UNIVERSIDAD POLITÉCNICA DE MADRID

ETSI y Sistemas de Telecomunicación y

ETSI y Sistemas Informáticos

Máster Universitario en Internet of Things

MASTER'S THESIS

"CONVOLUTIONAL NEURAL NETWORK DEPLOYMENT USING AZURE MACHINE LEARNING FOR INTRAOPERATIVE BRAIN TUMOR CLASSIFICATION"

ALBERTO MARTÍN PÉREZ

JULY 2021

Máster Universitario en Internet of Things
Master in Internet of Things

Master's Thesis		
Title	Convolutional Neural Networks Deployment Using Azure Machine Learning For Intraoperative Brain Tumor Classification	
Author	Alberto Martín Pérez	Signature
Tutor / Co-tutor	Eduardo Juárez Martínez	Signature
Director		Signature
Board of examiners		
Presidente/ President	José Fernán Martínez	
Secretario/ Secretary	Iván Pau de la Cruz	
Vocal	Rubén de Diego Martínez	
Date	July 16, 2021	
Grade		

Secretary

Abstract

Hyperspectral imaging (HSI) approaches have been widely used in the medical field for diagnosis purposes. They can provide a useful non-ionizing and non-invasive tool which does not require any contact with patients. When combining HSI with machine learning (ML) algorithms such as convolutional neural networks (CNN), in-vivo classifications can be done with trained CNN models to reliably discern between healthy and tumor tissue during brain tumor surgery. Although some research projects such as NEMESIS-3D-CM have been able to classify brain tumors in a single operating room, with this thesis the aim is to provide a brain classification service to multiple hospitals at the same time. For that purpose, PyTorch CNN models have been trained in Microsoft Azure using the Azure Machine Learning service and the Azure software development kit (SDK) for Python. Experiments have shown that at least 3 and a half hours have been needed to train CNN models with 12 hyperspectral images using double cross-validation techniques. Although not using double cross-validation when training under the same conditions have taken 13 minutes, which is almost 16 times less, overall accuracy ($OACC$) and sensitivity (SEN) classification metrics have not provided sufficient evidence to omit cross-validation during training. Besides, deploying registered CNN models trained with Azure Machine Learning have taken around 47 seconds, no matter if cross-validation techniques were used during training. These models have been deployed to containers using Azure Kubernetes Service (AKS), which when they have been used to classify raw hyperspectral brain images, they have lasted around 27 seconds to return a classification map. Nonetheless, faster responses could be provided if data preparation tasks became more optimized before prediction.

Resumen

Las imágenes hiperespectrales se han utilizado ampliamente en el campo de la medicina con fines de diagnóstico. Han demonstrado poder proporcionar herramientas útiles al ser una tecnología no ionizante y no invasiva, no requieriendo contacto alguno con los pacientes. Cuando se combina con algoritmos de aprendizaje automático, como las redes neuronales convolucionales, se pueden realizar clasificaciones in vivo con modelos entrenados para discernir, de forma fiable, entre el tejido sano y el tumoral durante cirugías de tumores cerebrales. Aunque algunos proyectos de investigación como NEMESIS-3D-CM han sido capaces de clasificar tumores cerebrales en un solo quirófano, con esta tesis se pretende proporcionar un servicio de clasificación cerebral a múltiples hospitales al mismo tiempo. Para ello se han entrenado modelos de redes neuronales convolucionales con PyTorch en Microsoft Azure, utilizando el servicio Azure Machine Learning y el kit de desarrollo de software de Azure para Python. Los experimentos han demostrado que se han necesitado al menos 3 horas y media para entrenar modelos con 12 imágenes hiperespectrales utilizando técnicas de doble validación cruzada. Aunque no utilizar la doble validación cruzada al entrenar en las mismas condiciones ha llevado 13 minutos, lo que supone casi 16 veces menos, las métricas de clasificación de precisión global ($OACC$) y sensibilidad (SEN) no han proporcionado pruebas suficientes para omitir la validación cruzada durante el entrenamiento. Además, el despliegue de los modelos registrados y entrenados con Azure Machine Learning ha tardado alrededor de 47 segundos, sin importar si se utilizaron técnicas de validación cruzada durante el entrenamiento. Estos modelos se han desplegado en contenedores usando el servicio de Azure Kubernetes, que cuando se han utilizado para clasificar imágenes cerebrales hiperespectrales en bruto, han tardado 27 segundos en devolver un mapa de clasificación. No obstante, se podrían proporcionar respuestas más rápidas si se optimizasen las tareas de preparación de los datos antes de la predicción.

Contents

Abbreviations

5-ALA	**5-a**mino**l**evunilic **a**cid
ACI	**A**zure **C**ontainer **I**nstances
AKS	**A**zure **K**ubernetes **S**ervice
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
DNN	**D**eep **N**eural **N**etwork
FN	**F**alse **N**egative
FP	**F**alse **P**osivite
GBM	**G**lio**B**lastoma **M**ultiforme
GIV	**G**rade **IV** (glioma)
GPU	**G**raphical **P**rocessing **U**nit
HSI	**H**yper**s**pectral **I**maging
HTTP	**H**ypertext **T**ransfer **P**rotocol
HU12O	**H**ospital **U**niversitario **12** (de) **O**ctubre
iMRI	**I**ntraoperative **M**agnetic **R**esonance **I**maging
IoT	**I**nternet **o**f **T**ings
IOUS	**I**ntraoperative **U**ltrasounds
JSON	**J**ava**S**cript **O**bject **N**otation
ML	**M**achine **L**earning
NN	**N**eural **N**etwork
OACC	**O**verall **A**ccuracy
ReLU	**R**ectified **L**inear **U**nit
RNN	**R**ecurrent **N**eural **N**etwork
SDK	**S**oftware **D**evelopment **K**it
SEN	**S**ensitivity

SEOM	**S**ociedad **E**spañola (de) **O**ncología **M**édica
SNN	**S**andard **N**eural **N**etwork
TN	**T**rue **N**egative
TP	**T**rue **P**osivite
UPM	**U**niversidad **P**olitécnica (de) **M**adrid

Thanks to Gonzalo Rosa, Guillermo Vázquez and Manuel Villa for their continuous support during the Master degree. It has been an enormous pleasure to have met you, learned from you, and spent time all together for this entire year. In all honesty, we have formed such a great group of friends and I think we will contribute lots to the GDEM research group in our following PhD. years.

