into *eight sections*, including the "*1. Introduction*" chapter that presents the topic, summarizes the existing research and the position of the thesis approach, and provides a general outline of the thesis structure.

In the "*2. Literature Review*" section, the limitations of different diagnostic methods for cancer detection versus AFM are briefly discussed; subsequent, a systematic review of previous research on combining DL algorithms and, more specifically, the CNN architecture with AFM techniques mainly focused on biosystems is included.

Following in the "3. *Methodology*" section, the current experimental approach and the implementation of CNN algorithms are presented. Specifically, every stage is explained in detail, along with the reasoning behind the choices made, including the tools used. In particular, the operating principle of the AFM microscope is outlined, the main characteristics of Python programming language are presented, and the critical features of the Google Colab platform, the Keras deep learning open-source library, and the InceptionV3 image classification model are delineated. The Pipeline steps for improving the model's performance and management, resulting in quick and easy deployment, are also presented.

The "4. *Analysis of the Workflow: limitations and solutions*" section provides detailed information regarding the methods and techniques that made the analysis possible.

The "5. *Results*" section, the key results of the model's training and its ability to make accurate predictions are presented. The best results connected with the study's goal achieved by the process analyzed in the methodology section are described via the metrics.

Finally, the "6. *Discussion*" section explains and evaluates the results and presents future research directions. The significance of parameter selection on CNN performance is highlighted, and a comparison is made with the results of relevant publications. A conclusion is included to summarize the key supporting ideas discussed throughout the work.

The "*Annex*" section entails a step-by-step code tutorial and a "*User manual*" containing operating instructions.

## 2. Literature Review

This section compares diagnostic methods for cancer detection that employ AFM versus those that do not and reviews previous research that combines the CNN architecture with AFM techniques in various sectors, including the medical field.

### ***2.1 Limitations in the accurate and quantified prognosis of metastatic tumor from imaging methods.***

On a global scale, it is estimated that metastatic solid tumors are responsible for approximately 66.7% of cancer patients' deaths [10]. A significant percentage of cancer patients display minimal symptoms of metastasis [11] if that is to occur, the latter varying depending on the migration tissue-organ. Furthermore, a method for detecting cancer patients' metastasis at its primitive stage has not yet been clinically approved. As a result, while overall cancer death rates have declined by 2.1% between 2015 and 2019, getting us closer to ultimately controlling the disease, an accessible method of early identification remains the holy grail of the medical field.

Cancer detection by standard diagnostic procedure involves pathological evaluation of tissue samples revealed by optical microscopy, optical and electronic microscopy imaging methods, positron emission tomography (PET), magnetic resonance imaging (MRI), and the analysis of tissue chemical composition by spectroscopical methods.

Histopathologists define the malignancy level for simple and fast cancer diagnoses and visually evaluate ultra-thin tissue slices under the light microscope. The ability to analyze the samples with or without staining is the pronounced advantage of light microscopy. However, this method for identification of an abnormality in tissue architecture and nuclear morphology often suffers from limitations such as low resolution (a few hundred nanometers and maximum sample magnification of approximately 1000x) due to the light-diffraction limit [12] and images being limited to two dimensions due to the small focal depth even for 4 to 6  $\mu\text{m}$  thin sections. Also, a significant parameter is that the method depends on the operator's visual interpretation, which cannot accurately distinguish metastatic and non-metastatic states.

For higher magnification and resolution, electron microscope-based methods have been proposed. However, the methods involve complicated preparation, such as vacuum conditions and sputtering with conductive materials that can easily damage bio-samples before imaging and problems in locating the area of interest already observed by the light microscope [12]. Also, it remains a costly method, which limits the clinical transformation of this technique [13]. Moreover, vacuum sample preparation alters the initial image and biological sample signatures. In contrast, the AFM imaging addresses all the aforementioned constraints [5].

The AFM is a non-optical, non-destructive surface analysis technique that uses a probe (tip) to record the surface morphology of samples with atomic resolution in different environmental conditions (e.g., ambient, liquid, and vacuum), making it a valuable imaging method for biomedical research. The typical resolution of AFM is on the order of several nanometers for in-plane resolution (X-Y axis) and sub-nanometers for out-of-plane resolution (Z axis, Z-height).

To overcome the challenges of standard diagnostic procedure, in addition to optical imaging for early and reliable prediction of metastasis, the scientific community has come up recently with a state-of-the-art method that can extract a remarkably accurate prediction by utilizing variograms and fractal analytical methods on AFM of human colorectal cancer (CRC) histological images and texture algorithms, exploiting the tiny variations of cells' proteins biological signatures, shapes and tissue morphologies [5].

Then the natural question emerges: Could this method be improved upon, and could it be done to increase its availability to the public? With the advancements in computer science, one might turn to that field in search of answers [14].

Indeed, an early cancer diagnosis can benefit from the fast analysis of the data and the capacity to quantify the cell and tissue morphology for instantaneous real-time feedback. For early cancer diagnosis, a supervised deep-learning method for identifying and categorizing cancer tissue as metastatic and non-metastatic with high accuracy is significant for cancer staging.

## 2.2 Basic concepts of artificial Neural Networks

Owing to AI, machines can learn from experience, adapt to new inputs, and carry out activities that humans usually cannot complete. The majority of AI examples that are discussed nowadays mainly rely on DL, a branch of AI, and natural language processing. With these technologies, computers may be taught to process large amounts of data and identify patterns in the data to do particular tasks. Thus, DL has attracted much attention due to its notable performance in image recognition and classification tasks in various fields, including medical imaging, even though few intelligent systems are now self-adjusting [15],[9], [16].

Artificial neural networks are the main idea behind AI systems. The artificial network comprises multiple processing units, named nodes, and neurons (or perceptions) that are structured into layers. Neurons work together to simulate the activation state of a real neuron based on inputs that interact with it.

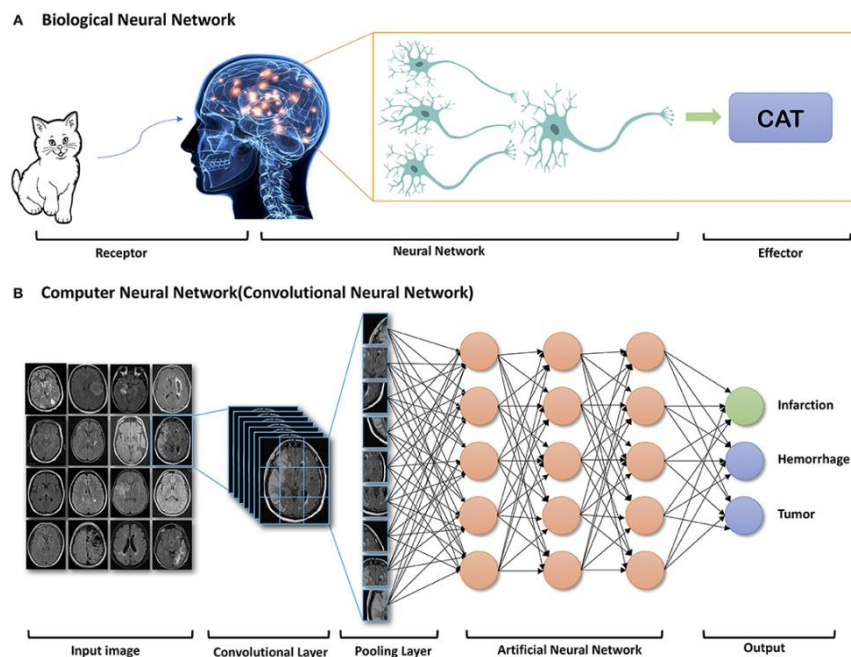


Figure 2: (a) Mechanisms of Biologic Neural Network and (b) Computer Neural Network. Reprinted from [17].

Specifically, neurons receive electrochemical signals from other neurons or signals generated from the different tissues (input data) Figure 2, [17],[6]. The signals are processed by each neuron and, depending on the interactions with the other neurons,



produce other signals (output data) when the input exceeds a certain threshold, allowing the transmission of the electrical signal to continue (synapse). The function is called the activation function.

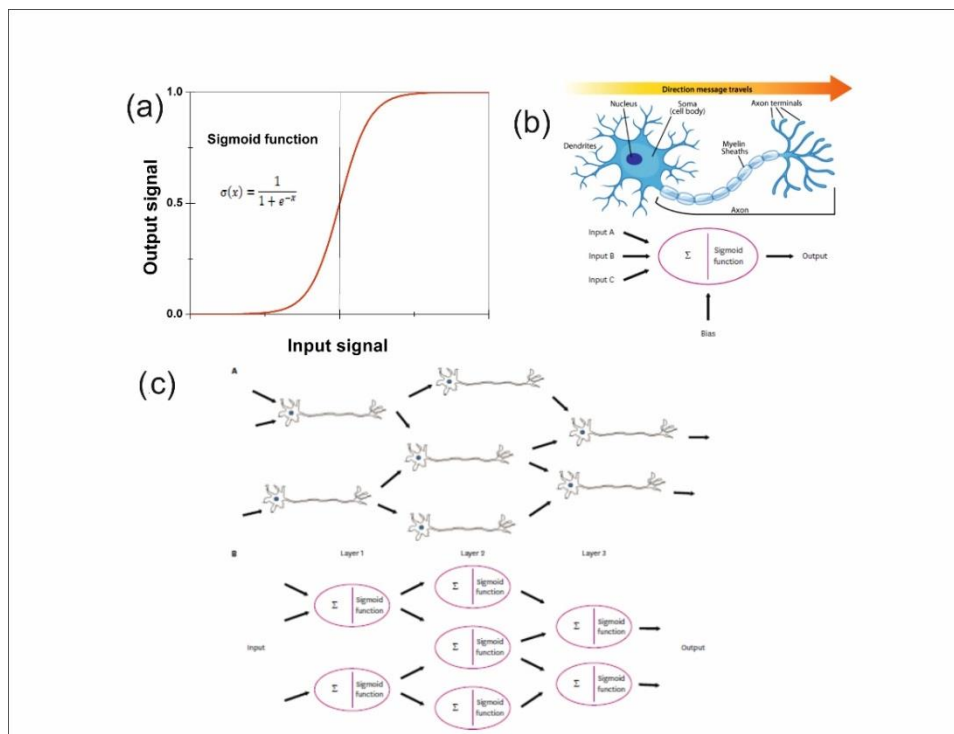
The following equation describes the output of a node (e.g., node 1 of *Figure 3*),

$$y_1(x, w_1, b_1) = f(b_1 + \sum_{i=1}^i w_{1,i} x_i) \quad \text{Eq. 1}$$

$f$  is the activation function and is a nonlinear function,  $x_i$  are the inputs,  $b$  is a constant (bias) value that permits the activation function to be shifted to the left or right, and  $w$  is a function associated with the weights or training parameters (e.g.,  $w_1 [w_{11}, w_{12} \dots w_{1n}]$ ). Changes in weights ( $w$ ) alter the steepness of the sigmoid curve [18].

Next, the activation function and training weights are briefly discussed. In the case of artificial neural networks, mainly *sigmoid activation functions* are used, which are very simple to calculate compared to other functions, *Figure 3a*.

$$\sigma_x = \frac{1}{1 + e^{-x}} \quad \text{Eq. 2}$$



*Figure 3* : (a) Sigmoid function; (b) Input and output signals from artificial neural and biological neurons; (c) (A) Biological neural network and (B) multi-layer perceptron in an artificial neural network. Reprinted partly from [19].

Similarly to biological neurons, artificial neural networks receive multiple inputs, then add them and process the sum with a sigmoid function (Figure 3b). Figure 3c displays models of an artificial neural network compared to biological neurons. Neurons are arranged in multiple layers. Each single neuron is connected to several neurons but without connections between the units of the same layer.

In the artificial neural network, the input neurons, forming the input layer (layer 1), receive the input data sum of the input values multiplied by the weights (known as training parameters). The weighted sum is introduced to the activation function (nonlinear function) and sent by synapses to layer 2; the hidden layer is also called the "black box" (hidden layers can be one or more). Finally, the hidden layer transfers the information to the output layer (layer 3).

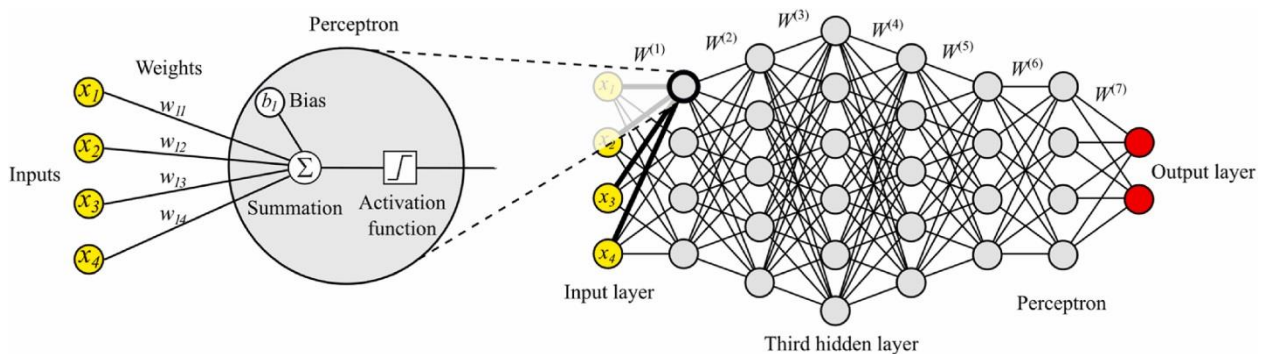


Figure 4: General artificial neural network model showing the way of generating the output of each neuron. Reprinted from [15].

Figure 4 shows a general image of an artificial neural network model and its main components. Specifically,  $x_1, \dots, x_4$  represents the information (input) the neuron receives from the external sensory system or other neurons.  $w_{11}, \dots, w_{14}$  are the vectors of synaptic weights that modify the received information, emulating the synapse between the biological neurons. These weights can attenuate or amplify the values they wish to propagate toward the neuron. Inside the magnified circle, the Parameter  $b_i$  is known as the bias (threshold) of a neuron.

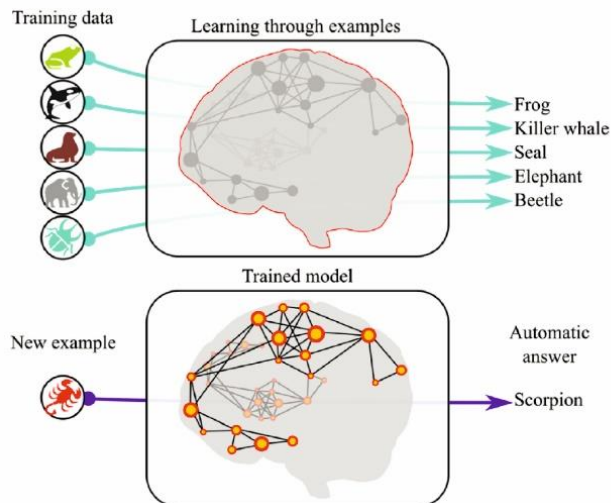
From Eq. 1, the output of the first layer is given by

$$y = f(W x) \quad \text{Eq. 3}$$

Since the layers are similar, the general form of Eq. 3 is

$$y^l = f^l(w^l y^{l-1}) \text{ Eq. 4}$$

Through training or learning processes, neurons in an artificial neural network alter the connections between them to increase the accuracy of the output results. Instead of modifying the sum of the input values or the sigmoid function, the connection strength between the nodes is adjusted by reformulating the weight applied to the connection signal strength. Low weights weaken the signal, and high weights enhance the signal. This process continues until the intended result is achieved based on the error between the predicted and the correct outputs, *Figure 5*.



*Figure 5: A model or network that can produce automatically accurate results in new examples after efficient training. Reprinted from [15].*

Calculating weights is complicated; thus, alternatives such as the gradient descent method are proposed to obtain the correct answer [19]. Backpropagation, a special case of the automatic differentiation general technique, is commonly used by the gradient descent optimization algorithm to alter the weight of neurons by calculating to minimize a loss function.

### **2.3 Convolutional Neural Networks**

The exponential growth of DL is primarily attributable to the developments in computer science, which enable us to derive information from images (computer vision).

As such a technique, CNN has garnered significant interest across various domains [15],[20]. CNN has become the mainstream algorithm in image classification, experiencing a rise in prominence since 2012. It has been applied to various visual recognition tasks with significant outcomes, including detection, localization, and semantic segmentation [21].

The CNN architecture comprises several layers resembling the brain's neurons. The brain processes visual information as an electrical signal from the retina using the optic nerve. Also, the visual cortex is formed by dense layers of cells. Using different filters by CNN to extract information from the input image resembles the brain's mechanism to extract various information, like edges and essential parts of the image.

Unlike traditional neural networks, which mainly rely on matrix multiplications, CNNs utilize convolution. More analytically, CNN consists of two function operators: the image and a small matrix of numbers, the kernel (filter). The image is transformed based on the values from kernel as the function takes part of the image and highlights patterns by multiplying each point of the image fragment with the kernel elements, and the results are summed together. Each kernel is intended to detect different features. The feature map values are calculated using the following formula:

$$G[m,n] = f * h [m,n] = \sum_j \sum_k h[j,k] f[ m - j, n - k] \quad \text{Eq. 5}$$

$f$  denotes the input image, and  $h$  is the kernel. The  $m$  and  $n$  are indexes of rows and columns of the result matrix.

The generated values are placed in the position corresponding to the image fragment. The process is repeated by moving the filter through the image for further processing and generating feature maps with highlighted features depending on the filter's structure.

Each map is subjected to the activation function according to Eq. 1.

In this case, the weights assigned to each filter serve as the network's training parameters; that is, the network learns which filters best highlight the high-level characteristics that converge to the intended objective [22], [23].

Figure 6 is the graphical description of the convolution process and onomatology [9],[24].

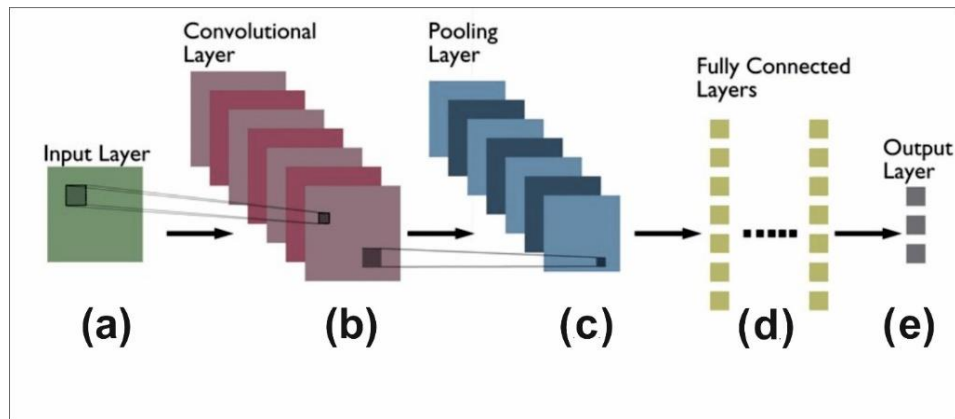


Figure 6: CNN Architecture. Reprinted from [24].

1. An input layer (image), (a) in Figure 6, generates one or more feature maps depending on the filter's number.
2. The convolutional layer, (b) in Figure 6, is the most crucial. It consists of kernels that allow the expression of the input image as N-dimensional metrics/feature maps, e.g., when the network has two kernels in the first layer, two feature maps will be created in the output, Figure 7. Feature maps act as intermediaries that capture and amplify essential patterns in the image. Also, a filter of any size can be applied to each layer. Another parameter of the convolution operation is the stride, which refers to the number of pixels by which the filter matrix moves across the input layer.

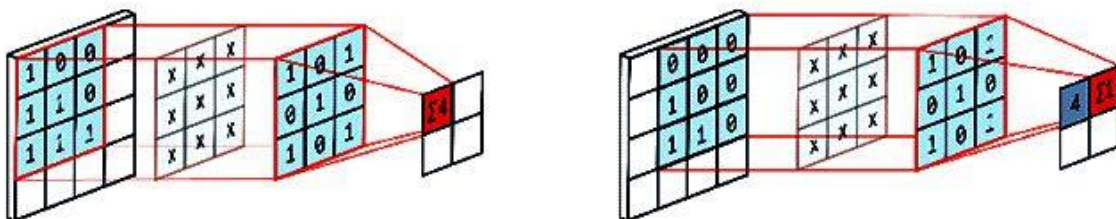
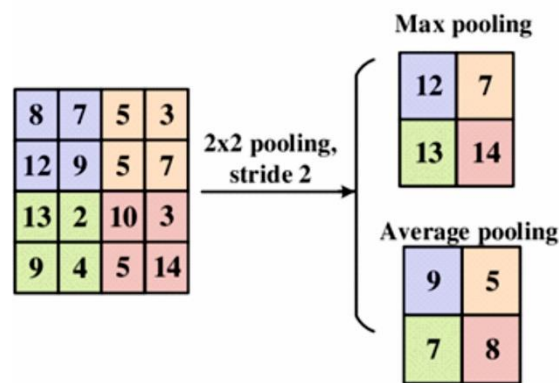


Figure 7: Convolutional Layer Operation. Reprinted from [25].