Chapter 1

Introduction

Cancer is the result of normal cells turning into malign cells. These cells start growing severely, killing any healthy tissue encountered and creating masses, commonly known as tumors. Spanish Society of Medical Oncology (SEOM) annually gathers data regarding the prevalence, mortality, incidence, and survival of cancer [1]. They indicate in their article that cancer has caused almost 9.6 million deaths globally in 2018, making cancer one of the most common causes of death. Every year more cases appear and they have estimated that by the year 2040, the incidence could reach 29.5 million new patients suffering from cancer globally. Regarding brain tumors, glioblastoma multiforme (GBM) or grade 4 glioma (GIV) is the most aggressive [2], reducing the survival rate of patients.

Unfortunately, treating GBM unavoidably involves going through surgery in the operating room. Although these surgical interventions can increase the patient's life expectancy, it is crucial to resect as much malign tissue while removing as little healthy tissue as possible. However, the main problem with GBM surgery comes from its high infiltration ability, which makes the distinction of tumor and healthy tissue boundaries highly challenging [3]. Even though technologies such as neuronavigation tools [4], intraoperative magnetic resonance imaging (iMRI) [5], intraoperative ultrasounds (IOUS) [6], and the use of markers such as 5-aminolevunilic acid (5-ALA) [7] are frequently used by neurosurgeons, they are either invasive or greatly time consuming. An alternative tool could be hyperspectral imaging, which is convenient for medical use since it is non-invasive, non-ionizing, and does not require any contact with the patient. Furthermore, it has been proved its ability to distinguish between in vivo healthy and tumor tissues when used with ML algorithms [8]. HSI gathers the diffuse reflectance data from the measured spectrum, which can start from the near ultraviolet to the near infrared, including the visible spectrum. HSI information is stored in three-dimensional datasets containing spatial and spectral information from the captured scene. These datasets are

often called hyperspectral cubes, providing what is known as spectral signatures. These signatures incorporate the diffuse reflectance for every captured wavelength, helping to distinguish different materials or tissues.

This thesis is included within the NEMESIS-3D-CM research project, collaborating Fundación para la investigación biomédica del Hospital universitario 12 de Octubre (HU12O) and Universidad Politécnica de Madrid (UPM). The project aims to provide real-time diagnosis tools to neurosurgeons to help them during brain tumor interventions, using as the main technology HSI. Therefore, all hyperspectral brain images to train ML models for this thesis have been provided by NEMESIS-3D-CM.

Moreover, an Internet of Things (IoT) approach for NEMESIS-3D-CM is presented in Figure 1.1. One goal of it would be to provide a classification tool service to multiple hospitals whenever it is necessary. This way, a trained ML model could be distributed and used in many operating rooms at the same time to classify brain images. ML models would be trained in the cloud using already captured hyperspectral brain images. They would be gathered using a hyperspectral imaging acquisition system, with the help of additional tools like a neuro-navigator, to then be adapted. These images would be labeled by neurosurgeons after surgery, when a pathological analysis of brain samples has been done. This is shown in Figure 1.1, where the slow feedback is provided by neurosurgeons after at least one week. Afterwards, the images and their labels would be uploaded to the cloud where the training occurs. Nonetheless, this IoT approach also considers a fast feedback, which would be used whenever a model needs to be trained again using a few samples from the patient being operated. Either way, this fast feedback procedure has not been studied in this thesis.

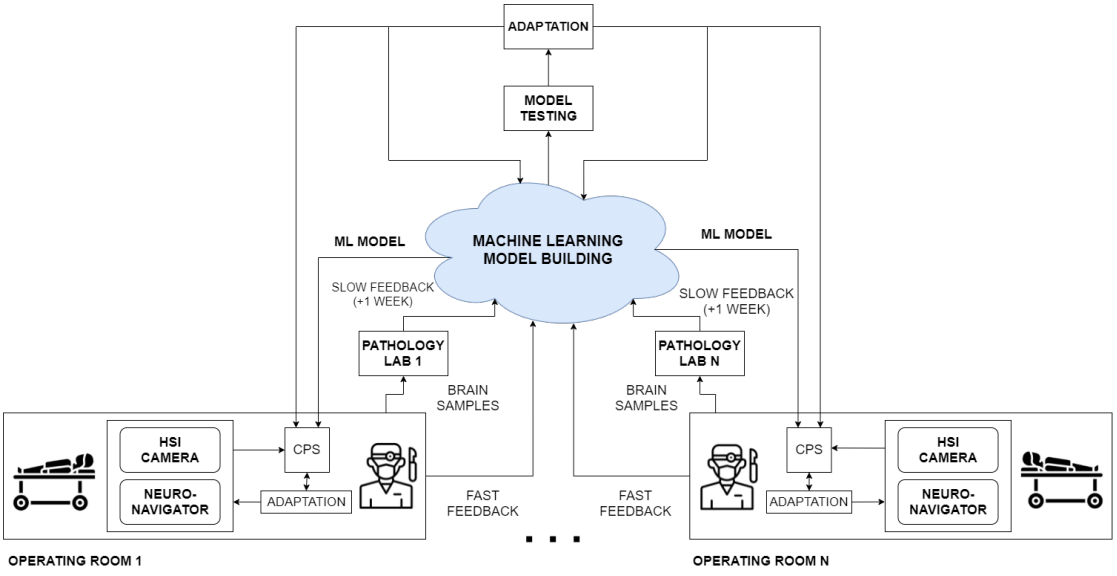

FIGURE 1.1: Internet of Things schema adaptation for the NEMESIS-3D-CM research project.