3. The pooling layers consolidate the features learned by CNNs, as shown in (c) in Figure 6; slight alterations in the input image cause tiny alterations in the feature maps of convolutional networks. The central objective of the pooling layer is the sub-sampling

of feature maps, meaning it is used for shrinking large feature maps into smaller ones using a particular pooling technique that gives some transitional invariance to the models and to ease the computational load. Like convolutional layers, pooling produces a single value for a window that moves across the picture. Sizes and stride can vary inside the window depending on whether it is the highest value (Max pooling) or the average (Average pooling), *Figure 8*. The standard type of pooling used in CNNs is Max Pooling due to its capacity to retain more pertinent data for tasks involving object detection and recognition.



*Figure 8: Pooling Layer Operation. Reprinted from [26].*

4. Activation function: By generating the relevant output, the activation function decides if a neuron will be fired, depending on the input. (more details were given previously in the text, e.g. Eq.2)
5. Fully connected layers, or dense layers, are the CNN classifiers, (d) in *Figure 6*. They come after the convolutional and pooling layers. These layers play a crucial role in combining the high-level structures learned by the convolutional layers (learnable parameters: weights and biases) and making predictions based on those features [27]. Its input is a vector created by flattening the feature map (arranging all the elements in a single row), while the output represents the output of the whole architecture *Figure 9*. Typically, at the end of the architecture, each neuron of the layer is linked to all the neurons of the previous layer.

Equation 6 describes the convolution process [15].  $M$  is the feature map, "\*" symbolizes the convolution between the  $i^{th}$  map  $X_i$  and the filter  $K_{il}$  at the map equivalent depth,  $b_1$  and  $f$  are the bias and the activation function, respectively.

$$A_1 = f(\sum_{i=1}^M X_i * K_{il} + b_i) \quad \text{Eq.6}$$

A model is considered overfit when its performance on the training set is significantly higher than on the testing set.

In particular, random noise and meaningless trends in the training data significantly impact decision-making.

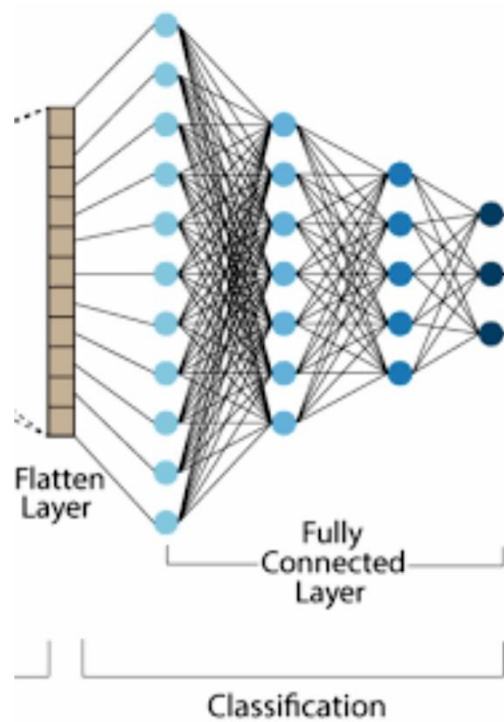


Figure 9: Fully Connected Layer. Reprinted from [28].

Performance on the training dataset is improved, but fresh data are not well-generalized as a result. There are numerous methods for lowering over-fitting. The most popular are [29]:

- *Increasing the sample data set size.* Data augmentation is a technique to expand the size of the sample data set by applying various transformations, like flipping, rotating, shape- or color-distorting, or changing the scale.
- *Using less complex models with fewer parameters.*

- *Using regularization methods:* Such a method is the dropout that turns off some of the "neurons" at random. As a result, during the optimization process, some "neurons" are disregarded in each training iteration, and their parameters are not updated.
- *Transfer learning* applies information from one problem to a related but distinct one.
- *Tuning hyperparameters of the model with cross-validation:* For a given set of hyperparameters, the model is trained on the training data, and thereafter, for every set of hyperparameters, the model presentation is evaluated on the validation set.
- *Early stopping of the training process:* Stopping the training process before it overfits is one method of preventing over-fitting. In that instance, greater generalization can be accomplished because the model does not retain the training data.

#### **2.4 DL tools for AFM-based applications**

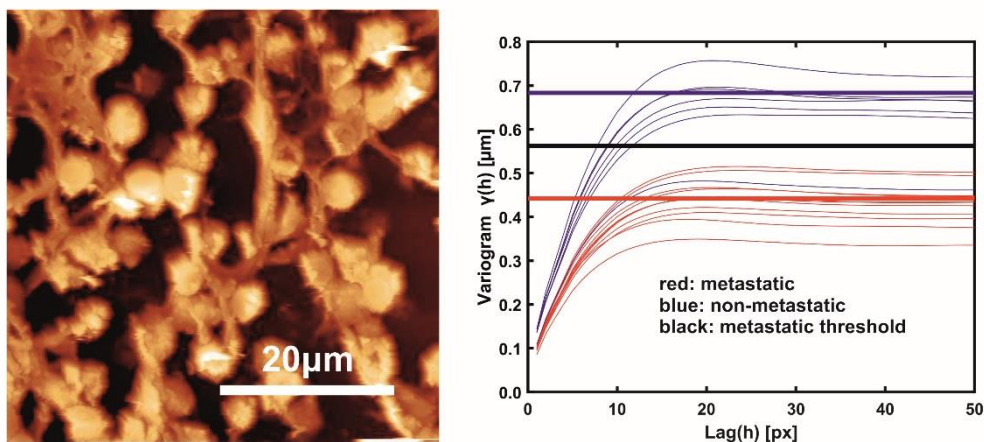
AFM provides atomic-scale resolution and real-time imaging of bio-surface properties compared to other imaging techniques (optical, confocal, and holographic microscopies). Likewise, to acquire cell or tissue surface topology or mechanical properties in almost physiological conditions, AFM can image samples' properties at the nanoscale in various environments such as gas, liquid, or vacuum.

Tissue micro-environmental factors, including stiffness (mechanical properties) and topography (the nuclei's shapes, morphology, and texture specificity), contribute to the targeting preferences of metastatic cancers because biological and mechanical/topographic parameters are associated with cancer cell proliferation, migration, and metastasis. Although AFM is a powerful tool that has been essential to the study of cancer, its application to the study of tumor pathology at the tissue scale is relatively new since most AFM research, especially in the area of cancer, is mainly concentrated on the nanomechanical characterization of cells falling short in capturing the intricate and diverse nature of tumors and their host organs. The studies that have utilized AFM to characterize the mechanical properties of various cancer tissues, as discussed in the review article by Najera et al., also highlight the potential clinical applications of this methodology [30].



Recently, the malignancy condition of a few questionable cells ( $50\ \mu\text{m} \times 50\ \mu\text{m}$  scan sizes) not detected by optical image analysis was classified by establishing metastatic indexes and thresholds at the nanoscale level from relatively large histological sections ( $5\ \text{mm} \times 5\ \text{mm}$ ), elucidating thus the early metastatic progression [5]. The early detection of tiny morphological differentiations of proteins with  $80\ \text{nm}$  resolution indicates the potential cell transition from epithelial cell phenotypes of low metastatic potential to those of high metastatic potential. The procedure uses variograms and theta statistics of colorectal cancer tissue images, *Figure 10*, grouping the well-defined threshold lines for metastatic and non-metastatic tissues. Above the black threshold line, tissues are non-metastatic and metastatic below the line.

Analyzing surface characteristics of AFM images based on statistical models and texture algorithms requires expert personnel with a solid physical and mathematical background.

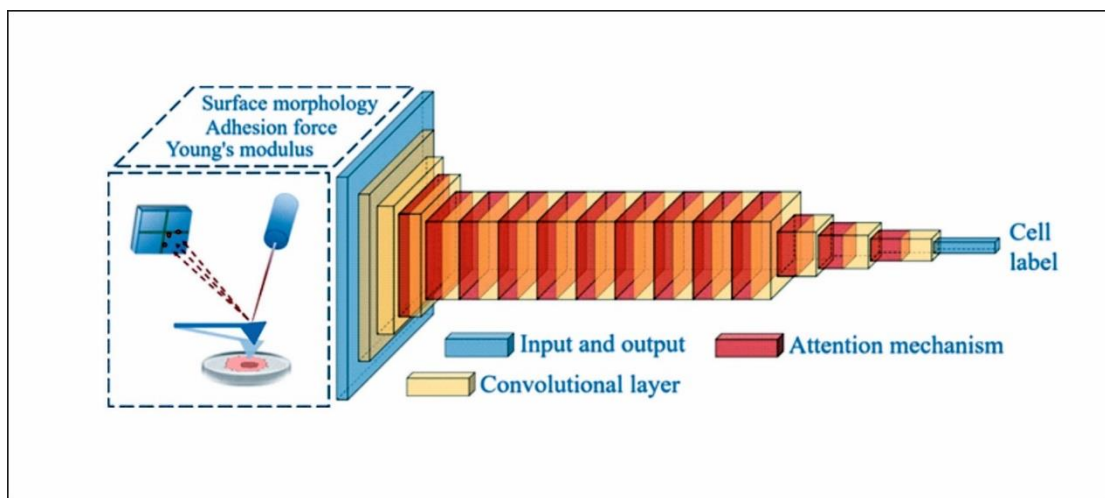


*Figure 10: Left: AFM image of human CRC histological section. Right: Variograms of metastatic and non-metastatic tissue images. Above the black threshold line, tissues are non-metastatic and metastatic below the line. Reprinted from [5].*

Thus, instead of a direct prognostic response, variograms make effective use of the AFM practically inapplicable in a hospital.

Integrating DL, particularly CNN, with AFM offers a transformative solution to these challenges. Unlike AFM image processing methods, which rely on user annotation, CNN can automatically decrypt or learn the most complex features in a collection of images, achieving efficient prediction and thus being more feasible for clinical translation.

Unlike traditional machine learning, DL technology can learn the differences in cell characteristics from the initial input to the final output without the need for intermediate manual intervention. Thus, it is not influenced by the combination of manually selected features, making it more versatile [2]. The schematic diagram of *Figure 11* shows an improved CNN network. The multi-scale convolutional fusion method extracts features based on different sensing domains at various scales. This approach enhances the input features at multiple scales and improves the network's adaptability to scale variations.



*Figure 11: Graphical abstract on cell recognition based on AFM and modified residual neural network. Reprinted from [2].*

The opportunities and limitations of combining DL with scanning probe microscopy (SPM) techniques, including AFM, have recently been reviewed by Azuri et al. [29]. The subgroup of DL algorithms, CNNs, is highlighted, and their potential for applications in SPM is discussed, including examples.

The multidisciplinary field embraces various research areas for nanoscale surface characterization by recording surface topography and nanomechanical properties, contributing thus to cancer cell detection and classification, cell size and shape recognition, protein identification, etc. [31].

Along the above lines, the mechanical stiffness of cells recorded with force-distance curves from 15 patients was used to distinguish cancer from healthy brain tissues. The classification of results was more precise than those based on the calculation of fitting parameters [32]. (a) Figure 12 shows the layout of the NN algorithm used to

classify different tissue types and detect meningotheial meningioma neoplastic cells within healthy tissue. The neural network was trained using FD curves measured on healthy and meningioma tissues. Representative FD curves from both groups are shown in (b) and (c). The Young's modulus map obtained from the analysis of FD curves is shown in (d) on a color scale expressed in different Pa values. The NN result is shown in (e) and (f).

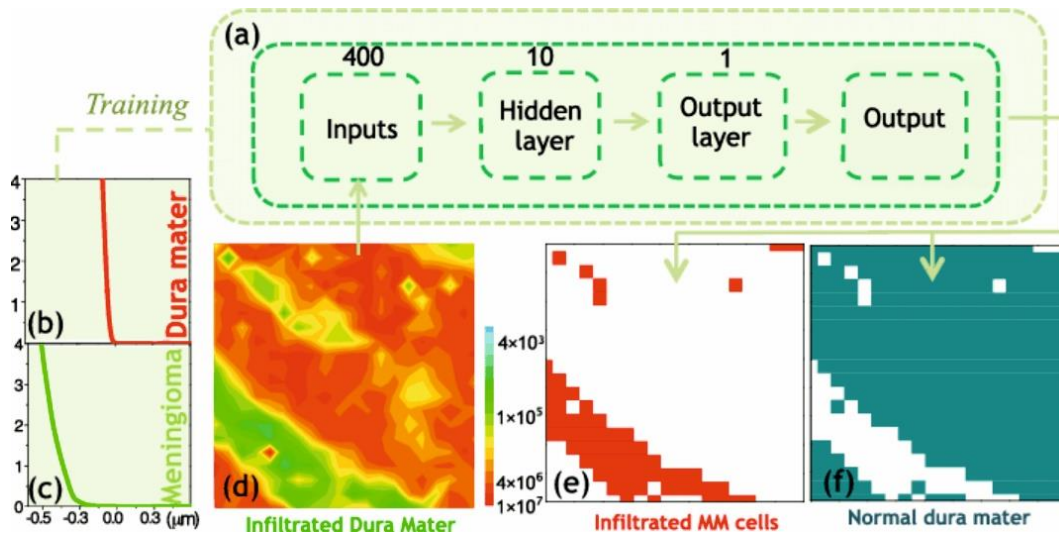


Figure 12: The neural network algorithm's architecture for classifying various tissue types and identifying MM malignant cells that have invaded healthy. Reprinted from [32].

ML analysis of data obtained by AFM in Ringing mode (base on the AFM identification of cells) was reported by Prasad et al. for distinguishing between two similar human colon epithelial cancer cell lines with varying degrees of neoplastic aggressiveness. The classification accuracy achieved was up to 94% base on single-cell images (area under the receiver operating characteristic curve of 0.99) [33].

The introduction of ML methods for the analysis of AFM adhesion maps allowed the differentiation between precancerous and cancerous cervical cells with improved precision, compared to the one based on fractal dimension analysis [34].

An analytical approach based on the physical characteristics (morphology, adhesion force, Young's modulus) of groups of cells and DL was used to identify different types of cells. AFM was used to image the cells, and an optimized CNN was employed to evaluate the generated images. The cell lines' recognition results show that the methodology can be efficiently applied for applications like cancer cell detection [2].

In another application, automated analysis involving pore detection (defects) of AFM images of biological membrane models is employed to understand the nature of membrane damage caused by pore-forming toxins. The specific object-detection algorithm used to determine the defect coordinates in AFM images was CNN [35].

Nano-biomechanical (force-distance curves) on living pathogenic microorganism cells of medical interest acquired by AFM force spectroscopy. DL techniques, specifically those based on the CNN architecture, were used to classify such data to predict cell adhesiveness, enhancing the understanding and identification of cellular properties and behaviors [36].

The DL framework was also implemented to automatically select samples based on the cell shape for AFM probe navigation during AFM biomechanical mapping [37]. The use of DL-assisted image recognition in AFM facilitates fluorescence-independent identification of individual co-cultured cells and enables efficient AFM force measurements on these identified cells in a liquid environment. By employing a deep learning-based image recognition model, the AFM probe can be precisely moved to the cells; different parts (central areas and peripheral areas) to perform mechanical measurements under the guidance of deep learning- based optical image automatic recognition. Thus, the viability and the type of individual cells in co-culture environments we identified, and fluorescent labeling results confirmed these identifications [16].

Besides bio-related applications, a DL model trained with a theoretically generated dataset was used to directly predict the molecular structure of AFM and images for molecular identification [38],[39]. Also, an approach based on CNN was used for AFM image reconstruction to eliminate image distortions or tip-related artifacts (as it requires a slow- scanning speed to safeguard its image quality), manually selecting image processing steps, and parameter tuning [40].

Overall, the convergence of AFM and DL represents a multidisciplinary approach that leverages both technologies' strengths. CNNs can autonomously learn and decode complex features in AFM images, facilitating efficient prediction and enhancing clinical applicability. This combination has shown potential in various applications, from detecting cancer cells and classifying tissue types to understanding membrane damage and predicting cell adhesiveness. Moreover, DL techniques have improved AFM image

processing by eliminating distortions and artifacts, thereby increasing the accuracy of molecular identification, *Table 1*.

This thesis investigates the surface modification at the nanoscale of surgically removed human colorectal tumors to assess the AFM data analysis approach for the automated classification of cancer tissues. The classification results based on AFM images were compared with those of clinical methods, and those predicted using variograms on the same images. The cell and tissue recognition method combining AFM and CNN is expected to bring new progress for bio-applications, including cancer studies.

*Table 1: Machine learning techniques and their bio-applications with AFM.*

Microscopy tool	Nanoscale imaging of	Bio-sample	Applications	ML techniques	Publication
AFM	morphology adhesion mechanics	cell	cancer diagnosis	CNN	J. Wang [2]
AFM/SPM	morphology	red blood cell	healthy vs. damaged red blood cells	CNN	I. Azuri, Review article [29]
AFM/SPM	mechanics	tissue  cell	cancer diagnosis  size and shape  Protein identification	ANN  SVM (support vector machine)  CNN	M. Rahman Laskar, Review article[31]
AFM	mechanics	tissue	cancer diagnosis method	NN	E. Minelli, [32]
AFM	mechanics	cell	cancer diagnosis	ML	S. Prasad, [33]
AFM	mechanics (adhesion)	cell	cancer diagnosis	ML	M.Petrov [34]
AFM	morphology	membrane	membrane damage diagnosis	CNN	T. Raila,[35]
AFM	mechanics	yeast cells	Predict the adhesiveness	CNN	A. Martinez-Rivas [36]
AFM	morphology phase-contrast	cell	automated selection of cell locations	CNN	J. Rade, [37]
AFM	mechanics	cell	label-free recognition	DL	X. Yang [16]

### 3. Methodology

The methodology for this thesis follows a well-structured pipeline, utilizing various advanced tools to ensure an efficient approach. Detailed explanations of each step and the rationale behind the decision-making process are presented in the subsequent sub-chapters, which include the tools utilized, the specs of the deep learning model, and methods used in pre-processing the data.

The main steps of the process involving AFM images of metastatic and non-metastatic tissue samples for developing DL algorithm, as shown in *Figure 13*, are:

1. *Acquisition of AFM Images* of both metastatic and non-metastatic colorectal cancer tissue samples. (This step was carried out previously and thus is not part of this thesis.)
2. *Reduction of Dimensionality* of the AFM images to simplify the data while retaining significant features.
3. *Data Splitting* into Training (86%) and Testing (14%) groups: This split ensures that the model is trained and evaluated on separate data to avoid overfitting.
4. *Developing the DL Algorithm*. The training subgroup was used to develop a deep learning model involving CNNs for image analysis, adjusting parameters to improve performance.
5. *Statistical Analysis Using Testing Subgroup*. The developed DL model was tested on the testing subgroup to evaluate its performance. Key metrics to assess accuracy are the confusion matrix, the area under the ROC curve, etc.

The development process leverages Python for scripting and automation, with Google Colab providing a collaborative and flexible coding environment. The Keras Deep Learning API and TensorFlow and Keras Applications are utilized to build and train deep learning models.

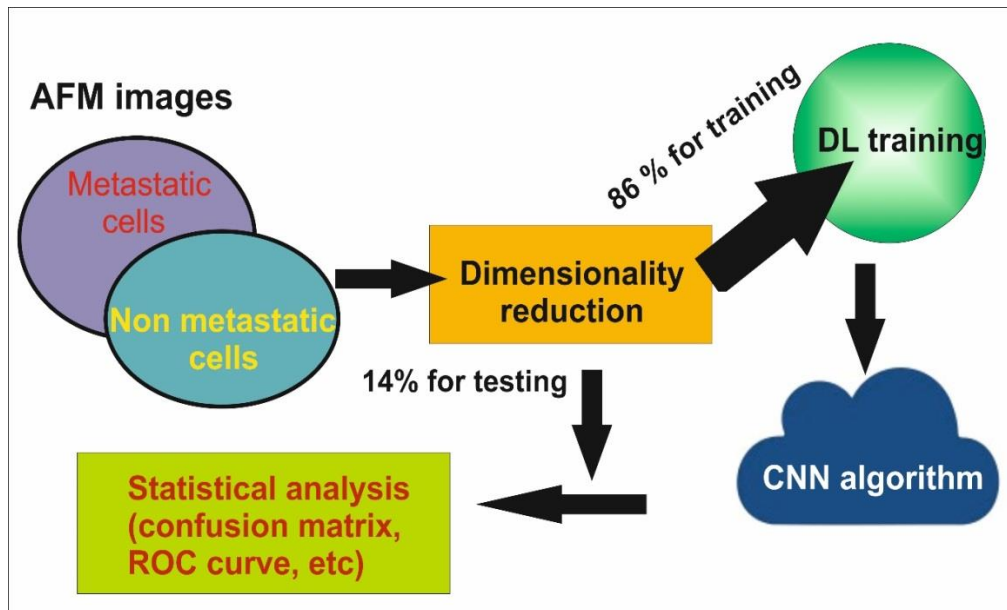


Figure 13: Graphics of Deep Learning (DL) analysis. Reduction of dimensionality of the AFM metastatic and non-metastatic CRC images; splitting the data into the training and testing groups; developing a DL algorithm using just the training subgroup; using the testing subgroup to perform the statistical analysis of the developed algorithm while employing cross-validation.

### 3.1 Tools

The tools for implementing the project development strategy are briefly presented, including AFM, Python, Google Colab, Keras Deep Learning API, TensorFlow, and Keras Applications.

#### 3.1.1 AFM

AFM is an ideal tool for recording the surface topology at the nanoscale in non-destructivity. Below is a concise overview of the operating principles of AFM topography aimed at readers who are less familiar with the topic. Binnig et al. were the first to invent the tunnel electron microscope in 1981 (scanning tunneling microscope, STM) [41]. Atomic resolution imaging is possible thanks to this ground-breaking technology, and it overcomes the limitations of visible light, which is the primary constraint of an optical microscope. Early experiments with STM showed that when the tip and sample distance is only a few Angstroms, a set of forces emerges, allowing an atomic scale morphological imaging of non-conductive materials, *Figure 14*, [42]. Since its invention, AFM has had tremendous success across various scientific areas, including biological sciences.

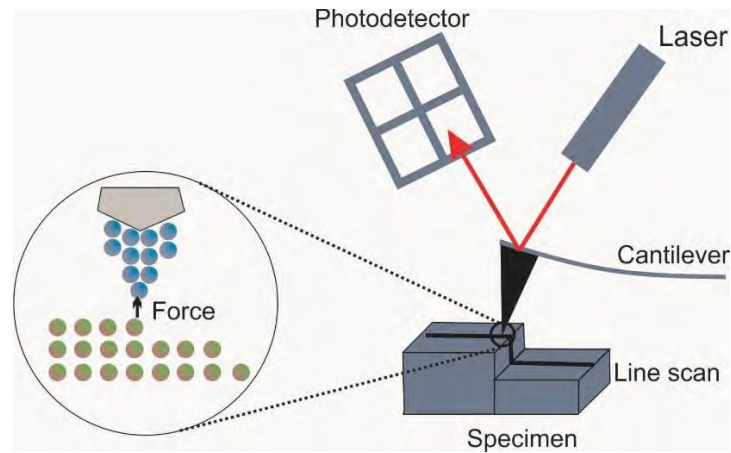


Figure 14: Schematic layout of the AFM operation principle. Reprinted from [43]. .

The AFM comprises a 100  $\mu\text{m}$  cantilever equipped with the tip, the sensor, and the power actuator. The cantilever/tip assembly, also known as the probe, interacts with the material and forms the basis of the AFM concept. While scanning, the AFM probe uses a raster scanning motion to interact with the substrate. During the AFM tip's motions (up/down and side-to-side) on the surface, the laser beam reflected off the cantilever. There are multiple modes of AFM (contact, non-contact, force-distance, electrical, conductive, etc.)

Specifically, a translational x-y stage scans the plane of the sample, and a piezo-crystal controls the separating distance at the atomic scale and, thus, the tip-sample force. The interaction force between the tip and the surface under investigation is translated into cantilever deflection, allowing two and three-dimensional topographic imaging. A position-sensitive photo-detector detects the probe's lateral and vertical motion and tracks this reflected laser beam. The detectors' deflection sensitivity has to be calibrated in terms of the number of nanometers of motion that a unit of voltage measured on the detector corresponds to.

A reflective metal film covers the backside of the Si or  $\text{Si}_3\text{N}_4$  cantilever with aluminum or platinum/iridium alloys, so a laser photodiode records the optical detection of its motion. The AFM tip, located in the cantilever's free end, usually has a height of a few  $\mu\text{m}$  and a radius of a few nm, while its shape depends on the application. The

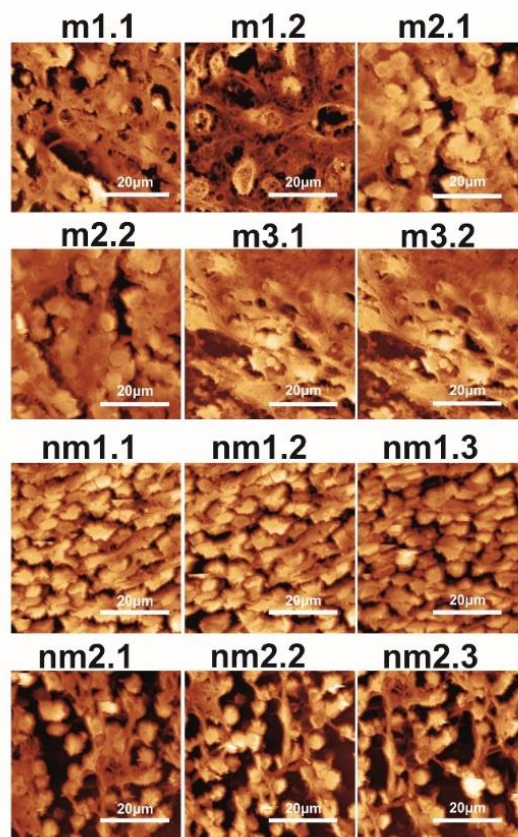


wavelength resolution of an optical microscope is influenced by the wavelength of the illuminating light (mainly from 250 to 420 nm).

The AFM resolution is primarily determined by the curvature radius of the AFM tip, which varies between 2 and 5 nm, resulting in a significantly better resolution. Usual pin shapes are wedge-shaped, pyramidal, spherical, hemispherical, and conical.

The AFM can be used in soft materials research, including polymers and biomaterials. The cantilever's spring constant is critical to image soft samples in contact mode.

The histological tissues used in the present study were imaged with Innova AFM (Bruker/Veeco, Inc., Santa Barbara, CA, USA) operating in tapping mode; examples are shown in *Figure 15*, [5]. In tapping mode, the cantilever oscillates near its resonant frequency above the sample's surface, and the tip only briefly touches/knocks the surface periodically.



*Figure 15: Examples of typical  $20 \times 20 \mu\text{m}^2$  AFM images of (m) metastatic and (nm) non-metastatic cancer tissue used in this study. Reprinted from [5].*

### **3.1.2 Python**

In programming terms, this project materialized in the Python programming language, utilizing the Google Colab interface.

Python is a powerful programming language created by Guido van Rossum, released in 1991, and quickly drew the interest of data scientists, web and software developers for various applications and scientific research, including system scripting, mathematics, software development, and server-side web development. To do these, Python makes handling enormous volumes of data and performing intricate mathematical computations possible. It also can be used with other technologies to develop workflows, communicate with database systems, and read and edit files; thus, it is very suitable for data analysis and machine learning.

Since Python is an interpreted language, source code is handled in real-time instead of compiling ahead of time; thus, it is a valuable tool for producing production-ready software and quick prototyping. Last but not least, it is open source, easy to learn, compatible with a wide range of graphical user interfaces (GUIs), works with other codes such as Java, C, or C++, and is used on various operating systems in a portable way (e.g., Windows and Linux) [44].

### **3.1.3 Google Colab**

The Colaboratory platform (Google Colab) is a free cloud-based tool allowing users to write and run Python code together in a Jupyter Notebook environment (a web-based interactive development environment for notebooks, code, and data) [45]. Google Colab is a well-accepted option for data scientists, machine learning specialists, and academics. It simplifies learning (ML) and data science activities by giving users access to free graphics processing unit (GPU) resources in a virtual environment called Google Colab Python. Besides being pre-installed with many popular Python libraries (for machine learning, data analysis, and visualization) and integrated with Google Drive, other advantages are that no setup is required, and it supports collaborative editing, easy real-time viewing, and editing sharing.

Since the project was implemented using Python, a model from those available via the Keras Deep Learning API (Application Programming Interface) was deemed appropriate.

### 3.1.4 Keras Deep Learning API

The Keras is a deep learning API framework developed in Python. It covers each step of the deep learning process, from data processing to hyperparameter tuning to deployment [46],[47]. The hyperparameters are tuned external parameters of the model (e.g., the number of neurons/kernels or kernel size). Other parameters of the models, such as weights and biases, are internal and learned automatically from the data. Its development is centered on implementing trials as rapidly as possible.

The main characteristics of Keras' open-source library, allowing developers to experiment quickly with neural networks, are the following [48] :

- Simplicity by offering simple, consistent interfaces.
- Offers flexibility by minimizing the number of actions needed for routine use cases. It also assists in providing clear error messages and writing concise, readable code.
- Powerfulness as it covers every step of the deep learning workflow based on the methodology of progressive resolution of complex issues.



Figure 16: Keras main layers. Reprinted from [49].

Layers are the fundamental structure blocks of neural networks in Keras, *Figure 16*, [50] :

- *0.1 Core Layers:* Input object, InputSpec object, Dense layer etc. The Dense layer is among the most critical core layers. This layer contains all the neurons. It is used to generate output in the preferred form.
- *0.2 Convolutional layers.* These are the primary building blocks used in CNNs. This layer creates a convolution kernel. It is convolved over a single input to produce a tensor of outputs. It detects patterns such as edges, textures, and shapes in the input data.
- *0.3 Pooling layers:* These layers are used to reduce the input's spatial dimensions, which helps reducing the amount of parameters and computations in the network. It is an additional layer for the convolutional layer to implement pooling operations. It is added to a CNN between the layers and uses the MaxPoolind1D () method, a valuable method for max-pooling operations on temporal data.
- *0.4 Recurrent layers:* These layers are present for abstract batch class. Handle sequential data, making them suitable for tasks like time-series analysis and natural language processing.

Other Keras layers are the Embedding, Merge, Dropout, Noise, Normalization, and Locally Connected.

### 3.1.5 TensorFlow

Keras can run on top of popular DL frameworks (e.g., JAX, TensorFlow, or PyTorch), *Figure 17*. This strategy offers:

- Optimal performance for the models.
- Increased the amount of ecosystem surface the models may use.
- Utilized pipelines for data from any source and expanded the accessibility of open-source model releases.

The TensorFlow ecosystem was used in the current study for efficient computation towards building and training deep learning models and using the simplicity and flexibility of the Keras API.

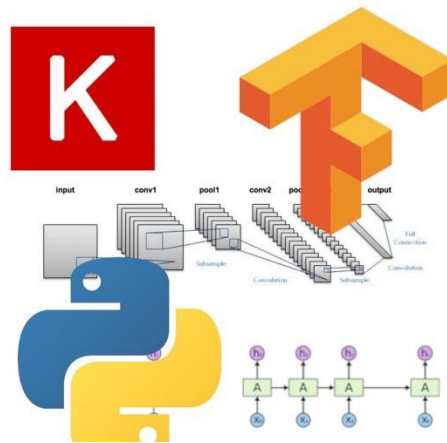


Figure 17: TensorFlow provides libraries for Python. It makes it easier for data collection, model training, prediction serving, and future result refinement. Reprinted from [51].

### 3.1.6 Keras Applications

Keras Applications are DL models for prediction, feature extraction, and fine-tuning and are made available alongside pre-trained weights.

One such architecture is the Inception, which has demonstrated exceptionally high performance at comparatively little computational cost.

InceptionV3 belongs to the Inception family and is among several Keras models for image classification, optionally loaded with weights pre-trained on ImageNet [52]. The InceptionV3 was chosen as a CNN sequential model [53], along with other reasons to be detailed further below. Its architecture is progressively built step-by-step, as displayed below in Figure 18.

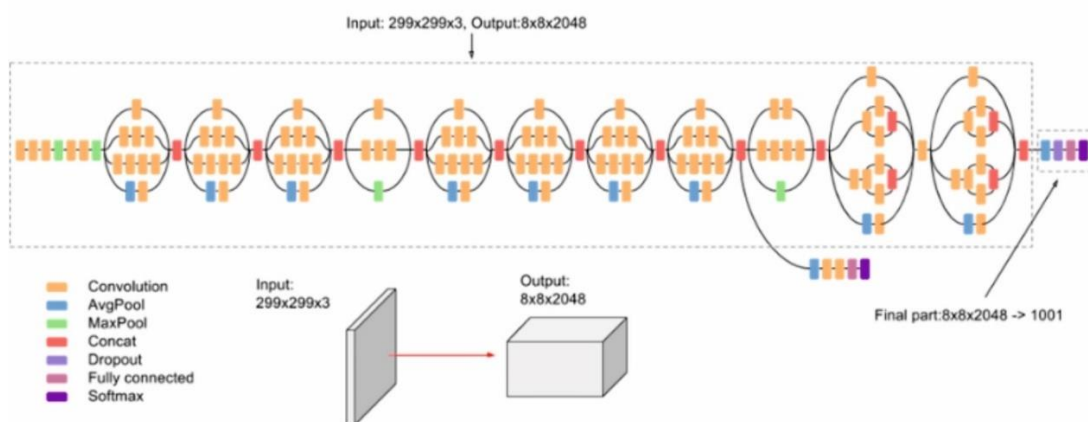


Figure 18: InceptionV3 Architecture. Reprinted from [51].

The main concepts of InceptionV3 combined into the final architecture are [54] :

- *Factorized Convolutions*: This helps reduce computational efficiency by reducing the number of parameters elaborated in a network. It also monitors the effectiveness of the network.
- *Smaller convolutions*: replacing bigger convolutions with smaller convolutions leads to faster training.
- *Asymmetric convolutions*: e.g., replace the 3×3 convolutions with a 1×3 convolution followed by a 3×1 convolution. That is the same as sliding a two-layer network with the equivalent receptive field as in a 3×3 convolution.
- *Grid size reduction*: Grid size reduction is typically done by pooling operations. In the Inception V3 model, the activation dimension of the network filters is expanded to reduce the grid size efficiently.

### 3.2 Pipeline

Automating the learning task's workflow is the learning pipeline (placement of the sub-parts of techniques). A pipeline supports engineers and data scientists in creating precise, scalable solutions for a range of use cases while also managing the complexity of the entire process. One way to accomplish this is to allow a series of data to be modified and associated within a model so that the output may be examined. Building, training, testing, and deploying deep learning models can be done more quickly, consistently, and automatically using a deep learning pipeline, a connected set of data processing and modeling operations. Raw data input, features, outputs, model parameters, machine learning models, and predictions are standard pipeline components.

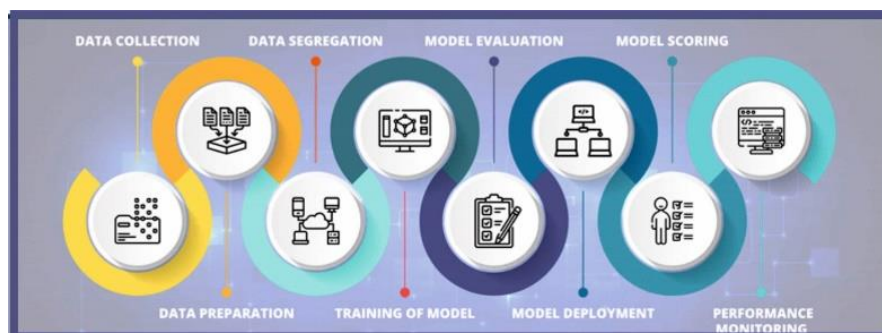
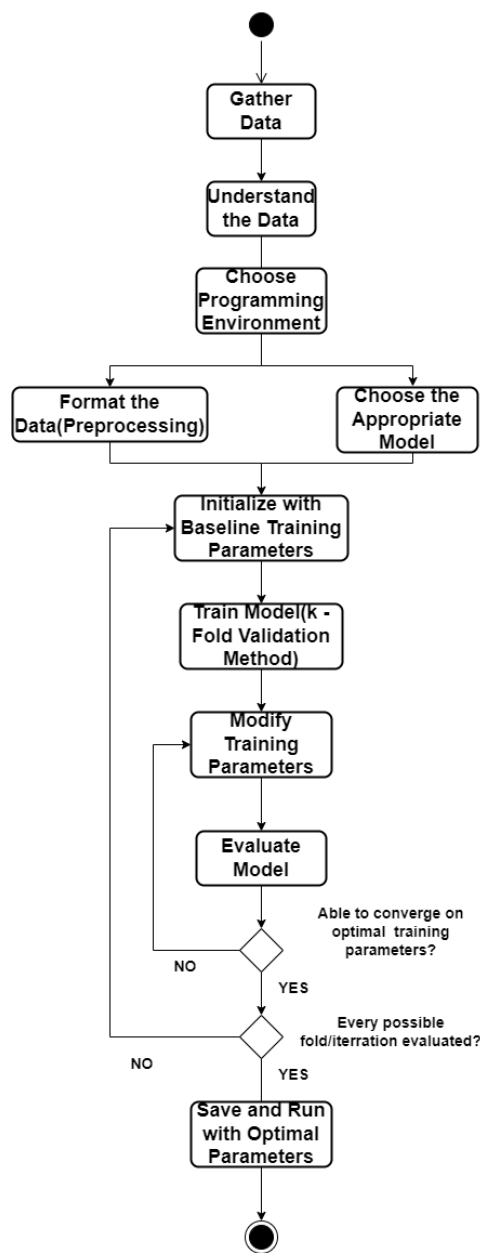


Figure 19: A typical pipeline workflow (modified reprint from [55]).

The Pipeline comprises several consecutive steps that carry out various tasks modularly, from data collection and pre-processing to model training, evaluation deployment, and scoring. Each pipeline step is designed as a distinct module, and the result of linking all of these modules is the finished product, *Figure 19*.

The workflow that was followed on the materialization of this master thesis is displayed below in *Figure 20*, in the form of an activity diagram, which includes data gathering, understanding, preprocessing, model selection, training, validation, hyperparameter tuning, and finalizing the model.



*Figure 20: Methodology Activity Diagram.*

## 4. Analysis of the Workflow: limitations and solutions

### 4.1 Dataset optimization

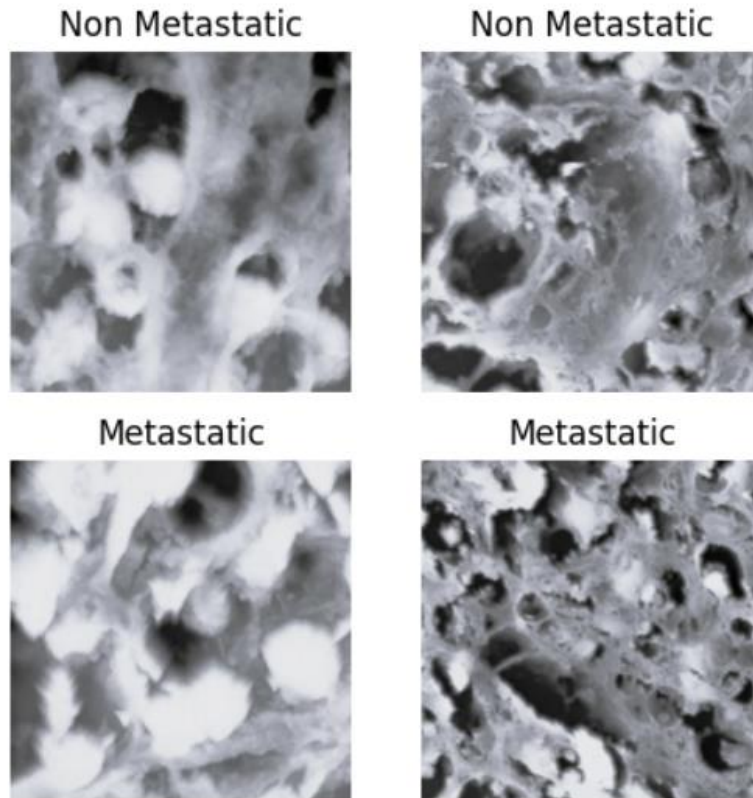
The first step of the process was to study the dataset to understand the data. The complete dataset consists of 45 \*.jpg image files, classified as metastatic or non-metastatic by their individual legends, *Figure 21*. Said images were captured by an AFM Microscope, depicting colorectal cancer cells' structures at the nanoscale. The dataset was determined to be balanced since it comprises 46.67% non-metastatic and 53.33% metastatic images.

Some technical issues became apparent when studying the dataset further and entering the pre-processing phase. Firstly, the images come with a border depicting the microscale of each sample and a heatmap scale depicting the intensity of the individual pixels. Secondly, the images' resolution is inconsistent across the board, meaning there is a high chance that a rescaling method needs to be applied. Lastly, a significant portion of the dataset is in grayscale.