Therefore, this final master thesis aims to evaluate the viability of training ML models in the cloud and deploy them to provide the HSI classification service to multiple hospitals. For this purpose, Microsoft Azure platform has been used as the cloud provider for the development of the experiments. This is due to the partnership between Microsoft and UPM. Although many ML algorithms can be used for classifying brain tumors, CNN has been the chosen option since they have demonstrated their ability to differentiate brain tissues [8].

In the following chapters, explanations will be given regarding how the implementation has been done. Besides, the thesis will start with the state of the art in Chapter 2, where an overview of the convolutional neural networks theory is given as well as a description of hyperspectral imaging and all Microsoft Azure services and tools used. Afterwards, in Chapter 3 a presentation of the NEMESIS-3D-CM images is provided, how they have been preprocessed as well as how they have been managed for training CNN models. Then the evaluation and test methodology are presented alone with the CNN architecture utilized during this thesis. Chapter 4 describes how Azure Machine Learning has been used and all resources utilized to train and deploy CNN models. Later, in Chapter 5, all conducted experiments are described followed by their corresponding discussion. Finally, in Chapter 6 the estimated budget for this thesis is provided before ending with the conclusions in Chapter 7.

Chapter 2

State of the art

In this chapter, a brief description of the convolutional neural networks theory is given along with some references for regularization and optimization. Then, a presentation is provided regarding hyperspectral imaging and its use in various fields including the medical one. Finally, detailed information is given regarding Microsoft Azure, Microsoft Azure Machine Learning, and Azure SKD for Python, including a brief description on how some of their services utilize Docker and Kubernetes for deployment.

2.1 Convolutional neural networks

CNN are supervised machine learning algorithms, meaning that they require labeled data during training. Neural networks (NN) can be used in different areas where each has its specific NN structure. Some areas are advertising, audio, or image, where standard neural network (SNN), recurrent neural network (RNN) and convolutional neural network are used respectively. CNNs are often used for medical imaging analysis. They consists of three layer blocks, input layer, hidden layers, and output layer. Between the input and output layers, multiple hidden layers can be used to create what is often referred as deep neural network (DNN). Deep within this context means a large number of hidden layers.

2.1.1 Forward and backward propagation

Any neural network does have two important steps during training, the forward and backward propagation. During forward propagation, the CNN computes the outputs of every activation function in all layers for the given input, storing in cache the weights and bias parameters. Once a prediction has been made in the output layer, then a cost

function is used to determine how well the model is predicting the outputs in comparison with the labelled data. The idea is to have a low error during prediction by searching those weights and bias parameters that minimize the cost function. During the backward propagation, this search is made with the gradient descent technique, which takes small steps using the computed cache during forward propagation to update the weights and bias parameters on each layer. It computes the derivatives of each layer parameter to detect where the slope is negative and reach the minimum of the cost function. Those derivatives are multiplied by a user-defined constant called learning rate, which indicates how large the steps will be.

2.1.2 Convolution, pooling and fully connected layers

Figure 2.1, which is taken from [9], shows a basic CNN architecture, indicating that convolutional layers are used as feature extraction, while fully connected layers are used for classification tasks.

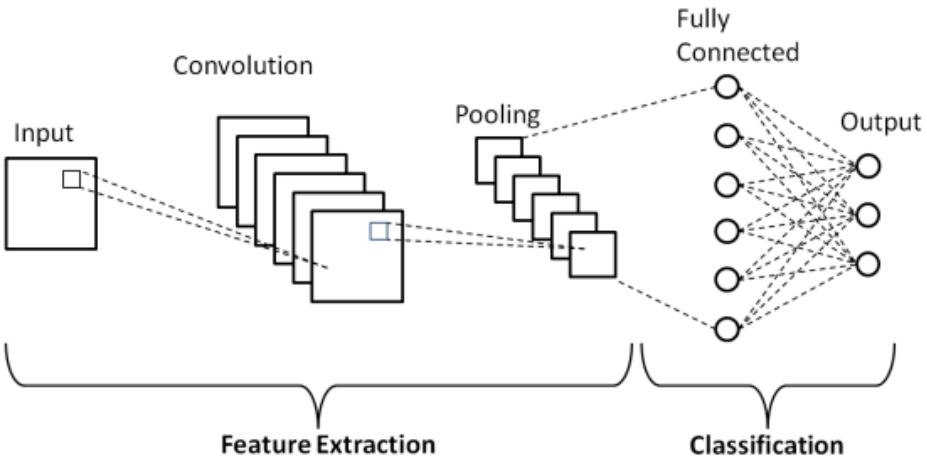

FIGURE 2.1: Schematic diagram of a basic convolutional neural network (CNN) architecture.

To provide further explanation, a description of convolution, pooling and fully connected layers is now provided:

- Convolutional layers [10] are used as high-level feature detectors such as vertical and horizontal edge detectors. One of their properties is parameter sharing, since detecting features in one part of the image is probably useful in another part of the image. To detect any feature, convolutional layers need to update the learning parameters of the image during the backpropagation step. To avoid the output matrix of a convolutional layer to shrink, it is widespread in CNN to apply the padding technique to the inputs, otherwise it would not utilize the information

contained in the edges of the images. Strided convolutions are also used to move the convolutional filters over the input image to a different number than 1. This can be done with lesser computations while losing the minimum amount of information from the input image. Both padding and stride are additional hyperparameters of the convolution operation. Furthermore, activation functions [11] are used in deep learning to define the output of hidden layers and the output units. The rectified linear unit (ReLU) [12]: $ReLU(x) = max(0, x)$ activation function is widespread and is used to compute more complex aspects of the neural network as well as speed computation, since derivations from linear slopes are quicker to calculate.

- Pooling layers [13], on the other hand, do not have learning parameters to be updated during backpropagation. What these layers do is reduce the size of the input representation to speed computation as well as detecting low-level or sharp features such as edges or points. The most common pooling layer is the max pooling, which is a down-sampling method to extract the maximum value when a filter is used on each location of the image. Another pooling layer which is not very often used nowadays is the average pooling, which is useful for extracting smooth features.

- Fully connected layers [14] are used to classify the image and provide and output with the most probable class. Once all features have been extracted through the convolutional layers, the output dimension is flattened to a one-dimensional vector to become the input for the fully connected layer. To provide classification in multiclass problems, the last layer uses a softmax activation function to give probabilities to each class available in the input image.

2.1.3 Regularization and optimization algorithms

In some cases, trained CNN models may overfit the training data. To prevent overfitting or to reduce the errors in the CNN, regularization techniques are often used. If suffering from high variance problems, some techniques such as L2-regularization [15] and dropout [16] are used. However, if the CNN model suffers from high variance problems, it is often recommended to get more reliable training data and use some techniques such as data augmentation [17] or early stopping [18].

Training CNN on large data sets is very slow. One way to speed up the training is by having fast optimization algorithms. Some of them are mini-batch gradient descent [19], exponentially weighted averages [20], gradient descent with momentum [21], RMSprop [22], and Adam optimization algorithm [23].

2.1.4 PyTorch

PyTorch is an open source machine learning library developed by Facebook's AI Research lab to work with Python programming language [24]. It provides two high level features: Tensor computing with strong graphics processing unit (GPU) acceleration support, and building DNN with a tape-based automatic differentiation system. This system stores the performed operations to properly compute the gradients during backward propagation, which helps saving time when computing the differentiation of the network parameters. Although there are many options available to train DNN like TensorFlow [25] or Keras [26], PyTorch offers a simple interface to operate with, it is well integrated with Python, and provides a platform to work with dynamic computational graphs. Therefore, due to PyTorch simplicity and Python integration, it was the chosen library to develop this thesis.

2.2 Hyperspectral imaging

2.2.1 What is a hyperspectral image?

While RGB images only have three discrete wavelengths in the visible spectrum, red, green, and blue hyperspectral images can gather spectral information including the visible and some of the infrared spectrum. Therefore, the main differences between RGB and hyperspectral images are the number of spectral bands gathered and the spectrum from which they gather data. As shown in Figure 2.2 (a), it can be seen monocrohome, RGB, multispectral and hyperspectral images. Monochrome only contains spatial dimensions from a single channel. RGB, as already mentioned, only has three discrete wavelengths in the visible spectrum. Multispectral images are very similar to hyperspectral images, being the only difference the reduced number of wavelengths gathered. And hyperspectral images, also known as hyperspectral cubes, include the spatial dimension represented by the height and width of the image and the depth dimension indicating all wavelengths gathered. Figure 2.2 (b) also shows the spectral information that can be extracted from both the hyperspectral cube and a RGB image, where the continuous graph of the hyperspectral image is known as the spectral signature of the pixel. With these spectral signatures, they can be used to differentiate materials in a single hyperspectral image such as healthy brain and tumor tissue.

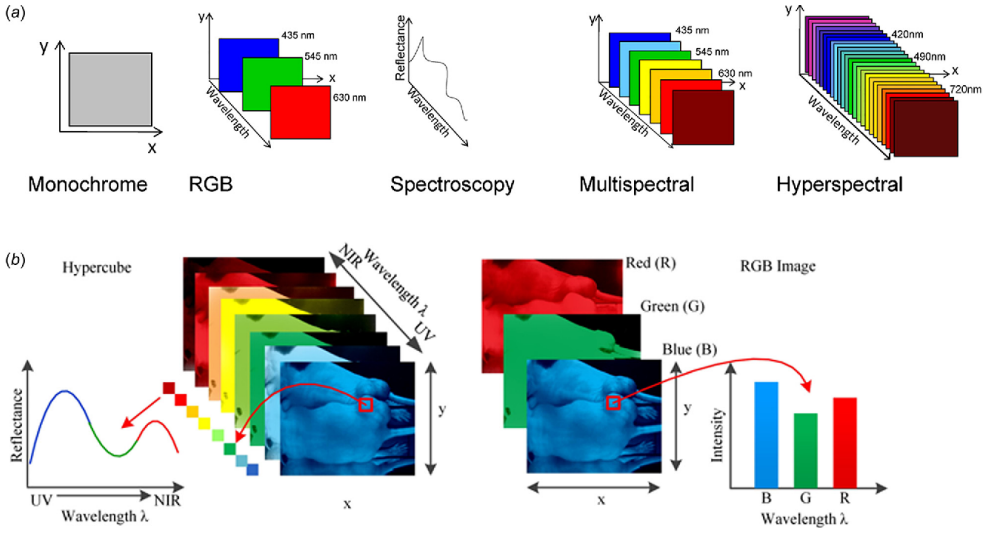

FIGURE 2.2: (a) Schematic showing the features of monochrome, red-green-blue (RGB), spectroscopy, multispectral, and HSI. (b) Detailed comparison showing the differences between HSI and RGB imaging. Source: 'Single-Cell Analysis Using Hyperspectral Imaging Modalities.' [27].

2.2.2 Hyperspectral imaging for medical diagnosis

Hyperspectral imaging is often used in the medical field since it does not require contact with the patient, it is non-ionizing as well as noninvasive. HSI collects information from a wider spectrum than the visible one (380-750 nm) [28], gathering more data than any conventional RGB image, which only captures 3 discrete wavelengths. Although HSI has been utilized for many years in remote sensing applications [29], drug analysis [30], inspection of food quality [31] and defense [32], it is now commonly used in the medical field for cancer detection and diagnosis of different kinds of diseases. Some cancers HSI has been able to detect are breast [33], skin [34], prostate [35] or tongue [36]. However, the main issue with brain cancer is how it infiltrates within the tissue, making it a difficult task to properly delineate the region of the tumor with the naked eye. While in other kinds of tumors an ample block of healthy tissue is resected, this is not possible in the brain. Therefore, it is essential to delimit proper brain tumors to reduce the extraction of healthy tissue [37]. A work done by Fabelo et al. [38] utilized supervised machine learning algorithms on HS data to provide an intraoperative visualization. With this tool, they provide an aid in brain tumor resection to help neurosurgeons be more precise and reduce possible damage to healthy tissue.