In order to combat the issues stemming from said observations the samples were cropped so that only the image generated by the microscope is shown, followed by a uniform resize at 299x299 resolution (details will be provided in later stages as to why that specific aspect ratio) and, the whole dataset was then grayscaled so that a consistent input to the model was provided, even though gray scaling is a form of data reduction.

Wrapping up the pre-processing stage, the percentage of the images that are going to be used for training and validation and that for testing and evaluation of the model needed to be determined. Firstly though, the choosing of a fitting model must occur beforehand.





*Figure 21: Preprocessed Data Sample.*

#### **4.2 Model selection**

The fact that the dataset is classified as small, consisting only of 45 samples, was the primary problem that needed to be solved. The best practice for training a deep learning model is to provide a large amount of data at the training stage so that the model can optimally adjust the weights of each layer to accommodate every possible input image and provide the most accurate prediction possible. As is our case, having a small dataset, certain strategies were employed to combat the lack of training data for the accuracy to improve. This observation, mentioned earlier, played a pivotal role in the selection of the model.

Out of the available models in Keras, InceptionV3's accuracy metrics, being the result of training with the ImageNet dataset, and the fact that it comes with pre-trained weights as a result of training by said dataset were the key reasons as to why this model was the one to work with, alongside with it being a CNN sequential model. While studying the documentation of the InceptionV3, it is stated that the input images must

strictly be at a 299x299 aspect ratio. Circling back to the previous point of choosing a uniform aspect ratio for our samples, this condition reinforces the choice of model. That is because, during the review of the data, it was discovered that the aspect ratio of the images ranges from 255 255 all the way to 512x512, leading to the conclusion that a middle-of-the-road option is optimal for uniformity among the data. Again, note that doing this results in data reduction, as was the case with grayscaling, though necessary to achieve uniformity, uniformity being the desirable result for the dataset.

### 4.3 Training method

Besides choosing an appropriate model to accommodate the lack of data, training methods that nullify this problem need to be employed. The method in question was the K-fold cross-validation technique, Figure 22.

The process goes as follows: The whole dataset is partitioned into k parts of equal size. Each such partition is called a fold, hence the name of the technique. A random fold is applied for validation, and the residual K-1 folds are used for training. In order to make use of the whole dataset, the aforementioned partitioning is repeated K times (iterations) until each fold is used once. Having done all that, the pre-processing stage and the model selection concluded, making way for the training process.

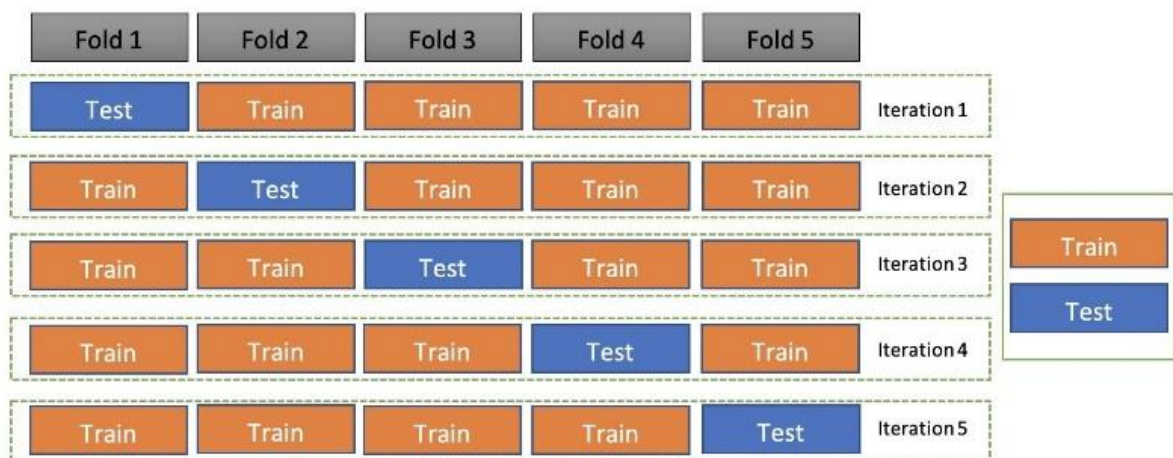


Figure 22: K-Fold validation. Reprinted from [54].

The rough standard for splitting a dataset into training, validation, and test is 60-80%, 10-20%, and 10-20%, respectively. From the 45-image total, 80% were used to represent the test set. That is 36 images that, in turn, need to be partitioned into training and validation data, leaving nine images for evaluation. For each different selection of K, the partition into training and validation data needs to be done with respect to the percentages mentioned above.

The initialization of the training step took place with binary cross-entropy as the dissimilarity metric. It measures how two probability distributions differ from one another: the true distribution and the predicted distribution (which is the model output, typically a probability score between 0 and 1 demonstrating the probability that the sample pertains to the positive class). In binary classification, where there are only two classes (commonly denoted as 0 and 1 or negative and positive), the binary cross-entropy loss function is calculated as follows:

$$\text{Binary Cross Entropy} = -\frac{1}{N} \sum_{i=1}^N (y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)) \quad \text{Eq. 7}$$

where:

- $N$  is the number of samples in the dataset.
- $y_i$  is the true label of the  $i$ th sample. It is either 0 or 1.
- $p_i$  is the predicted probability that the  $i$ th sample belongs to class 1 (the positive class), usually output by the model's final layer.

The term  $y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)$  ensures that the loss is minimized when the predicted probability matches the true label. If the true label is 1 (positive class), the loss is  $\log(p_i)$ , and if the true label is 0 (negative class), the loss is  $\log(1 - p_i)$ . The negative sign before the summation indicates that we are minimizing this quantity.

The training parameters that are interactable at the training stage are the number of epochs, the number of training steps, and the number of validation steps.

- The number of epochs dictates the number of times the model parses the entirety of the training dataset.

- The number of training steps in model training refers to the number of iterations or updates made to the model's parameters (weights and biases) during the training process. Each training step involves processing a batch of training examples through the model, calculating the loss (error) among the predicted output and the actual target, and updating the model's parameters to minimize this loss.
- The number of validation steps in model training refers to the number of batches of validation data used during each evaluation epoch. During training, after processing a certain number of training steps (where each step involves processing a batch of training data), the model's performance is evaluated on a separate validation dataset. This evaluation is performed in batches, similar to training, to utilize computational resources efficiently.
- After completing the training step, the evaluation step took place.

#### **4.4 Evaluation method**

As stated in section 4.3, nine images were kept for use as the test set for evaluation purposes. Out of those nine, three will be used to simulate an actual use case and comprise the prediction dataset, a subsection of the test set to be detailed further in this section.

In model training, the test set refers to a separate portion of the dataset that is held out and not used during training or validation. It serves as an independent dataset to evaluate the final performance of the trained model after training and validation are complete.

The test set's objective is to assess the model's effectiveness fairly with unclassified data. By evaluating the model on data it has never seen before, an assessment can be made on how well it generalizes to new, uncategorized examples. This is essential for understanding how the model will perform in real-world scenarios, where it will encounter data that is not part of the training or validation sets.

The specific evaluation metrics used were accuracy, evaluation loss, and the occurring confusion matrix and ROC curve:

I. In *model evaluation*, accuracy is a common metric used to estimate the overall performance of a classification model. It represents the proportion of correctly classified examples out of the total number of examples in the dataset.

II. The *evaluation loss* provides insight into how well the model is performing on the evaluation dataset. The chosen loss function is binary cross-entropy.

III. The *confusion matrix* is a table used in model evaluation to visualize the performance of a classification model. It presents a summary of the predicted classifications versus the actual classifications on a dataset, *Table 2*.

*Table 2: Confusion Matrix Template*

		Predicted class	
Actual class		Positive	Negative
Positive	True	Positive	False
	False	Positive	Negative
Negative	True	False	True
	False	Positive	Negative

- **True Positive (TP):** Instances where the model correctly predicts the positive class.
- **False Negative (FN):** Instances where the model incorrectly predicts the negative class (misses the positive class).
- **False Positive (FP):** Instances where the model incorrectly predicts the positive class (mistakenly identifies as positive).
- **True Negative (TN):** Instances where the model correctly predicts the negative class.

From the confusion matrix, various performance metrics can be derived to evaluate the model's classification performance, including:

1. **Accuracy:** The proportion of correctly classified instances out of the total number of instances. It is calculated as  $\frac{TP+TN}{TP+TN+FP+FN}$

2. **Precision:** The proportion of true positive predictions out of all positive predictions made by the model. It is calculated as  $\frac{TP}{TP+FP}$ . Precision focuses on the correctness of positive predictions.

3. **Recall (Sensitivity or True Positive Rate):** The proportion of true positive predictions out of all actual positive instances. It is calculated as  $\frac{TP}{TP+FN}$ . Recall focuses on capturing all positive instances.

4. **Specificity (True Negative Rate):** The proportion of true negative predictions out of all actual negative instances. It is calculated as  $\frac{TN}{TN+FP}$ . Specificity focuses on correctly identifying negative instances.

IV. The *ROC curve*, or Receiver Operating Characteristic curve, is a graphical representation illustrating how the diagnostic performance of a binary classifier system changes with different discrimination thresholds. It pictures the True Positive Rate (sensitivity) against the False Positive Rate (1 - specificity) for different threshold values. Each point on the ROC curve denotes a sensitivity/specificity pair consistent to a particular decision threshold. The area under the ROC curve (AUC) is also commonly used as a summary statistic to evaluate the performance of a classifier. An AUC closer to 1 indicates better classifier performance, while an AUC closer to 0.5 presents poor performance (similar to a random classifier).

The evaluation results were logged, and afterward, the training parameters were altered; the results were logged anew and compared to previous training cycles. This process repeats until convergence on the optimal training parameters that yield the best accuracy while avoiding overtraining or suboptimal resource usage. Having the optimal training parameters, in relation to the dataset, figured out leaves us with the demonstration part of the process. It was stated previously that a portion of the dataset was saved for extracting predictions. The prediction dataset is no different from the test dataset; it is not used only for the evaluation process but rather to display more tangible metrics for the user and simulate an actual application operated by a medical expert, for example.

Upon calling the model to generate a prediction on that dataset, the output entails the input image, with its classification as metastatic or non-metastatic, along with the model's confidence in said classification, which is transformed into a percentage value.

## 5. Results

This section presents the most notable results of the training and evaluation procedures regarding metrics. The most notable results are those that, when changing a training parameter, lead to a noticeable change in the model's prediction capability. In order to monitor the effects of altering training parameters to the prediction accuracy, a single training parameter is tampered with at a time until all cases are covered.

*Table 3* contains all the necessary information regarding the experimentation with the parameters. The model is instantiated for the first time with the following training parameters:

- K=4
- Training samples per iteration=29
- Validation samples per iteration=7
- Number of Epochs=2
- Number of Training steps=3
- Number of Validation steps=1

and yields the following results:

- Loss (training)=0.3364
- Accuracy=0.9259
- Validation Accuracy=1
- Loss (Evaluation)=0.4447
- Accuracy (Evaluation)=0.6667
- Sensitivity= 50%
- Specificity = 100%
- ROC AUC = 0.75

as can be observed in the first row of *Table 3*. This table row will be referred to as the base case.

As expected, due to the low number of Epochs during training, the results are underwhelming and are in no way suitable for extracting consistent predictions.



Table 3: Results under different parameter settings.

Folds/iterations	Training samples per iteration	Validation samples per iteration	Epochs	Training Steps	Validation Steps	Loss	Accuracy	Validation Loss	Validation Accuracy	Loss (Evaluation)	Accuracy (Evaluation)	Sensitivity (Metastasis)	Specificity (Metastasis)	ROC AUC
4	29	7	2	3	1	0.3364	0.9259	0.1166	1	0.4447	0.6667	50%	100%	0.75
4	29	7	10	3	1	0.0681	1	0.0195	1	0.3431	1	50%	100%	0.75
4	29	7	20	3	1	0.0133	1	0.0044	1	0.332	0.8333	50%	100%	0.75
4	29	7	100	3	1	1.18E-04	1	1.55E-05	1	0.2483	1	75%	100%	0.875
4	29	7	2	1	1	0.5893	0.556	0.2248	1	0.5446	0.5	50%	100%	0.75
4	29	7	10	1	1	0.332	0.8889	0.0949	1	0.4608	0.6667	50%	100%	0.75
4	29	7	20	1	1	0.1143	1	0.036	1	0.4004	0.6667	50%	100%	0.75
4	29	7	100	1	1	0.0026	1	0.001	1	0.2564	0.8333	75%	100%	0.88
4	29	7	2	3	3	0.3504	0.963	0.1152	1	0.466	0.6667	50%	100%	0.75
4	29	7	10	3	3	0.0699	1	0.0205	1	0.326	1	100%	100%	1
4	29	7	20	3	3	0.014	1	0.0047	1	0.2794	1	100%	100%	1
4	29	7	100	3	3	1.70E-04	1	1.59E-05	1	0.188	1	100%	100%	1
6	30	6	2	3	1	0.2923	0.8889	0.1439	1	0.4692	0.6667	100%	50%	0.75
6	30	6	10	3	1	0.0401	1	0.0399	1	0.3459	0.6667	100%	100%	1
6	30	6	20	3	1	0.0047	1	0.008	1	0.2286	1	100%	100%	1
6	30	6	100	3	1	4.33E-05	1	6.29E-05	1	0.0531	1	100%	100%	1
6	30	6	2	1	1	0.4952	0.5456	0.4291	0.8333	0.53	0.5	50%	50%	0.5
6	30	6	10	1	1	0.1416	1	0.1171	1	0.4348	0.6667	50%	100%	0.75
6	30	6	20	1	1	0.136	1	0.057	1	0.4541	0.6667	50%	100%	0.75
6	30	6	100	1	1	4.05E-04	1	7.78E-04	1	0.1331	1	100%	100%	1
6	30	6	2	3	3	0.3775	0.889	0.1601	1	0.4969	0.6667	50%	100%	0.75
6	30	6	10	3	3	0.0441	1	0.0492	1	0.3217	1	100%	100%	1
6	30	6	20	3	3	0.0046	1	0.0059	1	0.2477	1	100%	100%	1
6	30	6	100	3	3	5.79E-05	1	6.36E-05	1	0.0531	1	100%	100%	1
9	32	4	2	3	1	0.2535	0.9167	0.2668	1	0.4535	0.6667	75%	50%	0.62
9	32	4	10	3	1	0.0141	1	0.0249	1	0.2772	1	100%	100%	1
9	32	4	20	3	1	0.0016	1	0.0021	1	0.1684	1	100%	100%	1
9	32	4	100	3	1	2.25E-05	1	3.61E-05	1	0.0559	1	100%	100%	1
9	32	4	2	1	1	0.456	1	0.3827	1	0.513	0.5	50%	50%	0.5
9	32	4	10	1	1	0.1307	1	0.1646	1	0.434	0.6667	50%	100%	0.75
9	32	4	20	1	1	0.0378	1	0.0638	1	0.3711	0.6667	50%	100%	0.75
9	32	4	100	1	1	1.36E-04	1	2.47E-04	1	0.1022	1	100%	100%	1
9	32	4	2	3	3	0.2549	1	0.2763	1	0.5	0.6667	50%	100%	0.75
9	32	4	10	3	3	0.029	1	0.0293	1	0.2803	1	100%	100%	1
9	32	4	20	3	3	0.0016	1	0.0021	1	0.1684	1	100%	100%	1
9	32	4	100	3	3	2.58E-05	1	3.62E-05	1	0,0	1	100%	100%	1

By keeping all the training parameters constant, save from the number of Epochs, more data is gathered regarding the model's performance, and the cause-and-effect relationship between the number of epochs and the model's accuracy is made clear.

By comparing the results of the first row with those of the second row of *Table 3*, it is observed that increasing the number of Epochs yields a Loss reduction, but still not adequate enough. The same logic of increasing the number of epochs and keeping all the other parameters constant is employed until the fourth row of *Table 3*.

Now that the effects of the number of Epochs are observed in the evaluation results, the base case is made use of again, but with a different number of training steps (*Table 3*, row 5). By increasing the number of Epochs once again and keeping all other parameters constant, rows 6-8 are created.

By comparing the new evaluation results with their respective epoch counterparts, it is observed that a change of 300% in the number of training steps has a larger effect at a lower number of epochs and is made almost irrelevant as the number of epochs reaches higher values.

Returning to the base case and increasing the number of validation steps while employing the same process of incrementing the number of epochs, we get rows 9-12. A comparison is once again made between the base case and its epoch increments with respect to the newest evaluation results. Once again, tripling the number of validation steps does not seem to affect the training metrics (Loss, Accuracy, Validation Loss, and Validation Accuracy) in an impactful way; however, as the number of epochs increases, the model achieves better evaluation metrics (Evaluation Loss, Evaluation Accuracy, Sensitivity, Specificity, and ROC AUC) faster. Also note that while from row 10 onwards, the model correctly classifies the entire test set, the Evaluation Loss still improves. In other words, the training parameters row 10 do not guarantee 100% prediction accuracy every time on every test/prediction set. The same goes for any table containing a "perfect score."

In the next stage of this process, the results for different numbers of Folds are presented and compared to those above. Concerning the base case, the number of Folds/Iterations and, consequently, the number of Training and Validation samples per iteration are altered.

By directly comparing row 13 and the base case, it would seem that the model favors K=9 Folds over the base case's K=4.

Same as previously, row 14 through row 16 display the metrics for number of training samples = 3 and number of validation samples = 1, row 17 through row 20 for number of training samples = 1 and number of validation samples = 1 and finally, row 21 through row 24 for number of training samples = 3 and number of validation samples = 3, all for an incrementing number of epochs.

By making a direct, one-to-one comparison to the respective K=4 tables, the suspicion that the model favors K=9 over K=4 is confirmed.

Overall, apart from exceptions regarding training evaluation metrics, the model performs better when the data is partitioned in K=9 folds than when it is partitioned in K=4.

The resulting remarks from the K=6 metrics are omitted to avoid overloading this section with overlapping information. The critical observation is that overall, K=6 wielded the best results consistently compared to the respective number of epochs, training, and validation samples per iteration for both K=4 and K=9. The values included in *Tables 4 and 5* describe the best results attained by the procedure examined in the methodology section. While the training metrics are inferior to those displayed in row 16, the fact that it displays a lower evaluation loss deems the following the best-achieved result.

*Table 4: Training parameters*

Folds /Iterations	Training samples/iteration	Val. samples /Iteration	Epochs	Training steps	Val. steps
6	30	6	100	3	1

*Table 5: Evaluation results*

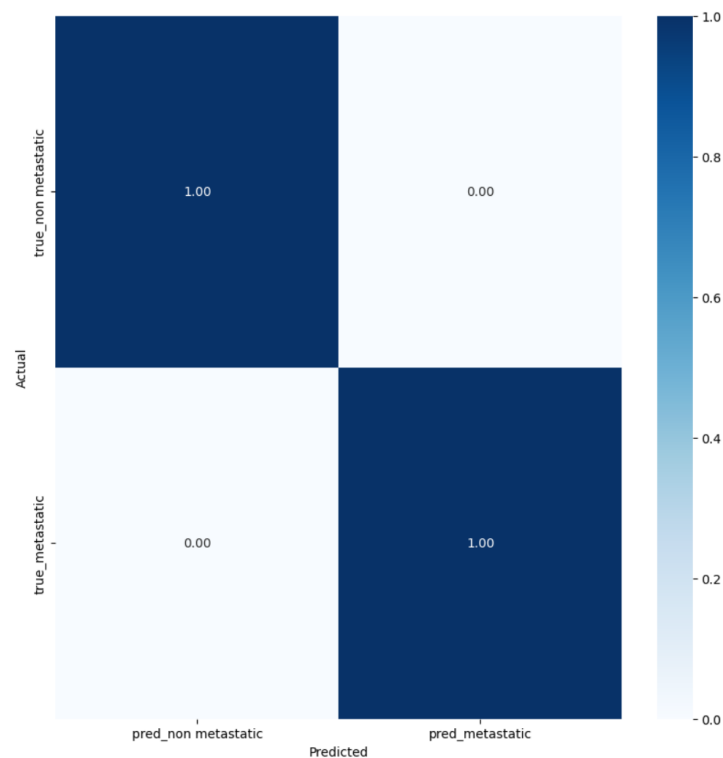
Loss	Accuracy	Val. Loss	Val. Accuracy	Loss (Evaluation )	Accuracy (Evaluation )	Sensitivity	Specificity	ROC AUC
4.33E-05	1	6.29E-05	1	0.0531	1	100%	100%	1

Upon inspection, the significance of the results is made clear. With merely 45 sample images that endured dimensionality reduction at the preprocessing stage, the

model achieved 100% accuracy at the evaluation stage, accompanied by a loss of 5.3%, *Table 5*.

The 100% accuracy is the result of the model correctly classifying the entirety of the evaluation set. These metrics can be improved further with the introduction of more data that will result in a further reduction in the evaluation loss.

The confusion matrix is another way to display the evaluation results efficiently. From it, it was deduced that the True Positive (TP) and the True Negative (TN) of the actual values are both 1, indicating truly accurate classification predictions on the evaluation set, *Figure 23*.



*Figure 23: Resulting Confusion matrix.*

The resulting ROC curve is also displayed in *Figure 24*, once again indicating 100% prediction accuracy on the evaluation (test) set.

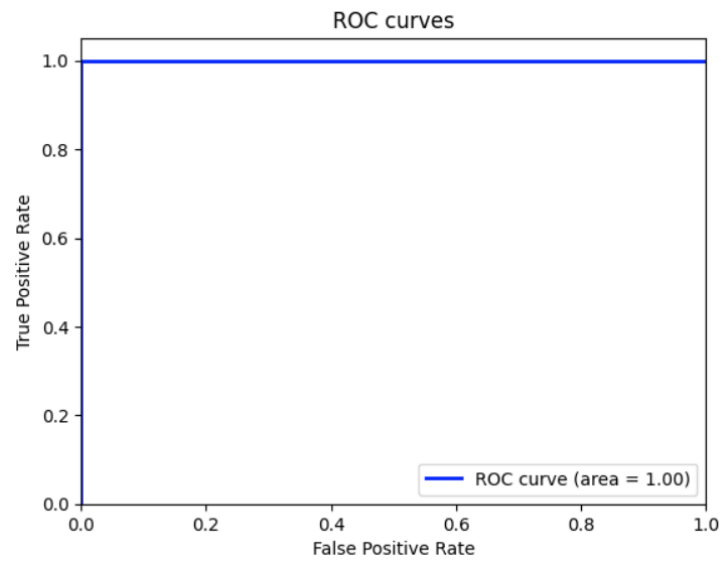


Figure 24: Resulting ROC Curve.

Finally, by calling the model to generate a prediction on the prediction dataset, a result best described by *Figure 25* surfaces. The model confidence in this instance that the input image is metastatic is 99.71%, which is confirmed to be the case in hindsight.

This image is 99.71% metastatic.

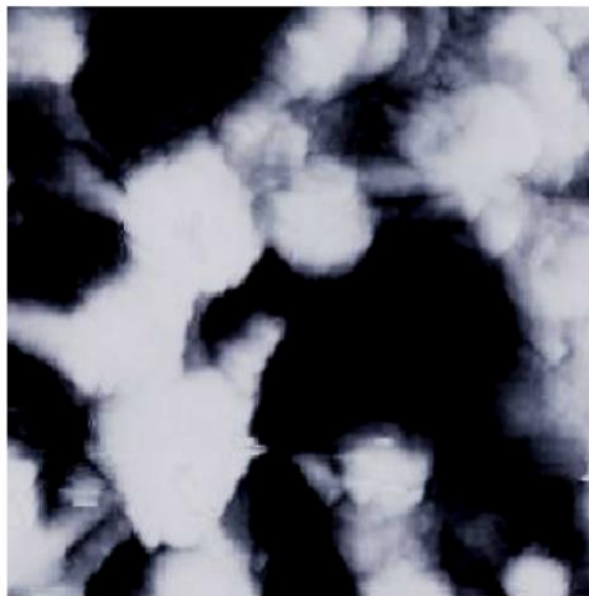


Figure 25: Generated Prediction result on the prediction dataset

## 6. Discussion

Histological image analysis reveals significant physiological and pathological information or knowledge from medical images, making it a critical tool in medical study, disease diagnosis, and treatment. In parallel, DL, especially CNNs, is now the most widely used method for image recognition and is well adopted in digital pathology, for example, in analyzing complex images for pathological grading in cancer research.

However, there are several limitations of the histological staining step; standardizing the imaging and tissue section staining process across multiple labs leads to many inconsistencies in pathological images and makes developing robust, stable, and broadly applicable models even more complex [56].

On the other hand, AFM is an essential tool for probing the nanoscale properties of biological samples, including pathological tissue, in a non-invasive manner. Therefore, there is growing interest in translating the AFM from the laboratory environment to clinical practice as a diagnostic tool.

This work considers the potential of the DL method to expand the AFM characterization of metastatic and non-metastatic cancer colorectal tissue samples for fast and bias-free characterization. This task is highly demanding because the analysis of AFM images functions best when many images are available for training and for predicting unidentified image data. However, the available datasets are limited, and at the same time, AFM microscopy is relatively slow, further limiting the number of images that can be acquired.

Alternative strategies can be the use of high-speed AFM microscopy for imaging [57] or applying DL methods to enhance the resolution of low-resolution AFM images, such as very deep super-resolution (VDSR) due to the large number of convolutional layers, e.g., twenty in the work of Azuri et al. [29].

The main reason for employing CNN in this work over other traditional neural networks is the weight-sharing feature, which decreases the number of trainable network parameters, permitting the network to improve generalization and avoid overfitting. Various techniques are proposed in the literature to overcome the overfitting drawback and improve the DL framework performance with few training samples [29].

Among them is the inclusion of a 'transition' module in CNNs, where filters of varying sizes are used to encourage class-specific filters at multiple spatial resolutions followed by global average pooling [58]. In our study, the over-fitting is reduced by finding the optimal set of hyperparameters for the model concerning k validation sets.

Alternative machine learning algorithms like decision trees were applied by Mikhail Petrov et al. to analyze AFM adhesion maps to distinguish precancerous and cancerous cervical cells with rather good precision (AUC, accuracy, sensitivity, and specificity are 0.93, 83%, 92%, and 78%, respectively) [34].

Parameter selection has a significant impact on CNN performance. Any slight change in hyperparameter values will affect overall CNN performance. As a result, careful parameter selection is a critical issue that should be considered during optimization scheme development. In our case, those hyperparameters were predetermined and were the same ones used for the model to partake in the ImageNet dataset competition. That way, we also use the pertained weights, the result of training over a massive dataset, to compensate for having a small dataset.

While it is hard to come to a concrete conclusion on the actual validity of the method presented in this work because of the small size of the test set, a comparison is attempted with the results of relevant publications. For instance, in the Analytical performance of ThyroSeq v3 Genomic Classifier for cancer diagnosis in thyroid nodules by Marina N. Nikiforova et al. [59], the sensitivity, specificity, and accuracy scores were 93.9%, 89.4%, and 92.1%, respectively. With these specs, the methodology pursued indicated clinical suitability. Perhaps in light of that statement, the results of this study also deem it suitable. With such a small dataset, it may be impossible to tell. A comparison of most interest is between the current results and those of Gavriil, V. et al. [5].

On the one hand, it is stated that the theta distribution skewness displays a 99.99% confidence, indicating a loss of 0.01, a loss comparable to our 0.0531, at least in terms of order. On the other hand, the variograms method identifies metastasis with 99.99999% confidence, making it a more accurate method for sure. Despite that fact, we are confident that by adding more data to the set, the CNN approach of extracting a prediction is entirely possible to approximate the variogram method accuracy while

retaining simplicity, as variogram analysis is a highly complex mathematical method requiring specialists present at execution times.

This work presents a novel approach for the early identification of metastasis from primary colorectal cancer histological tissues by processing AFM cancer tissue images using DL algorithms to extract a timely yet accurate prediction. It provides an example of how the rapid development of the DL space can bring it a step closer to cancer prognosis based on nanosized (97.7 nm) precision imaging. The limitations and challenges for applying deep learning in oncology, including the lack of a large number of high-quality labeled data, are being worked on with the singular goal of successfully combating the disease.



## 7. References

- [1] Sung H, Ferlay J, Siegel RL, Laversanne M, Soerjomataram I, Jemal A, et al. Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. *CA Cancer J Clin* 2021;71:209–49. doi:10.3322/caac.21660.
- [2] Wang J, Gao M, Yang L, Huang Y, Wang J, Wang B, et al. Cell recognition based on atomic force microscopy and modified residual neural network. *J Struct Biol* 2023;215:107991. doi:https://doi.org/10.1016/j.jsb.2023.107991.
- [3] Agus DB, Alexander JF, Arap W, Ashili S, Aslan JE, Austin RH, et al. A physical sciences network characterization of non-tumorigenic and metastatic cells. *Sci Rep* 2013;3:1449. doi:10.1038/srep01449.
- [4] Blank A, Roberts 2nd DE, Dawson H, Zlobec I, Lugli A. Tumor Heterogeneity in Primary Colorectal Cancer and Corresponding Metastases. Does the Apple Fall Far From the Tree? *Front Med* 2018;5:234. doi:10.3389/fmed.2018.00234.
- [5] Gavriil V, Ferraro A, Cefalas A-C, Kollia Z, Pepe F, Malapelle U, et al. Nanoscale Prognosis of Colorectal Cancer Metastasis from AFM Image Processing of Histological Sections. *Cancers (Basel)* 2023;15. doi:10.3390/cancers15041220.
- [6] Zaharchuk G, Gong E, Wintermark M, Rubin D, Langlotz CP. Deep learning in neuroradiology. *Am J Neuroradiol* 2018;39:1776–84. doi:10.3174/ajnr.A5543.
- [7] Lu J, Tan L, Jiang H. Review on convolutional neural network (CNN) applied to plant leaf disease classification. *Agric* 2021;11:1–18. doi:10.3390/agriculture11080707.
- [8] Taye MM. Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions. *Computers* 2023;12. doi:10.3390/computers12050091.
- [9] Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. vol. 8. Springer International Publishing; 2021. doi:10.1186/s40537-021-00444-8.
- [10] Dillekås H, Rogers MS, Straume O. Are 90% of deaths from cancer caused by metastases? *Cancer Med* 2019;8:5574–6. doi:10.1002/cam4.2474.
- [11] Staff BTAP. Colon Cancer Mortality Rates: Predictions Across the European Union and United Kingdom n.d. <https://ascopost.com/news/january-2024/colon-cancer-mortality-rates-predictions-across-the-european-union-and-united-kingdom/>.
- [12] Xia F, Youcef-Toumi K. Review: Advanced Atomic Force Microscopy Modes for Biomedical Research. *Biosensors* 2022;12:1–24. doi:10.3390/bios12121116.
- [13] Kawasaki H, Itoh T, Takaku Y, Suzuki H, Kosugi I, Meguro S, et al. The NanoSuit method: a novel histological approach for examining paraffin sections in a nondestructive manner by correlative light and electron microscopy. *Lab Invest* 2020;100:161–73. doi:10.1038/s41374-019-0309-7.
- [14] Shreve JT, Khanani SA, Haddad TC. Artificial Intelligence in Oncology: Current Capabilities, Future Opportunities, and Ethical Considerations. *Am Soc Clin Oncol Educ B* 2022;842–51. doi:10.1200/edbk\_350652.
- [15] Anaya-Isaza A, Mera-Jiménez L, Zequera-Diaz M. An overview of deep learning in medical imaging. *Informatics Med Unlocked* 2021;26. doi:10.1016/j.imu.2021.100723.
- [16] Yang X, Yang Y, Zhang Z, Li M. Deep Learning Image Recognition-Assisted Atomic Force Microscopy for Single-Cell Efficient Mechanics in Co-culture Environments. *Langmuir* 2024;40:837–52. doi:10.1021/acs.langmuir.3c03046.
- [17] Zhu G, Jiang B, Tong L, Xie Y, Zaharchuk G, Wintermark M. Applications of deep learning to neuro-imaging techniques. *Front Neurol* 2019;10:1–13. doi:10.3389/fneur.2019.00869.
- [18] Jaron Collis. Glossary of Deep Learning: Bias n.d. <https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2>.
- [19] Han S-H, Kim KW, Kim S, Youn YC. Artificial Neural Network: Understanding the Basic

- Concepts without Mathematics. *Dement Neurocognitive Disord* 2018;17:83. doi:10.12779/dnd.2018.17.3.83.
- [20] Patil A, Rane M. Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition. *Smart Innov Syst Technol* 2021;195:21–30. doi:10.1007/978-981-15-7078-0\_3.
- [21] Chen L, Li S, Bai Q, Yang J, Jiang S, Miao Y. Review of Image Classification Algorithms Based on Convolutional Neural Networks. *Remote Sens* 2021;13. doi:10.3390/rs13224712.
- [22] Potrimba P. 71e4494f28518628ef1f36686a8ab02f3697b7ab @ blog.roboflow.com n.d. <https://blog.roboflow.com/what-is-a-convolutional-neural-network/>.
- [23] Skalski P. Gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9 @ towardsdatascience.com n.d. <https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9>.
- [24] Mohsen Nabil. Unveiling-the-diversity-a-comprehensive-guide-to-types-of-cnn-architectures-9d70da0b4521 @ medium.com n.d. <https://medium.com/@navarai/unveiling-the-diversity-a-comprehensive-guide-to-types-of-cnn-architectures-9d70da0b4521>.
- [25] Understanding how convolutional layers work n.d. <https://datascience.stackexchange.com/questions/80436/understanding-how-convolutional-layers-work>.
- [26] Yingge H, Ali I, Lee KY. Deep neural networks on chip - A survey. *Proc - 2020 IEEE Int Conf Big Data Smart Comput BigComp 2020* 2020:589–92. doi:10.1109/BigComp48618.2020.00016.
- [27] Prathap P. The secret to understanding-CNNs convolution feature maps pooling and fully connected layers. n.d. <https://medium.com/@prajeeshprathap/the-secret-to-understanding-cnns-convolution-feature-maps-pooling-and-fully-connected-layers-97055431a847%0APrajeesh Prathap>.
- [28] Rguibi Z, Hajami A, Zitouni D, Elqaraoui A, Bedraoui A. CXAI: Explaining Convolutional Neural Networks for Medical Imaging Diagnostic. *Electron* 2022;11:1–19. doi:10.3390/electronics11111775.
- [29] Azuri I, Rosenhek-Goldian I, Regev-Rudzki N, Fantner G, Cohen SR. The role of convolutionsl neural networks in scanning probe microscopy: a review. *Beilstein J Nanotechnol* 2021;12:878–901. doi:10.3762/bjnano.12.66.
- [30] Najera J, Rosenberger MR, Datta M. Atomic Force Microscopy Methods to Measure Tumor Mechanical Properties. *Cancers (Basel)* 2023;15. doi:10.3390/cancers15133285.
- [31] Rahman Laskar MA, Celano U. Scanning probe microscopy in the age of machine learning. *APL Mach Learn* 2023;1. doi:10.1063/5.0160568.
- [32] Minelli E, Ciasca G, Sassun TE, Antonelli M, Palmieri V, Papi M, et al. A fully-automated neural network analysis of AFM force-distance curves for cancer tissue diagnosis. *Appl Phys Lett* 2017;111. doi:10.1063/1.4996300.
- [33] Prasad S, Rankine A, Prasad T, Song P, Dokukin ME, Makarova N, et al. Atomic Force Microscopy Detects the Difference in Cancer Cells of Different Neoplastic Aggressiveness via Machine Learning. *Adv NanoBiomed Res* 2021;1:1–12. doi:10.1002/anbr.202000116.
- [34] Petrov M, Sokolov I. Machine Learning Allows for Distinguishing Precancerous and Cancerous Human Epithelial Cervical Cells Using High-Resolution AFM Imaging of Adhesion Maps. *Cells* 2023;12. doi:10.3390/cells12212536.
- [35] Raila T, Penkauskas T, Ambrulevičius F, Jankunec M, Meškauskas T, Valinčius G. AI-based atomic force microscopy image analysis allows to predict electrochemical impedance spectra of defects in tethered bilayer membranes. *Sci Rep* 2022;12:1–11. doi:10.1038/s41598-022-04853-4.
- [36] Martinez-rivas A, Formosa-dague C, Emilio L, Espinal M, Thomas- O, Carillo K. Nanobiomechanical data classified by Deep learning based on convolutional neural

- networks 2023.
- [37] Rade J, Zhang J, Sarkar S, Krishnamurthy A, Ren J, Sarkar A. Deep Learning for Live Cell Shape Detection and Automated AFM Navigation. *Bioengineering* 2022;9:1–19. doi:10.3390/bioengineering9100522.
- [38] Alldritt B, Hapala P, Oinonen N, Urtev F, Krejci O, Canova FF, et al. Automated structure discovery in atomic force microscopy. *Sci Adv* 2020;6:1–10. doi:10.1126/sciadv.aay6913.
- [39] Carracedo-Cosme J, Romero-Muñiz C, Pérez R. A deep learning approach for molecular classification based on afm images. *Nanomaterials* 2021;11:1–22. doi:10.3390/nano11071658.
- [40] Kocur V, Hegrová V, Patočka M, Neuman J, Herout A. Correction of AFM data artifacts using a convolutional neural network trained with synthetically generated data. *Ultramicroscopy* 2023;246:113666. doi:https://doi.org/10.1016/j.ultramic.2022.113666.
- [41] Binnig G, Quate CF, Gerber C. Atomic Force Microscope. *Phys Rev Lett* 1986;56:930–3. doi:10.1103/PhysRevLett.56.930.
- [42] Sakai K. Atomic Force Microscope (AFM). *Meas Tech Pract Colloid Interface Phenom* 2019;56:51–7. doi:10.1007/978-981-13-5931-6\_8.
- [43] Gavriil VE. Optical and electrical properties of functional materials at the nanoscale. n.d. doi:10.12681/eadd/49922.
- [44] Soule D. Python Programming: An Overview of Scientific Literature n.d. <https://www.linkedin.com/pulse/python-programming-overview-scientific-literature-damien-soulé/>.
- [45] Google Colaboratory n.d. <https://colab.google/>.
- [46] About Keras 3 n.d. <https://keras.io/about/>.
- [47] Keras Tutorial – Ultimate Guide to Deep Learning n.d. <https://data-flair.training/blogs/keras-introduction/>.
- [48] Keras: The high-level API for TensorFlow n.d. <https://www.tensorflow.org/guide/keras>.
- [49] Keras layers – Parameters and Properties n.d. <https://data-flair.training/blogs/keras-layers/>.
- [50] Keras Layers – Everything you need to Know n.d. <https://techvidvan.com/tutorials/keras-layers/>.
- [51] Build your first Deep Learning Basic model using Keras, Python and Tensorflow step by step approach n.d. <https://medium.com/analytics-vidhya/build-our-first-deep-learning-basic-model-using-keras-python-and-tensorflow-step-by-step-approach-d61c41b8a866>.
- [52] Szegedy C, Ioffe S, Vanhoucke V, Alemi AA. Inception-v4, inception-ResNet and the impact of residual connections on learning. 31st AAAI Conf Artif Intell AAAI 2017 2017:4278–84. doi:10.1609/aaai.v31i1.11231.
- [53] Advanced Guide to Inception v3 n.d. <https://cloud.google.com/tpu/docs/inception-v3-advanced>.
- [54] Vihar Kurama. A Review of Popular Deep Learning Architectures: ResNet, InceptionV3, and SqueezeNet n.d. <https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/>.
- [55] Toshendra Sharma. How to build a machine learning pipeline? n.d. <https://www.globaltechcouncil.org/blockchain/how-to-build-a-machine-learning-pipeline/>.
- [56] de Haan K, Zhang Y, Zuckerman JE, Liu T, Sisk AE, Diaz MFP, et al. Deep learning-based transformation of H&E stained tissues into special stains. *Nat Commun* 2021;12:1–13. doi:10.1038/s41467-021-25221-2.
- [57] Ando T. High-Speed Atomic Force Microscopy (AFM) BT - Encyclopedia of Biophysics. In: Roberts GCK, editor., Berlin, Heidelberg: Springer Berlin Heidelberg; 2013, p. 984–7. doi:10.1007/978-3-642-16712-6\_478.
- [58] Akbar S, Peikari M, Salama S, Nofech-Mozes S, Martel AL. The transition module: a method

- for preventing overfitting in convolutional neural networks. *Comput Methods Biomech Biomed Eng Imaging Vis* 2019;7:260–5. doi:10.1080/21681163.2018.1427148.
- [59] Nikiforova MN, Mercurio S, Wald AI, Barbi de Moura M, Callenberg K, Santana-Santos L, et al. Analytical performance of the ThyroSeq v3 genomic classifier for cancer diagnosis in thyroid nodules. *Cancer* 2018;124:1682–90. doi:10.1002/cncr.31245.