2.3 Microsoft Azure, Azure Machine Learning and Azure SDK for Python

Microsoft Azure [39] is a cloud computing service that allows users to build, test, deploy, and manage applications and services. It strives to interoperate with data servers, being hybrid clouds one of its strengths. Microsoft Azure shares 18% of the worldwide cloud market as mentioned in the Synergy Research Group report in 2020 [40]. In relation to computation services, Azure provides their Virtual Machine service that supports Linux, Windows Server, SQL Server, Oracle, IBM and SAP. It also has a large catalog of instances that can be utilized, such as machine learning instances. Azure has its own container service known as Azure Container Service, which is based on Kubernetes and uses some Azure instances like Docker Hub and Azure Container Registry. For basic storage services, Azure has its own service called Blob Storage, which is used for REST-based object storage of unstructured data. More services like Queue Storage, File Storage and Disk Storage are available within the platform. For databases, Azure has three SQL-based options which are SQL Database, Database for MySQL, and Database for PostgreSQL while offering Backup services.

Azure Machine Learning [41] is one of the services provided by Microsoft Azure to build, train, and track machine learning and deep learning models. This platform is used when a developer wants to scale a trained ML model using the power of cloud scale computing. Additionally, it provides full support to open source libraries like PyTorch, TensorFlow, or scikit learn as well as giving useful tools to let developers scale up their models. These tools can be used when an Azure Machine Learning workspace has been created. Model training, model packaging, model validation, model deployment, and model monitoring are some of the tools that Azure Machine Learning service provides to developers. One way to utilize these tools through a Python environment is by using the Azure SDK for Python [42], which can manage cloud resources, train ML models, and even deploy web services so that trained models can be consumed and used. These web services contain the packaged model and additional files needed to host it as a web service inside a Docker [43] image. This Docker image can be stored inside Azure Container Instances (ACI) [44] or AKS [45], which allow developers to run their apps in containers without worrying about how to manage virtual machines or server infrastructures.

Chapter 3

Convolutional Neural Networks and Hyperspectral Imaging

3.1 Hyperspectral imaging dataset and data management

Hyperspectral data is a high dimensional data which requires special management to train Deep Learning models. Using the entire hyperspectral images to train CNN models would slow down the training process. Rather, creating batches of data is more convenient to speed up training. In this section, a description of the hyperspectral data used is provided as well as how it has been managed to train CNN models.

3.1.1 NEMESIS-3D-CM hyperspectral images and spectral signatures

All thirteen hyperspectral images used for this thesis have been provided by the NEMESIS-3D-CM project. Ground truth labeled pixels can be seen in Table 3.1 while false RGB images and ground truth maps are presented in Figure 3.1. As Urbanos et al. mentioned in their 'Supervised Machine Learning Methods and Hyperspectral Imaging Techniques Jointly Applied for Brain Cancer Classification' published paper [8]: *'They were captured from the brain surface during in vivo neurosurgery procedures, after the completion of the craniotomy and in some cases after resection of brain tissue, so that the tumor was exposed. The 13 images corresponding to high-grade gliomas (GIII and GBM) used are: ID0018C09, ID0025C02, ID0029C02, ID0030C02, ID0033C02, ID0034C02, ID0035C02, ID0038C02, ID0047C02, ID0047C08, ID0050C02, ID0051C05 and ID0056C02. These patients have been anonymized with a unique reference number in the following format: ID patient number - C capture number. For simplicity, in the rest of the document, these captures will be identified as ID18, ID25, ID29, ID30, ID33,*

TABLE 3.1: Ground truth maps provided by the NEMESIS-3D-CM project with their number of total labeled pixels. Source: 'Supervised Machine Learning Methods and Hyperspectral Imaging Techniques Jointly Applied for Brain Cancer Classification' [8]

Patient	**Healthy T.**	**Tumor**	**Venous B.**	**Arterial B.**	**Dura M.**	**Total**
ID18	648	1587	79	14	369	2697
ID25	801	206	90	15	94	1206
ID29	3752	64	98	11	1599	5524
ID30	2587	2737	487	381	1366	7558
ID33	6671	973	842	35	1443	9964
ID34	1186	1464	181	176	780	3787
ID35	3864	1389	837	58	586	6734
ID38	1740	487	113	191	825	3356
ID47C1	174	160	58	54	567	1013
ID47C2	715	182	14	148	129	1188
ID50	410	1282	712	47	628	3079
ID51	3888	2731	95	85	525	7324
ID56	1920	1967	75	37	5922	9921

ID34, ID35, ID38, ID47C1,ID47C2, ID50, ID51 and ID56, respectively. Regarding the ID47 patient, there are two existing captures: the first one taken after duroctomy and the second one taken after a partial brain resection. In both images, the tumor is exposed in different brain levels and therefore they are considered as two cases. (...) the synthetic RGB created from a hyperspectral cube band and its corresponding GT map of all patients included in this work. In the GT maps, the tumor is represented in red color, venous and arterial blood vessels in deep blue, healthy tissue in green, dura mater in pink color, and bone in light blue. The bone class will not be taken into account in this study. All these maps are generated by the neurosurgeon who operated on the patient labeled.'

During the work performed by Urbanos et al. [8], an analysis regarding the spectral signatures for all thirteen images was done among all 5 brain tissues. Figure 3.2 shows the mean spectral signatures between the 659-951 nm spectrum and the normalized reflectance of every labeled pixel in the ground truth maps presented in Table 3.1 and Figure 3.1. Figure 3.2 displays all 5 analyzed brain tissues in different colors. Green is healthy tissue, red is tumor, gray is venous blood, cyan is arterial blood, and purple is dura mater. Blood spectral signatures are similar along the spectrum, while the dura mater tissue had the most different spectral signature among all classes. However, it shows that healthy and tumor spectral signatures are very similar in the observed spectrum. This has influenced the ability of trained CNN models to differentiate healthy tissue from tumor cells.