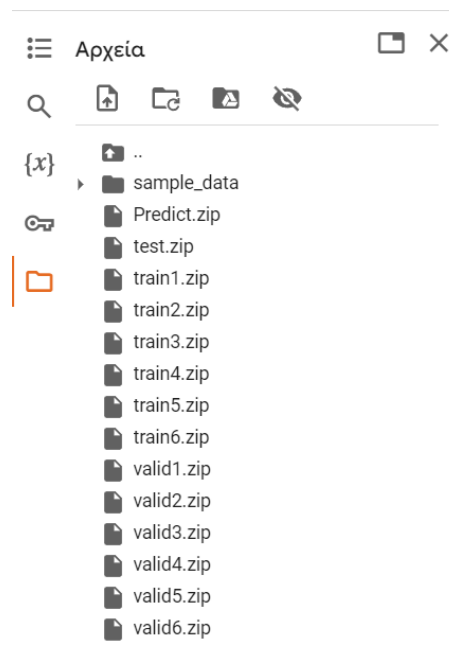
## 8. Appendix

### 8.1 Python code

In this section the Python code is presented along with a step-by-step explanation of it. The first K of the K-fold validation needs to be determined for the program to work. For the purposes of this demonstration, K=6 is assumed. With K set, the images are placed in folders to be converted into .zip files. The following .zip files are required:

1. train1.zip – train6.zip, for the purposes of model training
2. valid1.zip-valid6.zip, for the purposes of model validation during the training process
3. test.zip, for the evaluation of the model
4. Predict.zip for presentation purposes

Those .zip files are then uploaded in the files section of the Google Colab interface, displayed in *Figure 26*.



*Figure 26: Uploaded data*

It all begins with installing necessary libraries,

```
!pip3 install tensorflow tensorflow_hub matplotlib seaborn numpy pandas scikit-learn imblearn
```

followed up by importing said libraries along with additional ones, which provide the methods needed for the implementation.

```
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from tensorflow.keras.utils import get_file
from sklearn.metrics import roc_curve, auc, confusion_matrix, accuracy_score
from imblearn.metrics import sensitivity_score, specificity_score

import os
import glob
import zipfile
import random
```

Then the seed is instantiated as follows, so that consistent results after multiple runs are received.

```
tf.random.set_seed(7)
np.random.seed(7)
random.seed(7)
```

After this initial setup is complete, extracting the data from the .zip files occurs. This takes place for each .zip file, as displayed by the following line of code.

```
with zipfile.ZipFile('train1.zip', 'r') as zip_ref:
    zip_ref.extractall('data')
```

The process repeats for each .zip file. In the files section, a new data folder has appeared, containing our data, partitioned into their respective folders , *Figure 27*.

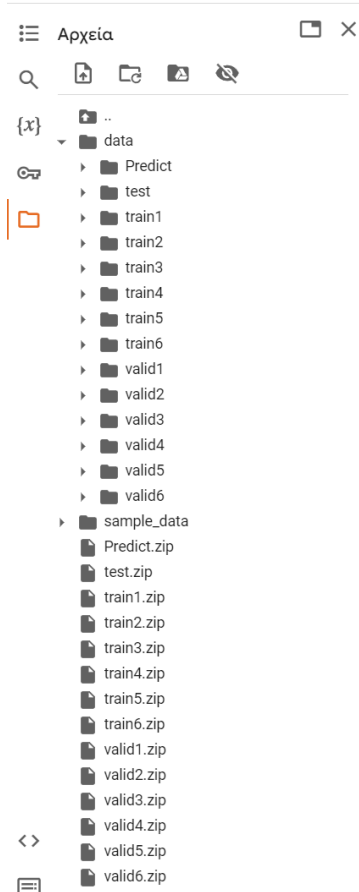


Figure 27: Extracted data

With access to our data, a function is created, its purpose being to generate a metadata .csv file for each set when called.

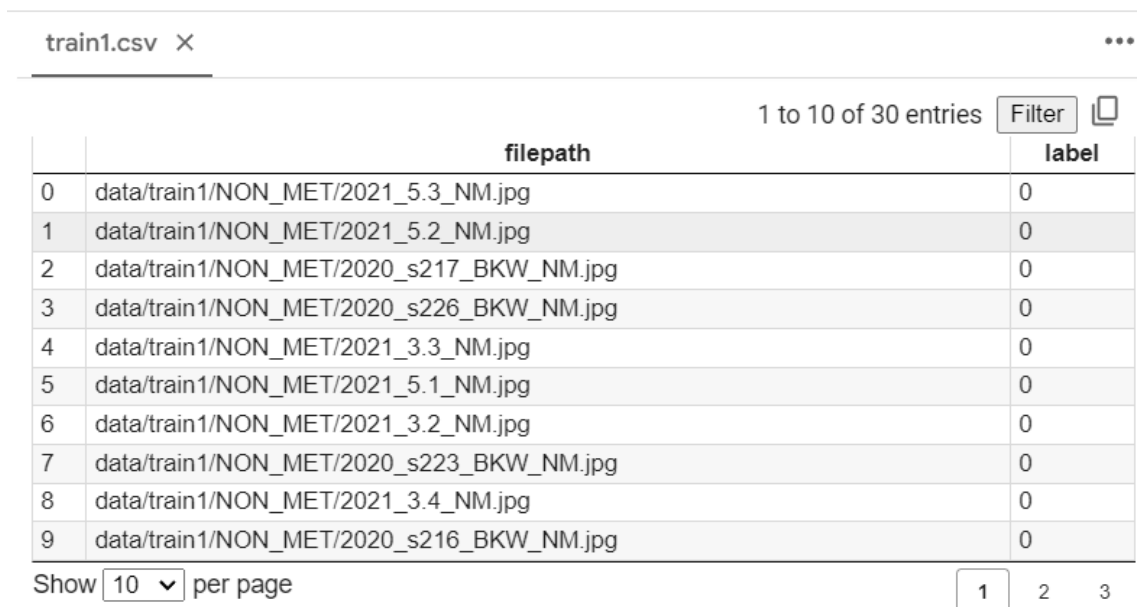
```
def generate_csv(folder, label2int):  
    folder_name = os.path.basename(folder)  
    labels = list(label2int)  
  
    df = pd.DataFrame(columns=["filepath", "label"])  
    i = 0  
    for label in labels:  
        print("Reading", os.path.join(folder, label, "*"))  
        for filepath in glob.glob(os.path.join(folder, label, "*")):  
            df.loc[i] = [filepath, label2int[label]]  
            i += 1  
    output_file = f"{folder_name}.csv"
```

```
print("Saving", output_file)
df.to_csv(output_file)
```

When indeed called, it generates a .csv file to our files section. The columns of the .csv are the index, the file path of the image and its classification as metastatic (1) or non-metastatic (0).

The function is called for all the data files and generates the metadata .csv files accordingly. For demonstration purposes, train1.csv is displayed in *Figure 28*.

```
generate_csv("data/train1", {"NON_MET": 0, "MET": 1})
```



	filepath	label
0	data/train1/NON_MET/2021_5.3_NM.jpg	0
1	data/train1/NON_MET/2021_5.2_NM.jpg	0
2	data/train1/NON_MET/2020_s217_BKW_NM.jpg	0
3	data/train1/NON_MET/2020_s226_BKW_NM.jpg	0
4	data/train1/NON_MET/2021_3.3_NM.jpg	0
5	data/train1/NON_MET/2021_5.1_NM.jpg	0
6	data/train1/NON_MET/2021_3.2_NM.jpg	0
7	data/train1/NON_MET/2020_s223_BKW_NM.jpg	0
8	data/train1/NON_MET/2021_3.4_NM.jpg	0
9	data/train1/NON_MET/2020_s216_BKW_NM.jpg	0

Figure 28: train1.csv

The next step of the process, utilizes the `from_tensor_slices()` method from the `tf.data` API to load these metadata files and create the datasets needed for training.

```
train1_metadata_filename = "train1.csv"
valid1_metadata_filename = "valid1.csv"

train2_metadata_filename = "train2.csv"
valid2_metadata_filename = "valid2.csv"
```



```
train3_metadata_filename = "train3.csv"
valid3_metadata_filename = "valid3.csv"

train4_metadata_filename = "train4.csv"
valid4_metadata_filename = "valid4.csv"

train5_metadata_filename = "train5.csv"
valid5_metadata_filename = "valid5.csv"

train6_metadata_filename = "train6.csv"
valid6_metadata_filename = "valid6.csv"

df_train1 = pd.read_csv(train1_metadata_filename)
df_valid1 = pd.read_csv(valid1_metadata_filename)

df_train2 = pd.read_csv(train2_metadata_filename)
df_valid2 = pd.read_csv(valid2_metadata_filename)

df_train3 = pd.read_csv(train3_metadata_filename)
df_valid3 = pd.read_csv(valid3_metadata_filename)

df_train4 = pd.read_csv(train4_metadata_filename)
df_valid4 = pd.read_csv(valid4_metadata_filename)

df_train5 = pd.read_csv(train5_metadata_filename)
df_valid5 = pd.read_csv(valid5_metadata_filename)

df_train6 = pd.read_csv(train6_metadata_filename)
df_valid6 = pd.read_csv(valid6_metadata_filename)

n_training1_samples = len(df_train1)
n_validation1_samples = len(df_valid1)
```

```

n_training2_samples = len(df_train2)
n_validation2_samples = len(df_valid2)

n_training3_samples = len(df_train3)
n_validation3_samples = len(df_valid3)

n_training4_samples = len(df_train4)
n_validation4_samples = len(df_valid4)

n_training5_samples = len(df_train5)
n_validation5_samples = len(df_valid5)

n_training6_samples = len(df_train6)
n_validation6_samples = len(df_valid6)

print("Number of training samples [1]:", n_training1_samples)
print("Number of validation samples [1]:", n_validation1_samples)
print("Number of training samples [2]:", n_training2_samples)
print("Number of validation samples [2]:", n_validation2_samples)
print("Number of training samples [3]:", n_training3_samples)
print("Number of validation samples [3]:", n_validation3_samples)
print("Number of training samples [4]:", n_training4_samples)
print("Number of validation samples [4]:", n_validation4_samples)
print("Number of training samples [5]:", n_training5_samples)
print("Number of validation samples [5]:", n_validation5_samples)
print("Number of training samples [6]:", n_training6_samples)
print("Number of validation samples [6]:", n_validation6_samples)

train1_ds = tf.data.Dataset.from_tensor_slices((df_train1["filepath"], df_train1["label"]))
valid1_ds = tf.data.Dataset.from_tensor_slices((df_valid1["filepath"], df_valid1["label"]))

train2_ds = tf.data.Dataset.from_tensor_slices((df_train2["filepath"], df_train2["label"]))
valid2_ds = tf.data.Dataset.from_tensor_slices((df_valid2["filepath"], df_valid2["label"]))

```

```
train3_ds = tf.data.Dataset.from_tensor_slices((df_train3["filepath"], df_train3["label"]))
valid3_ds = tf.data.Dataset.from_tensor_slices((df_valid3["filepath"], df_valid3["label"]))

train4_ds = tf.data.Dataset.from_tensor_slices((df_train4["filepath"], df_train4["label"]))
valid4_ds = tf.data.Dataset.from_tensor_slices((df_valid4["filepath"], df_valid4["label"]))

train5_ds = tf.data.Dataset.from_tensor_slices((df_train5["filepath"], df_train5["label"]))
valid5_ds = tf.data.Dataset.from_tensor_slices((df_valid5["filepath"], df_valid5["label"]))

train6_ds = tf.data.Dataset.from_tensor_slices((df_train6["filepath"], df_train6["label"]))
valid6_ds = tf.data.Dataset.from_tensor_slices((df_valid6["filepath"], df_valid6["label"]))
```

In the next cell a couple of things take place. Two functions are created and called, with the objective of transforming the dataset into a MapDataset form, by utilizing the `map()` method on the `process_path()` function.

The `decode_img()` function, first converts the images' compressed string to a 3D uint8 tensor (channels=3 is chosen even though the images are grayscale for minimal data loss). Then using the `convert_image_dtype` the floats are normalized in the [0,1] range, followed by a `resize` in the desired 299x299 resolution.

The `process_path()` function, loads the raw data from the file as string, the calls the `decode_image()` function and returns the decoded image along with the appropriate label.

```
def decode_img(img):
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return tf.image.resize(img, [299, 299])

def process_path(filepath, label):
    img = tf.io.read_file(filepath)
    img = decode_img(img)
    return img, label
```

```
train1_ds = train1_ds.map(process_path)
valid1_ds = valid1_ds.map(process_path)

train2_ds = train2_ds.map(process_path)
valid2_ds = valid2_ds.map(process_path)

train3_ds = train3_ds.map(process_path)
valid3_ds = valid3_ds.map(process_path)

train4_ds = train4_ds.map(process_path)
valid4_ds = valid4_ds.map(process_path)

train5_ds = train5_ds.map(process_path)
valid5_ds = valid5_ds.map(process_path)

train6_ds = train6_ds.map(process_path)
valid6_ds = valid6_ds.map(process_path)
```

As an example, by running,

```
for image, label in train1_ds.take(1):
    print("Image shape [1]:", image.shape)
    print("Label [1]:", label.numpy())
```

the result found in the console, for the first image in the `train1_ds`, can be viewed in *Figure 29*.

```
Image shape [1]: (299, 299, 3)
Label [1]: 0
```

*Figure 29: Console result, which provides the details of a `train1_ds` dataset.*

Up next, the training and validation datasets undergo the final transmutation before training. For that purpose, the `prepare_for_training()` function is employed. It shuffles the dataset then proceeds to partition in into batches. One might ponder on why implement

batches when the data is already partitioned in the .zip files. The answer lies in the fact that by utilizing batches, the road ahead for future upgrades that automate this whole process further is smoother, requiring little participation from the user. The methods used are `cache()`, used to save our preprocessed dataset into a local cache file (this will only preprocess it in the first epoch of training) and to lessen the computational load of each set (more efficient on larger dataset but a sound practice regardless), `shuffle()`, used to shuffle the images so samples are in a random order, `repeat()`, used each time the set is iterated over, to generate samples repeatedly (useful during training), `batch()`, used to partition the set based on the batch size and lastly `prefetch()`, used to fetch batches in the background while the model is in the training phase.

```
batch_size = 6
optimizer = "rmsprop"

def prepare_for_training(ds, cache=True, batch_size=6, shuffle_buffer_size=1000):
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()
    ds = ds.shuffle(buffer_size=shuffle_buffer_size)
    ds = ds.repeat()
    ds = ds.batch(batch_size)
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return ds

train1_ds = prepare_for_training(train1_ds, batch_size=batch_size, cache="train1-cached-data")
valid1_ds = prepare_for_training(valid1_ds, batch_size=batch_size, cache="valid1-cached-data")

train2_ds = prepare_for_training(train2_ds, batch_size=batch_size, cache="train2-cached-data")
valid2_ds = prepare_for_training(valid2_ds, batch_size=batch_size, cache="valid2-cached-data")

train3_ds = prepare_for_training(train3_ds, batch_size=batch_size, cache="train3-cached-data")
valid3_ds = prepare_for_training(valid3_ds, batch_size=batch_size, cache="valid3-cached-data")
```

```
train4_ds = prepare_for_training(train4_ds, batch_size=batch_size, cache="train4-cached-data")
valid4_ds = prepare_for_training(valid4_ds, batch_size=batch_size, cache="valid4-cached-data")

train5_ds = prepare_for_training(train5_ds, batch_size=batch_size, cache="train5-cached-data")
valid5_ds = prepare_for_training(valid5_ds, batch_size=batch_size, cache="valid5-cached-data")

train6_ds = prepare_for_training(train6_ds, batch_size=batch_size, cache="train6-cached-data")
valid6_ds = prepare_for_training(valid6_ds, batch_size=batch_size, cache="valid6-cached-data")
```

The following cell gets the images of the validation sets and plots the images alongside their corresponding classification label. The result displayed in *Figure 5* is similar to *Figure 21* of the 4.1 Dataset Optimization section of this report, but with batch size of 6.

```
batch1 = next(iter(valid1_ds))
batch2 = next(iter(valid2_ds))
batch3 = next(iter(valid3_ds))
batch4 = next(iter(valid4_ds))
batch5 = next(iter(valid5_ds))
batch6 = next(iter(valid6_ds))

class_names = ["non metastatic", "metastatic"]

def show_batch(batch):
    plt.figure(figsize=(10,10))
    for n in range(6):
        ax = plt.subplot(2,3,n+1)
        plt.imshow(batch[0][n])
        plt.title(class_names[batch[1][n].numpy()].title())
        plt.axis('off')

show_batch(batch1)
```

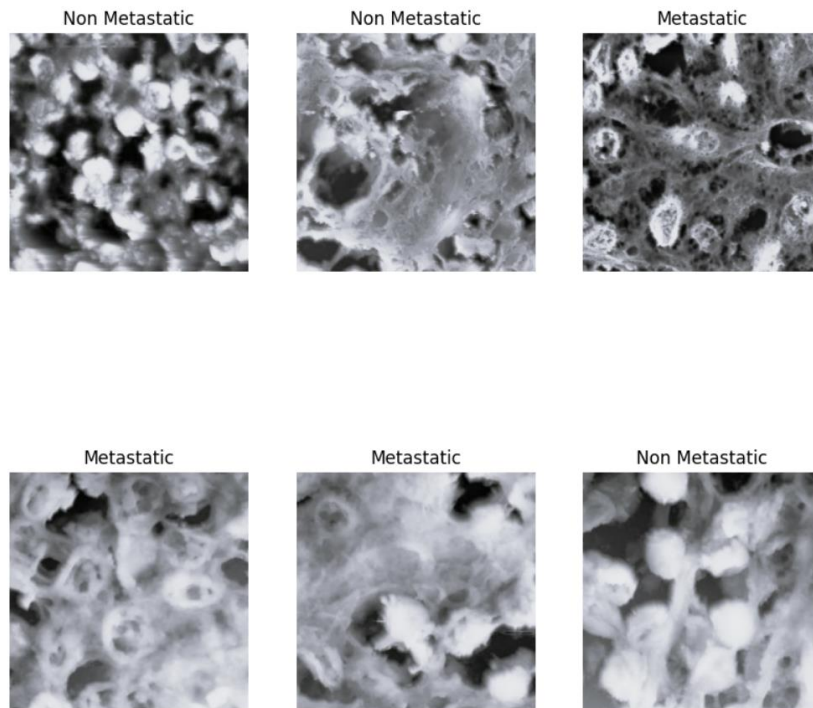


Figure 30: Preprocessed data batch.

With the data finally ready, the building of the model takes place. Since it's a binary classification problem, the loss parameter is assigned to binary cross entropy.

```

module_url = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4"
m = tf.keras.Sequential([
    hub.KerasLayer(module_url, output_shape=[2048], trainable=False),
    tf.keras.layers.Dense(1, activation="sigmoid")
])

m.build([None, 299, 299, 3])
m.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])
m.summary()

```

The `m.summary()` yields the following (Figure 6):

Model: "sequential"

Layer (type)	Output Shape	Param #
keras_layer (KerasLayer)	(None, 2048)	21802784
dense (Dense)	(None, 1)	2049

=====  
Total params: 21804833 (83.18 MB)  
Trainable params: 2049 (8.00 KB)  
Non-trainable params: 21802784 (83.17 MB)  
=====

*Figure 31: Model description*

Next the training begins for each iteration the K-fold validation method. Note that the best weights are saved via model checkpoint. Also, in this cell (and the ones for the other iterations) the user adjusts the training parameters.

```
model_name = f"metastatic-vs-non-metastatic_{batch_size}_{optimizer}"
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=os.path.join("logs", model_name))
modelcheckpoint = tf.keras.callbacks.ModelCheckpoint(model_name + "_{val_loss:.3f}.h5",
save_best_only=True, verbose=1)

history = m.fit(train1_ds, validation_data=valid1_ds,
                steps_per_epoch=3,
                #steps_per_epoch=n_training_samples // batch_size,
                validation_steps=1,
                #validation_steps=n_validation_samples // batch_size,
                verbose=1,
                epochs=100,
                callbacks=[tensorboard, modelcheckpoint])
```



Assuming training is complete, the evaluation of the model takes place afterwards. It begins with loading the test set. The procedure is the same as with the training and validation sets.

```
test_metadata_filename = "test.csv"
df_test = pd.read_csv(test_metadata_filename)
n_testing_samples = len(df_test)
print("Number of testing samples:", n_testing_samples)
test_ds = tf.data.Dataset.from_tensor_slices((df_test["filepath"], df_test["label"]))

def prepare_for_testing(ds, cache=True, shuffle_buffer_size=1000):
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()
    ds = ds.shuffle(buffer_size=shuffle_buffer_size)
    return ds

test_ds = test_ds.map(process_path)
test_ds = prepare_for_testing(test_ds, cache="test-cached-data")
```

Next the test set is converted to a numpy array, in order to alleviate pressure from the memory. This is not necessary with such a small dataset but once again it is implemented with upgradability in mind.

```
y_test = np.zeros((n_testing_samples,))
X_test = np.zeros((n_testing_samples, 299, 299, 3))
for i, (img, label) in enumerate(test_ds.take(n_testing_samples)):
    # print(img.shape, label.shape)
    X_test[i] = img
    y_test[i] = label.numpy()
```

This next part requires the participation of the user. It was mentioned that we create checkpoints, saving the best weights. The user then is required to find the newest .h5 file that contains the best weights and type the filename into the following method.

```
m.load_weights("metastatic-vs-non-metastatic_6_rmsprop_0.000.h5")
```

Coming up are the different model evaluation methods.

```
print("Evaluating the model...")
loss, accuracy = m.evaluate(X_test, y_test, verbose=0)
print("Loss:", loss, " Accuracy:", accuracy)
```

Before resuming listing code for the evaluation, a function is created, again with upgradability in mind, which allows the user to set a threshold to impact model accuracy, based on the bias of the dataset. In our case, having a balanced dataset this is disregarded and the threshold is the default 0.5.

```
def get_predictions(threshold=None):
    y_pred = m.predict(X_test)
    if not threshold:
        threshold = 0.5
    result = np.zeros((n_testing_samples,))
    for i in range(n_testing_samples):
        if y_pred[i][0] >= threshold:
            result[i] = 1
        else:
            result[i] = 0
    return result

threshold = 0.5

y_pred = get_predictions(threshold)
accuracy_after = accuracy_score(y_test, y_pred)
print("Accuracy after setting the threshold:", accuracy_after)
```

Still in the evaluation section, the code for the confusion matrix is presented. Normalization takes place in `cmn=cmn.astype('float')/cmn.sum(axis=1)[:, np.newaxis]`.

```
def plot_confusion_matrix(y_test, y_pred):
    cmn = confusion_matrix(y_test, y_pred)

    cmn = cmn.astype('float') / cmn.sum(axis=1)[:, np.newaxis]

    print(cmn)
    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(cmn, annot=True, fmt='.2f',
                xticklabels=[f"pred_{c}" for c in class_names],
                yticklabels=[f"true_{c}" for c in class_names],
                cmap="Blues"
                )
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

plot_confusion_matrix(y_test, y_pred)
```

The resulting plot is *Figure 23* of the Results section.

Following the confusion matrix plot, the sensitivity and specificity metrics are displayed

```
sensitivity = sensitivity_score(y_test, y_pred)
specificity = specificity_score(y_test, y_pred)

print("Metastasis Sensitivity:", sensitivity)
print("Metastasis Specificity:", specificity)
```

and to visualize them the ROC curve is plotted as well via the `plot_roc_auc()` function.

```
def plot_roc_auc(y_true, y_pred):
    """
    This function plots the ROC curves and provides the scores.
    """
    plt.figure()
```

```

fpr, tpr, _ = roc_curve(y_true, y_pred)
roc_auc = auc(fpr, tpr)
print(f"ROC AUC: {roc_auc:.3f}")
plt.plot(fpr, tpr, color="blue", lw=2,
         label='ROC curve (area = {f:.2f})'.format(d=1, f=roc_auc))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curves')
plt.legend(loc="lower right")
plt.show()

plot_roc_auc(y_test, y_pred)

```

The resulting plot is *Figure 24* of the Results section.

Finally, a function that utilizes the Predict.zip, for demonstrating in a visually satisfying and easy to comprehend manner the resulting prediction of an input image.

```

def predict_image_class(img_path, model, threshold=0.5):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=(299, 299))
    img = tf.keras.preprocessing.image.img_to_array(img)
    img = tf.expand_dims(img, 0)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    img = tf.image.convert_image_dtype(img, tf.float32)
    predictions = model.predict(img)
    #print(predictions)
    score = predictions.squeeze()
    print(score)
    if score >= threshold:
        print(f"This image is {100 * score:.2f}% metastatic.")
    else:
        print(f"This image is {100 * (1 - score):.2f}% non-metastatic.")
    plt.imshow(img[0])

```

```
plt.axis('off')  
plt.show()
```

When called,

```
predict_image_class("data/Predict/2016_15934tm2_M.jpg", m)
```

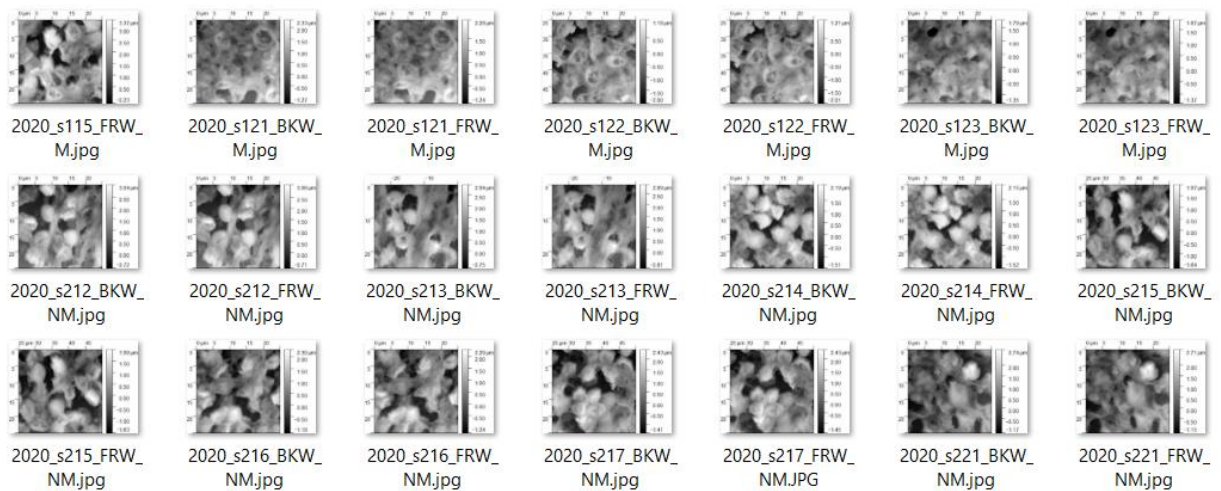
the console response is the one displayed in *Figure 25* of the Results section.

## 8.2 User Manual

Disclaimer: In its current state this application is not meant to be operated by a user with no prior experience in Python. This is a result of the fact that it was developed primarily as proof of concept and does not represent the complete vision for the project.

### 1. Preprocessing

So as was mentioned in the beginning of the appendix section, the first action to be taken by the user is preparing data-folders in an appropriate format, as well as making a selection for the K-number of the K-fold validation method. Assuming the data are collected in a folder like in *Figure 7* below:

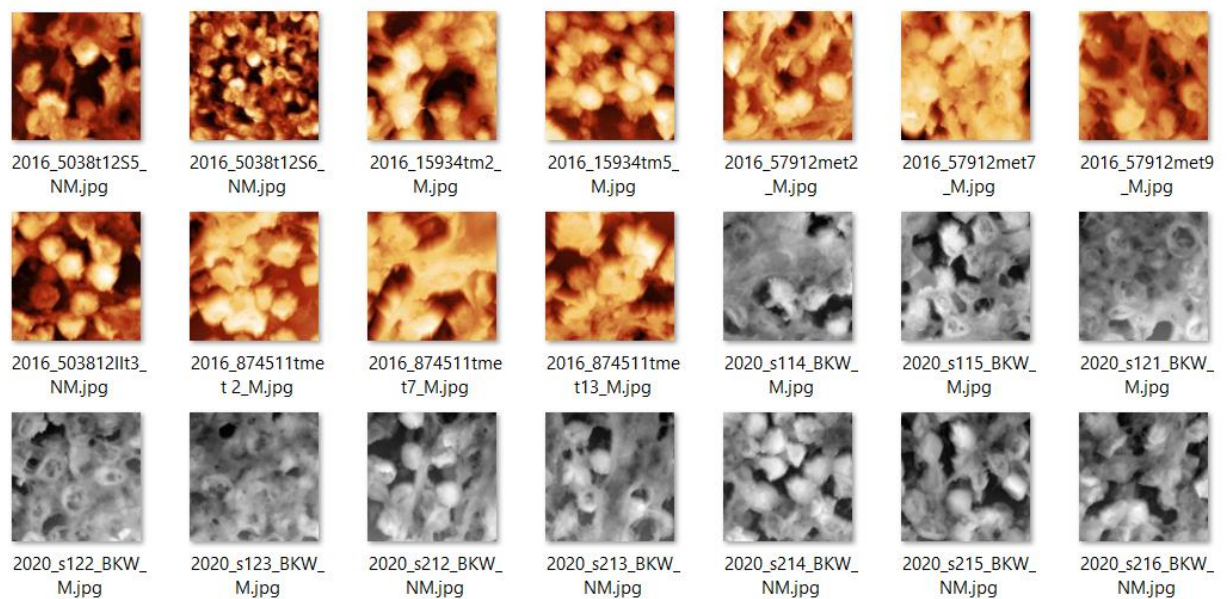


*Figure 32: Entire dataset sample*

Notice that the images are classified as metastatic or non-metastatic in the legend. This has no use in the code itself, but is rather a way for the user to be able to monitor for potential mistakes and is advised for those reasons.

With this in mind the next point of interest is determining the metastatic to non-metastatic ratio. To eliminate bias, it is advised to keep the ratio to 0.5 as much as possible, by either removing data or better yet by rotating existing images and adding them to the image pool.

After the creation of the data folder that contains the whole dataset is complete, the next step of the process is cropping any non-AFM elements from the images. You may have noticed for example that in *Figure 32* the data comes with a scale at the border of the AFM image. This border is considered noise for the scope of this program and should be removed to avoid model training on impure data. The result should resemble *Figure 33*. Any image processing application/program is equipped with the tools for image cropping, MS Paint for instance.



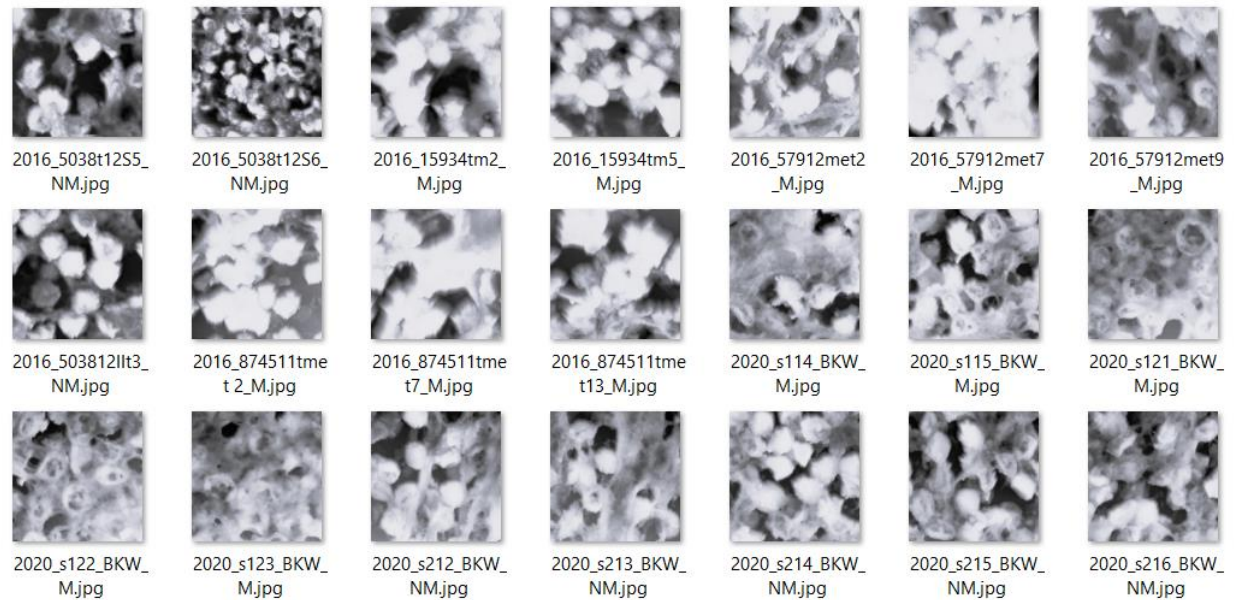
*Figure 33: Entire dataset sample cropped.*

The next step of the process is mandatory and can be summed up as image resizing. As previously mentioned, the input image size that the model accepts as input is 299x299. For this purpose, the data needs to meet this criterion. This can be accomplished manually with the use of an image processing application, though not necessary, since a resizing to the desired dimensions is taking place via the Python code automatically.

```
return tf.image.resize(img, [299, 299])
```

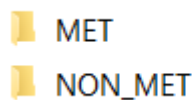
As can be observed in *Figure 33*, the data may come in either a colored or a grayscale form. To eliminate this disparity, the user needs to grayscale the entire dataset for uniformity's sake. This can once more be achieved via an appropriate image processing

tool. Be sure to apply the same grayscale settings to the entirety of the dataset. The outcome of this process should resemble *Figure 34*.



*Figure 34: Entire dataset sample cropped, resized and grayscale.*

With the dataset prepared, the selection of K determines the next steps. First create a folder named test. Inside that folder create two new folders named MET and NON\_MET respectively. The naming here is extremely important as the program relies on being precise with following these steps. Let's refer to this process of creating these folders as the MET nomenclature, *Figure 35*.



*Figure 35: MET nomenclature*

As the names imply, these folders should contain data of metastatic and non-metastatic data respectively, for testing and evaluation purposes. These test data will not be utilized



for training and are typically used either for evaluation for the model (if a legend is available) or they represent data that warrant a prediction generation by the model, so they can be classified. After the test folder is set up, make a copy of the folder which contains the grayscaled images as editing of said folder will take place. Then, from either of the identical folders, cut and paste data into the appropriate subfolders of the test folder. Note that typically the size in terms of data of the test set ranges from 10%-20% of the whole dataset. What should be the result is a master folder with the whole dataset grayscaled, the test folder with its two subfolders containing data and a third folder with grascaled imagery, identical to the master folder, apart from the imagery placed into test. Also create a Predict folder that contains the test dataset, without employing the MET nomenclature.

What needs to happen next is partitioning the remaining of the dataset, namely the data that will be used for model training, in a way that fulfills the K-fold validation method.

Based on the user's selection of K, 2xK folders need to be created, namely train1 through trainK and valid1 through validK. Inside each of these apply the MET nomenclature. The next part is a rather tedious but a necessary one. The logic is this: Let N be a random integer between 1 and K. Then the sum of the contents of trainN and validN must comprise the entire dataset. This needs to apply to every set of traini and validi folders.

Lastly, the folders that are going to be used for the execution of the program, namely test, predict, train1-trainK and valid1-validK need to be converted in the .zip format. This requires a separate piece of software, which performs the task. For instance, using WinRAR, right click on the folder and select the "Add to archive..." option, as shown in *Figure 36*.

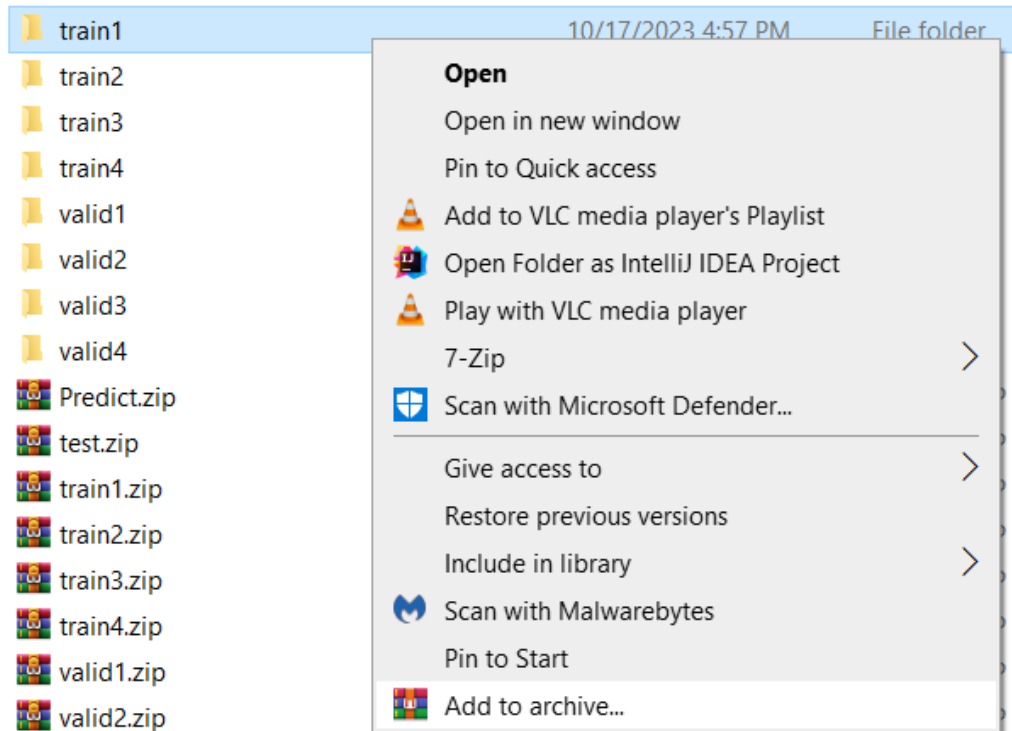


Figure 36: Add to archive

Then on the pop-up window select the .zip format and click Ok or hit the Enter key in the keyboard, *Figure 37*.

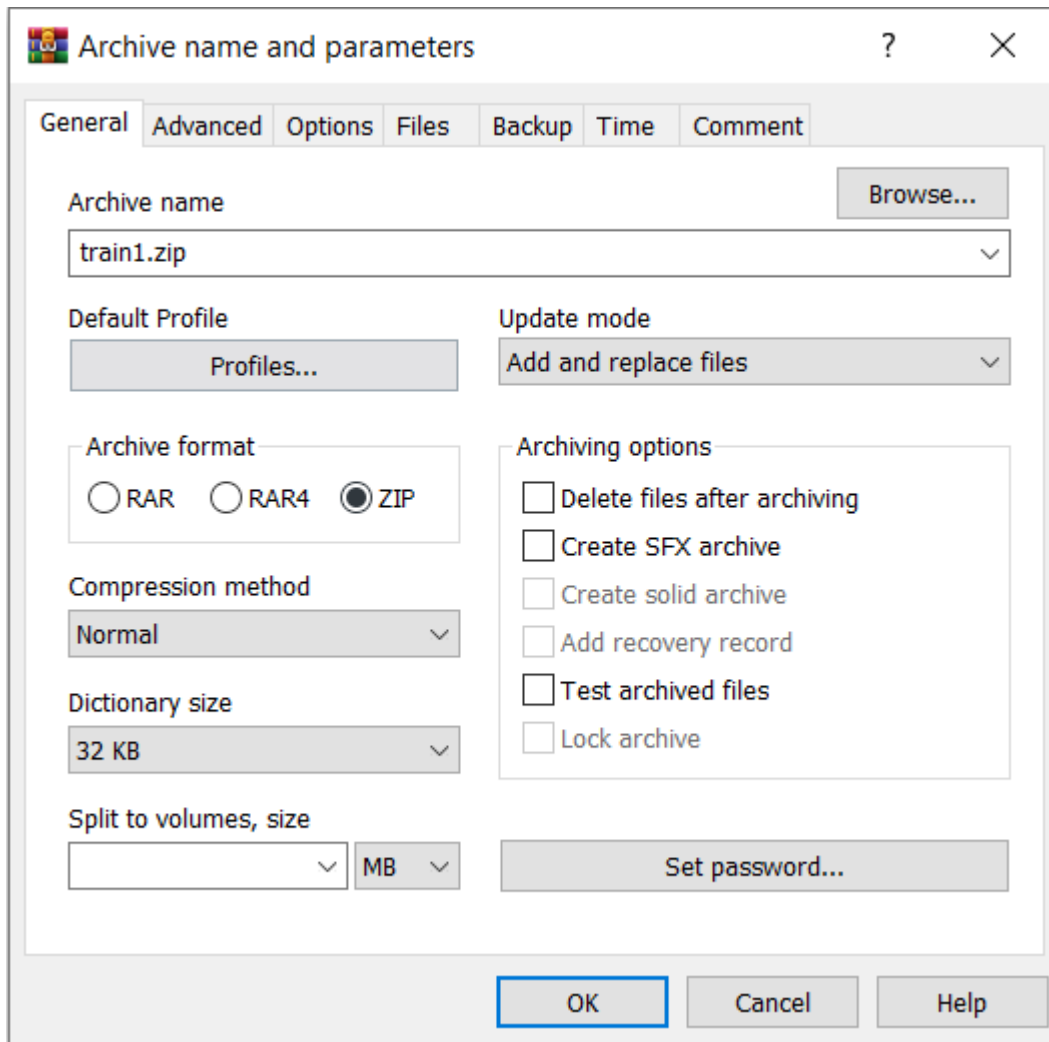


Figure 37: zip format

The steps above describe the way the user must format the data in order for the application to register the correctly and produce results.

## 2. Parameter tweaking

The selection of K affects the whole program structure so naturally some parameters need changing within the code itself. Before that though the user needs to upload the .zip files created to the Google Colab platform. Simply drag and drop the .zip files into the appropriate section as shown in *Figure 13*.