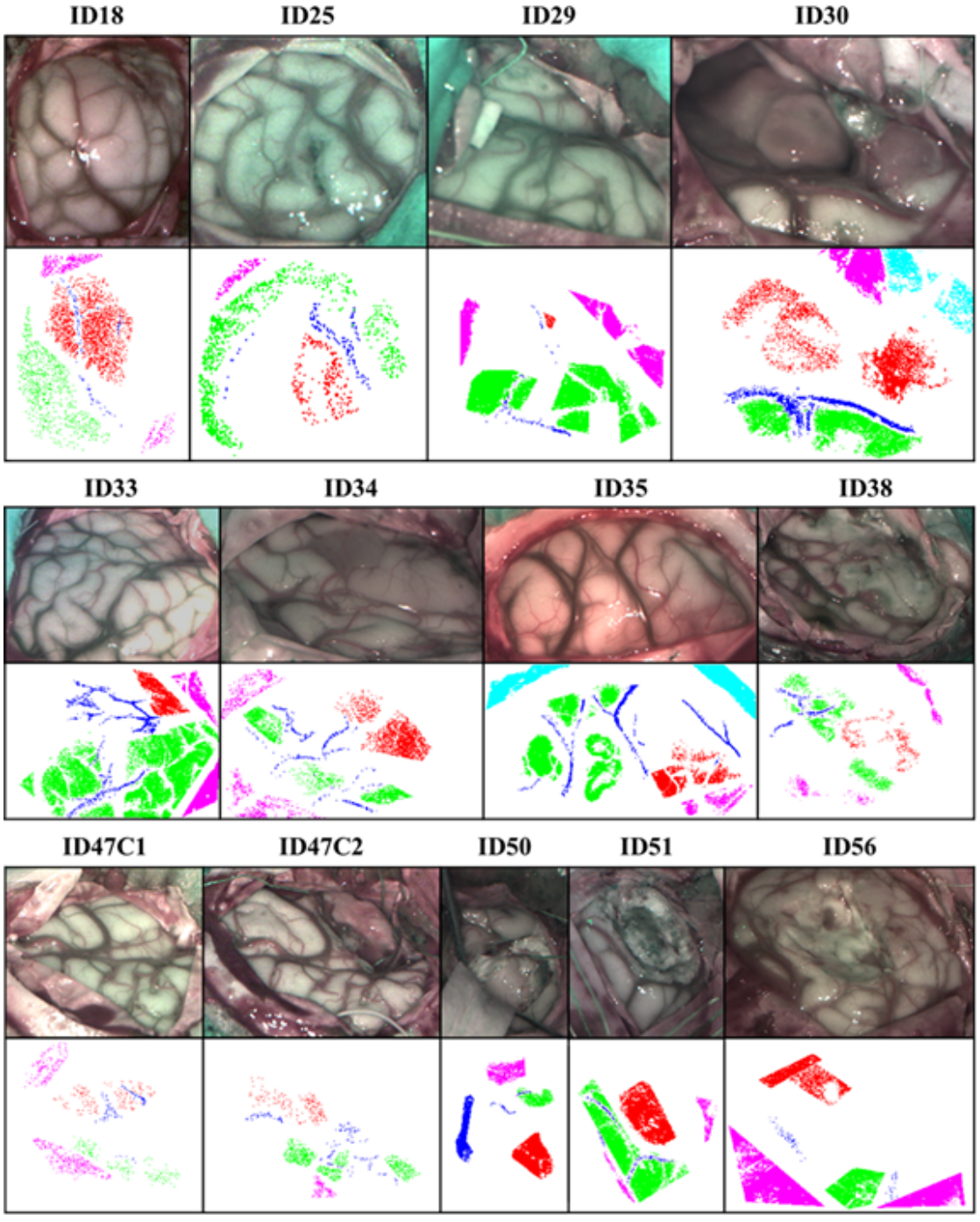

FIGURE 3.1: Synthetic RGB and ground truth maps of NEMESIS-3D-CM studied patients. Tumor is represented in red color, venous and arterial blood vessels in deep blue, healthy tissue in green, dura mater in pink color, and bone in light blue. Source: 'Supervised Machine Learning Methods and Hyperspectral Imaging Techniques Jointly Applied for Brain Cancer Classification' [8]

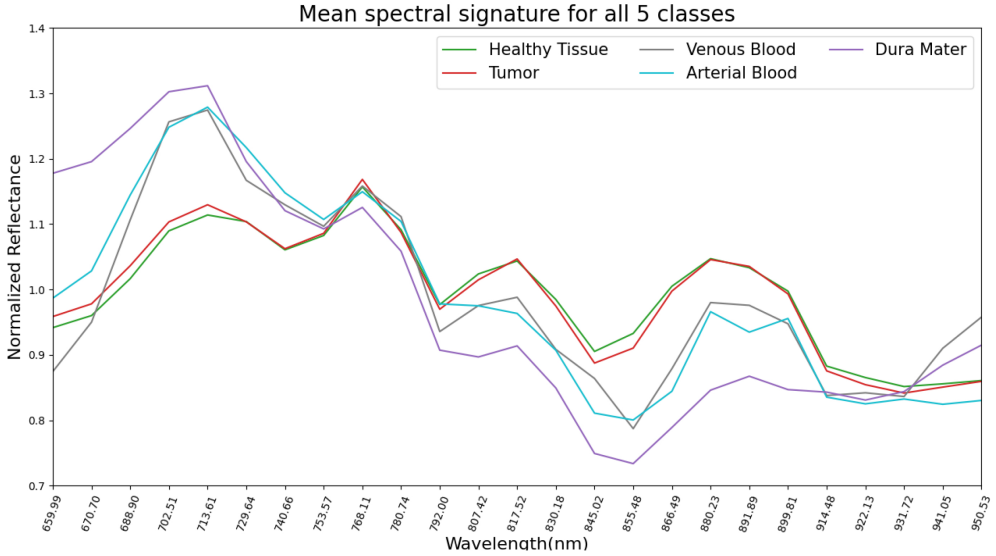

FIGURE 3.2: Mean spectral signatures for all 5 labeled pixel brain tissues among all thirteen images present in Table 3.1 and Figure 3.1. Source: 'Supervised Machine Learning Methods and Hyperspectral Imaging Techniques Jointly Applied for Brain Cancer Classification' [8]

3.1.2 Hyperspectral image processing chain

All images used for this thesis have been preprocessed to homogenize the spectral signatures before training the CNN models or predicting images. The procedures followed to preprocess the HSI images are:

1. Calibration: Necessary to maintain the reproducibility of the captured hyperspectral data. It is done to reduce the effects of the different light conditions in each surgery. To calibrate the hyperspectral cube, it is fundamental to have white and dark reference captures and then utilize the Equation (3.1), where I is the raw hyperspectral image, D is the black reference, and W is the white reference.

$$I_c = \frac{I - D}{W - D} \tag{3.1}$$

2. Hyperspectral cube formation: Once the raw brain image has been calibrated, the hyperspectral cube has to be formed. All raw images were captured using a Ximea Snapshot hyperspectral camera [46]. This camera includes the spectral resolution within the spatial resolution, since all 25 wavelength bands are stored in 5x5 repetitive mosaic blocks. Therefore, raw images are captured in 2D with a resolution of 2045 $\times$ 1085 pixels. Then they are transformed in a 3D hyperspectral cube with a final spacial resolution of 419 $\times$ 217 pixels and a spectral resolution of 25 bands.

3. Spectral correction: The Ximea snapshot camera [46] used has a high sensibility sensor, which gathers response curves with crosstalk at the peak wavelength of neighbors. To mitigate this, secondary harmonics need to be filtered with long or short pass filters. Afterwards, the manufacturer indicates to multiply the calibrated image by a provided correction matrix, shown in Equation (3.2):

$$I_{sc} = I_c \times SCM \tag{3.2}$$

where I_c is the calibrated hyperspectral cube band and SCM is the spectral correction matrix (25 × 25).

4. Normalization: Brain's curved shape implies that pixels are not located at the same height from the camera. Therefore, gathered light intensity is different among all image tissues, which may be classified differently depending on the brightness of each pixel. To diminish this effect, standardization techniques are used to preserve the spectral signature shape regardless of camera distance. The root mean square values (RMS) from spectral signatures over all bands are used as normalizing coefficients, calculated using Equation (3.3). Then, once the coefficients are computed, they are used in Equation (3.4) to normalize the spectral corrected cube.

$$coef\,[r,c] = \sqrt{\frac{\sum_{b=1}^{B} \left(I_{sc}\,[r,c,b]\right)^2}{B}} \tag{3.3}$$

$$I_{Norm}\,[r,c,b] = \frac{I_{sc}\,[r,c,b]}{coef\,[r,c]} \tag{3.4}$$

where I_{sc} is the spectral corrected cube with dimensions r x c x b (rows x columns x bands).

3.1.3 Data batches

As mentioned before, to speed up the training, data needs to be included in batches. Batches are small groups of data from the entire high dimensional data. To group small data chunks from the thirteen patient hyperspectral images used for this thesis, batches have included small hyperspectral patches. These patches are small 3D 7x7 pixels with 25 spectral channels, where the middle pixel corresponds to a labeled pixel from the available ground truth maps. Therefore, these patches not only have spectral information but also spatial information around the target pixel placed in the center. This can be seen in Figure 3.3 when randomly extracting a single patch from a patient hyperspectral cube. Generally, when multiple images are loaded for creating batches,

patches are randomly selected among all images. This would result in batches of patches from different patients.

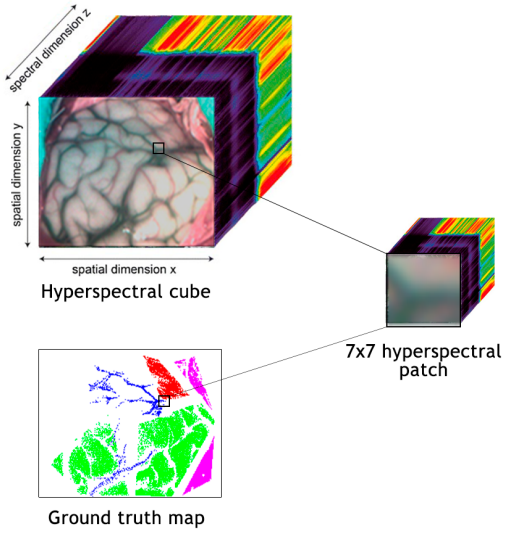

FIGURE 3.3: Simplified process of patch creation. Patch of size 7x7x25 generated from a hyperspectral cube and its ground truth map.

Batches have been configured to a fixed size of 16 patches, so in cases where the samples left can not conform a batch of 16 patches, they have been discarded. This means that from all loaded images for any experiment, a maximum of 15 labeled pixels would not have been used for training or test. Besides, to ensure all patches are balanced, the random stratified sampling [47] method have been used. Using this technique assures that each batch approximately has the same number of patches for each class. In cases where classes with few labeled samples have been already used, patches from the largest class have been added to the batch.