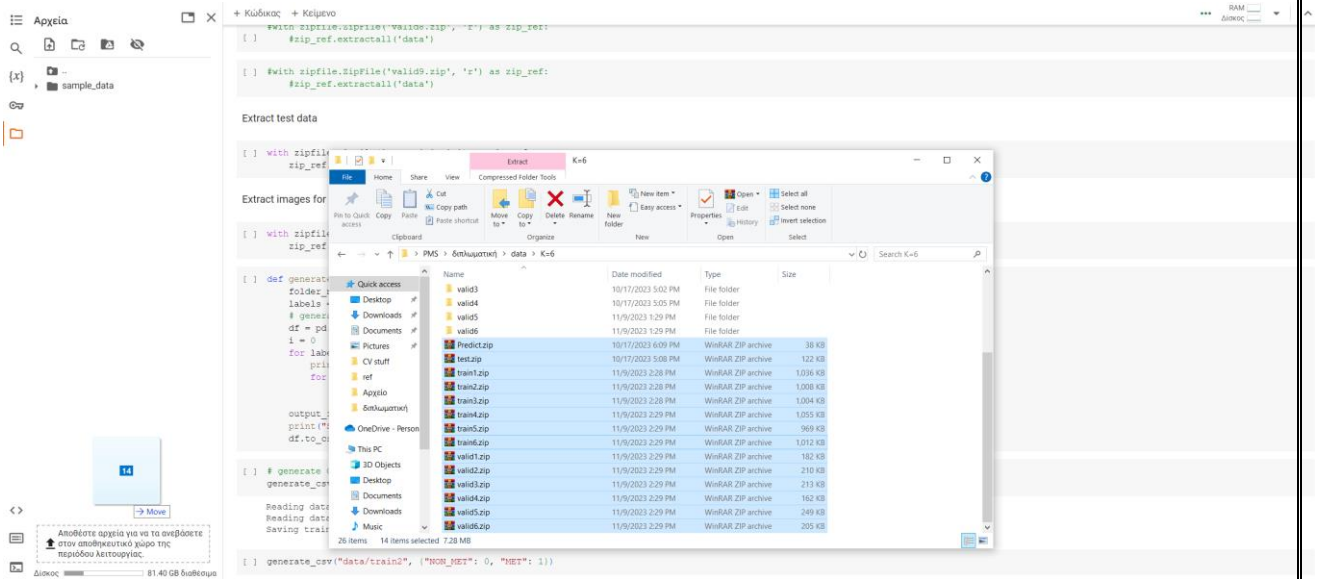


Figure 38 : Uploading the data

Again, a reminder here, this application demands from the user a relative experience with the Google Colab interface as well as the Python programming language, as they will need to edit the code itself at times.

Having said that, the selection of K as well as the batch size selection influence a number of code cells.

Starting off with the Extract train data and Extract validation data sections of the code, the user must make sure to add or remove lines of code based on the number of train1.zip and valid1.zip files, in such a way that every .zip file needed is extracted. To add a line of code select a cell and go to the Google Colab’s toolbar, navigate to the Insert option, then select Code cell. This will create an empty code cell below the selected cell. Copy and Paste the line of code shown below and be sure to change the name accordingly.

```

with zipfile.ZipFile('train1.zip', 'r') as zip_ref:
    zip_ref.extractall('data')

```

The same procedure must take place for the generate\_csv() method

```
# generate CSV files for all data portions, labeling
MET(metastatic) --> 1 and NON_MET(non_metastatic) --> 0
generate_csv("data/train1", {"NON_MET": 0, "MET": 1})
```

The same logic applies to the block of code for loading the data.

```
# loading data
train1_metadata_filename = "train1.csv"
valid1_metadata_filename = "valid1.csv"
...
trainK_metadata_filename = "trainK.csv"
validK_metadata_filename = "validK.csv"

# load CSV files as DataFrames
df_train1 = pd.read_csv(train1_metadata_filename)
df_valid1 = pd.read_csv(valid1_metadata_filename)
...
df_trainK = pd.read_csv(trainK_metadata_filename)
df_validK = pd.read_csv(validK_metadata_filename)

n_training1_samples = len(df_train1)
n_validation1_samples = len(df_valid1)
...

n_trainingK_samples = len(df_trainK)
n_validationK_samples = len(df_validK)

print("Number of training samples [1]:", n_training1_samples)
print("Number of validation samples [1]:", n_validation1_samples)
...
print("Number of training samples [K]:", n_trainingK_samples)
print("Number of validation samples [K]:", n_validationK_samples)
```

```

train1_ds =
tf.data.Dataset.from_tensor_slices((df_train1["filepath"],
df_train1["label"]))
valid1_ds =
tf.data.Dataset.from_tensor_slices((df_valid1["filepath"],
df_valid1["label"]))

...

train6_ds =
tf.data.Dataset.from_tensor_slices((df_train6["filepath"],
df_train6["label"]))
valid6_ds =
tf.data.Dataset.from_tensor_slices((df_valid6["filepath"],
df_valid6["label"]))

```

Same for the following

```

# preprocess data
def decode_img(img):
    # convert the compressed string to a 3D uint8 tensor
    img = tf.image.decode_jpeg(img, channels=3)
    # Use `convert_image_dtype` to convert to floats in the [0,1]
    range.
    img = tf.image.convert_image_dtype(img, tf.float32)
    # resize the image to the desired size.
    return tf.image.resize(img, [299, 299])

def process_path(filepath, label):
    # load the raw data from the file as a string
    img = tf.io.read_file(filepath)
    img = decode_img(img)
    return img, label

train1_ds = train1_ds.map(process_path)
valid1_ds = valid1_ds.map(process_path)

```

```

...
trainK_ds = trainK_ds.map(process_path)
validK_ds = validK_ds.map(process_path)

for image, label in train1_ds.take(1):
    print("Image shape [1]:", image.shape)
    print("Label [1]:", label.numpy())

...

for image, label in trainK_ds.take(1):
    print("Image shape [K]:", image.shape)
    print("Label [K]:", label.numpy())

```

Next up the selection batch size as well as that of K will have an effect on the following:

```

batch_size =6
optimizer = "rmsprop"

def prepare_for_training(ds, cache=True, batch_size=6,
shuffle_buffer_size=1000):
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()
    # shuffle the dataset
    ds = ds.shuffle(buffer_size=shuffle_buffer_size)
    # Repeat forever
    ds = ds.repeat()
    # split to batches
    ds = ds.batch(batch_size)
    # `prefetch` lets the dataset fetch batches in the background
    while the model is training.
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return ds

```

```

train1_ds = prepare_for_training(train1_ds, batch_size=batch_size,
cache="train1-cached-data")
valid1_ds = prepare_for_training(valid1_ds, batch_size=batch_size,
cache="valid1-cached-data")

...

trainK_ds = prepare_for_training(trainK_ds, batch_size=batch_size,
cache="trainK-cached-data")
validK_ds = prepare_for_training(validK_ds, batch_size=batch_size,
cache="validK-cached-data")

```

In this next cell, the batch size also dictates the figsize, namely the the size of the output figure in the console. It must be large enough to accommodate the whole batch. For that purpose the range(K) parameter needs to be altered as well, along with the subplot parameters, again to properly display the batch data.

```

batch1 = next(iter(valid1_ds))

...

batchK = next(iter(validK_ds))

class_names = ["non metastatic", "metastatic"]

def show_batch(batch):
    plt.figure(figsize=(10,10))
    for n in range(6):
        ax = plt.subplot(2,3,n+1)
        plt.imshow(batch[0][n])
        plt.title(class_names[batch[1][n].numpy()].title())
        plt.axis('off')

```

If the user so pleases, to display a certain batch, they should make sure to add a cell and type the following (or copy the code from a similar cell that is already provided) like so:

```
show_batch(batchi)
```



### 3. Training parameters

At this stage, the user will be familiarized with the way of interacting with the training of the model.

The selection of K also begets changes in the training stage. Firstly, the user needs to make sure that the following block of code is repeated K times.

```
history = m.fit(train1_ds, validation_data=valid1_ds,
                steps_per_epoch=3,
                #steps_per_epoch=n_training_samples // batch_size,
                validation_steps=1,
                #validation_steps=n_validation_samples // batch_size,
                verbose=1,
                epochs=100,
                callbacks=[tensorboard, modelcheckpoint])
```

The K blocks must differ from one another in that in the first line of code where train1\_ds and valid1\_ds are referenced, those need to be changed for each individual block, until in the final block they are replaced with trainK\_ds and validK\_ds.

Other than that, the training parameters that the users may insert their preference are the numbers that indicate the steps\_per\_epoch, dictating the number of training steps per epoch, the validation\_steps, dictating the number of validation steps per epoch and epochs, dictating the number of epochs of the training cycle, namely the number of times the model parses the entire dataset.

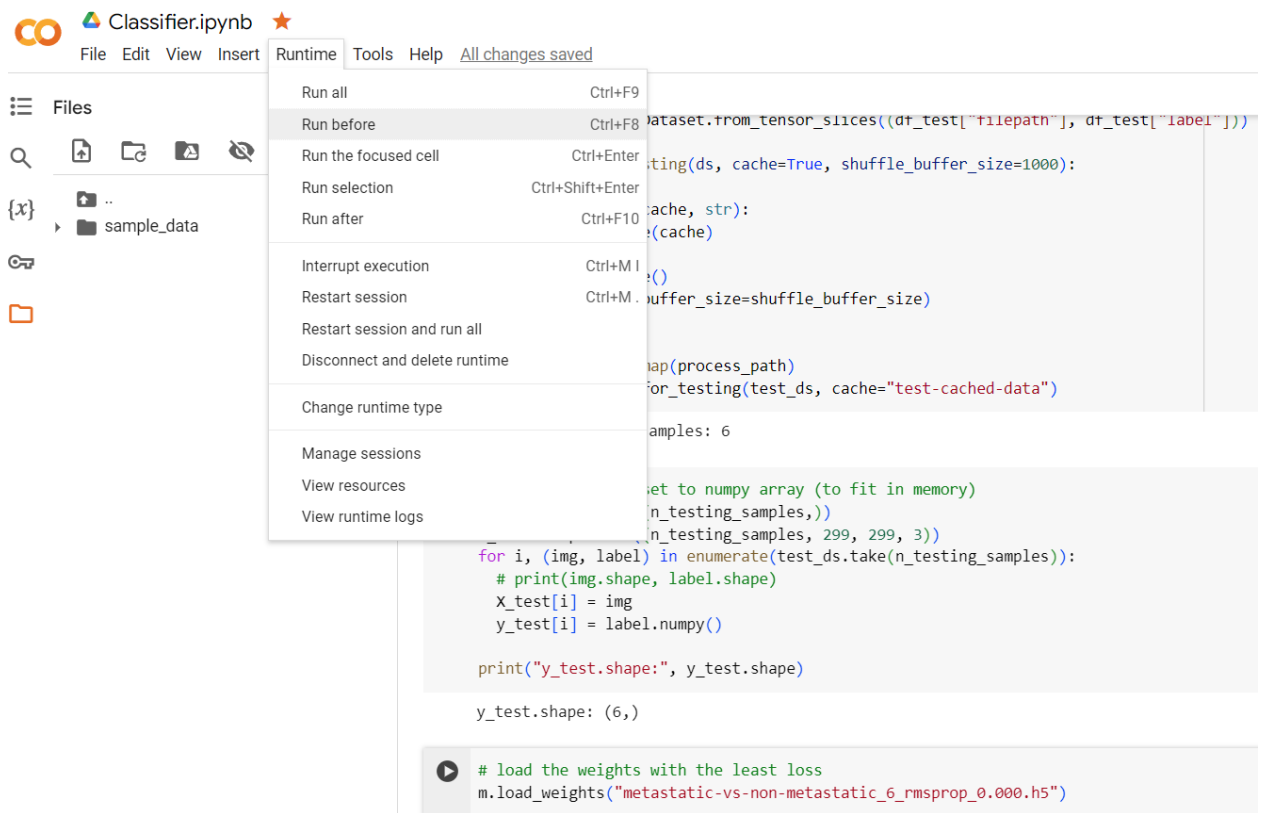
### 4. Running the application

Assuming the user followed the instructions thus far, they now must locate the line of code which mentions:

```
# load the weights with the least loss
```

After locating the cell of code where the loading of the best weights as a result of training takes place and selecting it by left clicking inside, hit Runtime in the toolbar on the top of

the interface and select Run Before. This will execute all lines of code prior to the selected one, *Figure 39*.



*Figure 39: Run Before command*

Once it's finished, in the Files section of the interface, a number of .h5 files will appear. Simply find the one created last and replace the name in the bracket of the `m.load_weights()` method. This will load the best weights for generating a prediction, calculated during training.

```
m.load_weights(\"metastatic-vs-non-metastatic_6_rmsprop_0.000.h5\")
```

Another way to accomplish the same result is to go to the training report of the final training block (details in the 3. Training parameters section) and search the training log for the name of the final .h5 file. Details in *Figure 40*.

```

▶ history = m.fit(train6_ds, validation_data=valid6_ds,
                 steps_per_epoch=3,
                 #steps_per_epoch=n_training_samples // batch_size,
                 validation_steps=1,
                 #validation_steps=n_validation_samples // batch_size,
                 verbose=1,
                 epochs=100,
                 callbacks=[tensorboard, modelcheckpoint])
3/3 [=====] - ETA: 0s - loss: 6.4898e-05 - accuracy: 1.0000
↳ Epoch 2: val_loss did not improve from 0.00006
3/3 [=====] - 4s 2s/step - loss: 6.4898e-05 - accuracy: 1.0000 - val_loss: 6.0086e-05 - val_accuracy: 1.0000
Epoch 3/100
3/3 [=====] - ETA: 0s - loss: 8.2083e-05 - accuracy: 1.0000
Epoch 3: val_loss did not improve from 0.00006
3/3 [=====] - 4s 2s/step - loss: 8.2083e-05 - accuracy: 1.0000 - val_loss: 5.9388e-05 - val_accuracy: 1.0000
Epoch 4/100
3/3 [=====] - ETA: 0s - loss: 8.3140e-05 - accuracy: 1.0000
Epoch 4: val_loss improved from 0.00006 to 0.00006, saving_model_to_metastatic-vs-non-metastatic_6_rmsprop_0.000.h5
3/3 [=====] - 6s 3s/step - loss: 8.3140e-05 - accuracy: 1.0000 - val_loss: 5.5072e-05 - val_accuracy: 1.0000
Epoch 5/100
3/3 [=====] - ETA: 0s - loss: 6.9033e-05 - accuracy: 1.0000
Epoch 5: val_loss did not improve from 0.00006
3/3 [=====] - 4s 2s/step - loss: 6.9033e-05 - accuracy: 1.0000 - val_loss: 5.9323e-05 - val_accuracy: 1.0000
Epoch 6/100
3/3 [=====] - ETA: 0s - loss: 7.5032e-05 - accuracy: 1.0000

```

Figure 40: h5 file location in the log

After that the user can resume running the rest of the application by once again going to Runtime and selecting Run after, assuming they are satisfied with the evaluation parameters that follow.

#### 4. Evaluation parameters and generating prediction

In these cells:

```

def get_predictions(threshold=None):
    y_pred = m.predict(X_test)
    if not threshold:
        threshold = 0.5
    result = np.zeros((n_testing_samples,))
    for i in range(n_testing_samples):
        if y_pred[i][0] >= threshold:
            result[i] = 1
        else:
            result[i] = 0
    return result

threshold = 0.5

y_pred = get_predictions(threshold)
accuracy_after = accuracy_score(y_test, y_pred)
print("Accuracy after setting the threshold:", accuracy_after)

```

and

```
# a function given a function, it predicts the class of the image
def predict_image_class(img_path, model, threshold=0.5):
    img = tf.keras.preprocessing.image.load_img(img_path,
target_size=(299, 299))
    img = tf.keras.preprocessing.image.img_to_array(img)
    img = tf.expand_dims(img, 0) # Create a batch
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    img = tf.image.convert_image_dtype(img, tf.float32)
    predictions = model.predict(img)
    #print(predictions)
    score = predictions.squeeze()
    print(score)
    if score >= threshold:
        print(f"This image is {100 * score:.2f}% metastatic.")
    else:
        print(f"This image is {100 * (1 - score):.2f}% non-metastatic.")
    plt.imshow(img[0])
    plt.axis('off')
    plt.show()
```

the user may edit the value of threshold. The threshold value is implemented in the cases that the dataset is not a balanced one and moreover in cases that come with innate bias. For example: Let's assume that most images are metastatic and furthermore assume that 60% of them are classified as such. As a result of that observation, setting the threshold value to 0.6 will yield more accurate results.

In the following block of code, a function is called that generates a prediction on a single image. The user needs to find the directory that the images is located along with the image name and substitute as a parameter value inside the predict\_image\_class() method.

```
predict_image_class("data/Predict/2016_15934tm2_M.jpg", m)
```

Important note: In most cases of running the code multiple times errors will occur. It is advised to go to Runtime and select Disconnect and delete runtime, re-upload the .zip datafiles and follow the instructions once more (Figure 41).

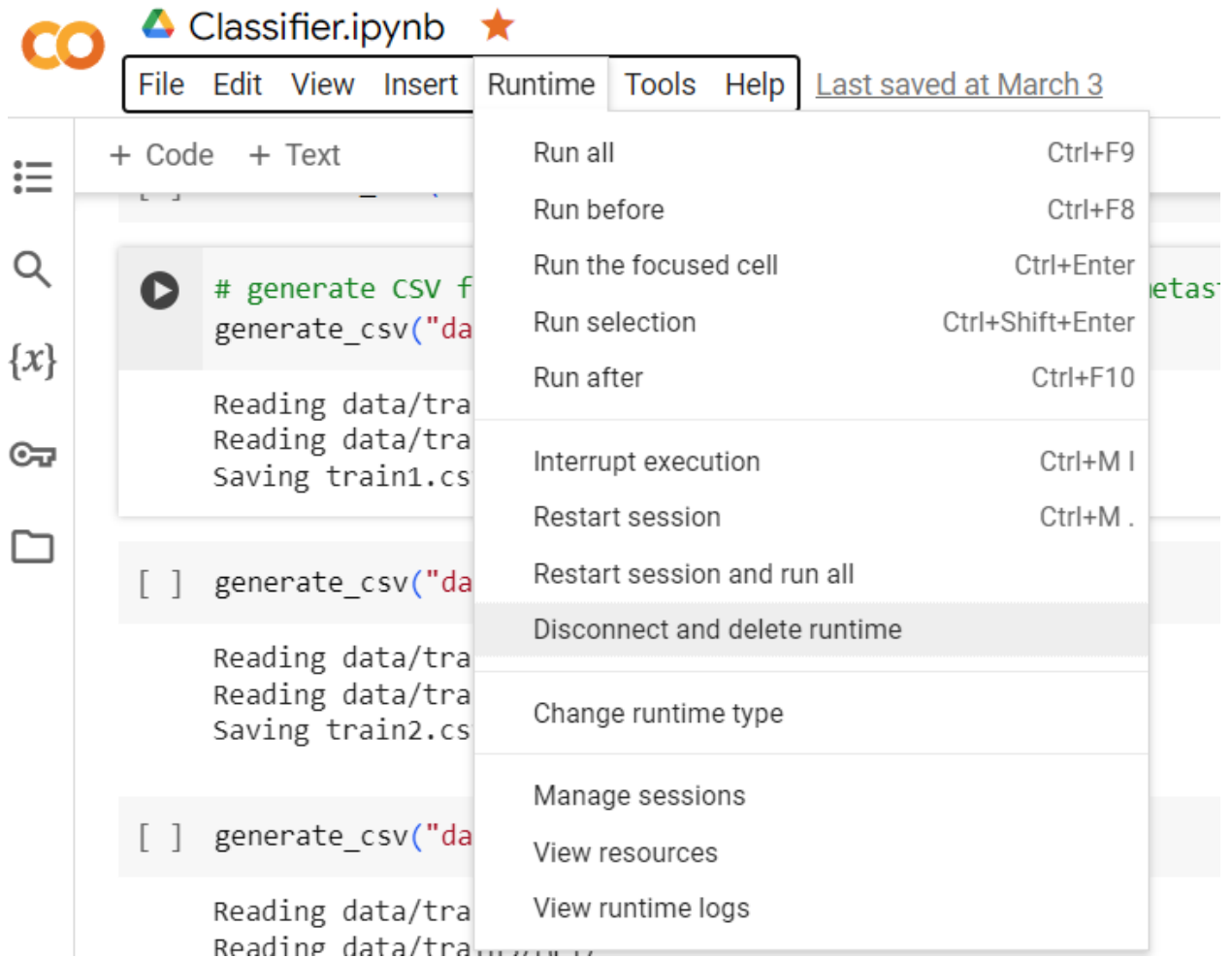


Figure 41: Disconnecting and deleting runtime