3.2 Validation and test methodology

3.2.1 Evaluation metrics

After training the models, a confusion matrix is obtained immediately after predicting either the validation or test data. In the confusion matrix, TP stands for true positives, TN for true negatives, FP for false positives and FN for false negatives. Please note that in this study, the main class of interest is tumor. Therefore, TN is any class which was not classified as tumor. From the confusion matrix, the $OACC$ metric is calculated to select the best model. This is done during the cross-validation to finally use the elected one to classify new images. The $OACC$ metric is described in Equation (3.5), where all correct predictions are divided by all predictions made.

$$OACC = \frac{Confusion\,matrix\,main\,diagonal}{Total\,number\,of\,predictions} \tag{3.5}$$

Additionally, the SEN metric has been used to evaluate how classifiers can correctly predict the class of interest, described in Equation (3.6):

$$SEN = \frac{TP}{TP + FN} \tag{3.6}$$

3.2.2 5-fold double-cross validation

Cross validation is a technique used in machine learning to estimate how statistic models overfit to an independent dataset. In a single cross-validation, a division of the dataset is done to have two sets, one for training models and the other to test the trained model. However, with single cross validation, models may encounter overfitting problems [48], [49]. To solve this issue, it is useful to divide the training set into calibration and validation sets, being the first one used to train models and the validation set to check how the models adjust to the calibration set. Then the trained model is evaluated on the test set. This way, the number of pixel samples that can be used to train the model is reduced, ensuring that every sample has been used at least once for validation, calibration, and testing. One way to ensure that model hyperparameters are not overfitting the dataset while avoiding bias is by using nested cross-validation, also known as double-cross validation. The dataset is first splitted into two sets, the test set and the other destined to train the model, representing the first cross-validation iteration. Then, inside the second cross-validation iteration, the train split is splitted to have the calibration and validation sets. The way the data has been splitted for this thesis was using K-fold cross-validation [50], with a K value of 5.

With the data batch and patch description provided in Subsection 3.1.3 and the 5-fold double cross validation explained in this section, Figure 3.4 shows how the implementation has been done. This figure illustrates how the K-fold cross validator has been used to generate indices for every batch. Once the indices for the first double cross-validation loop have been obtained, the batches have been splitted to run all five combinations. The set of batches destined to test the model have been isolated in every K iteration. Meanwhile, the other four sets of batches to train have been splitted again to obtain the indexes for validation and calibration. This indices have been used for the second loop of the double-cross validation. The set of batches for calibration have been used for training a CNN model and the validation set of batches has been used to evaluate the model. After evaluating every model, the $OACC$ has been obtained for the Kn iterations. The model with the highest $OACC$ has been tested on its corresponding K set

of test batches. Once the best five models during the double cross-validation have been trained, the CNN with the highest $OACC$ has been used for classification purposes.

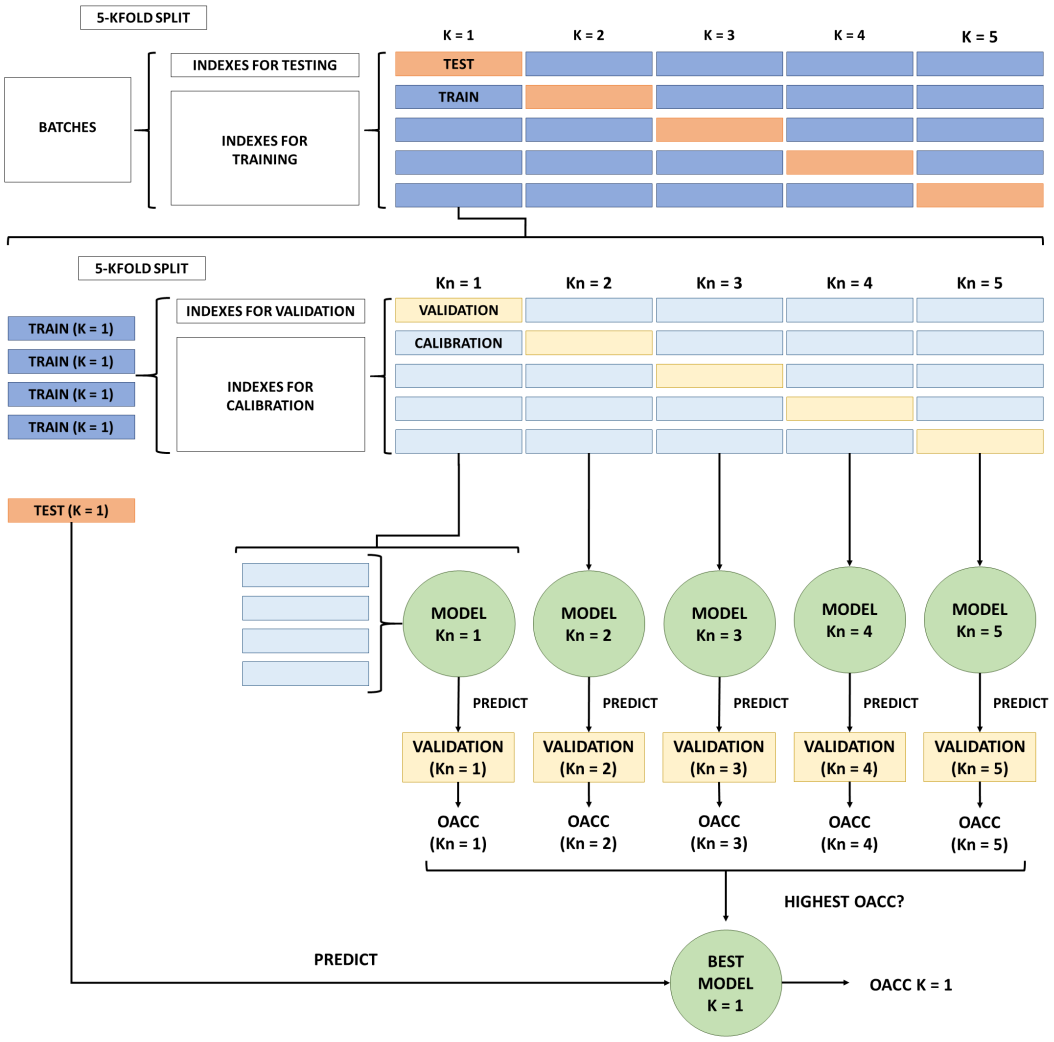

FIGURE 3.4: 5-fold double cross validation schema for one K iteration using the created set of batches.

3.3 PyTorch Convolutional Neural Networks

3.3.1 Conv2DNet architecture

The architecture used to train the CNN model can be seen in Figure 3.5. The name of Conv2Net comes from the single 2D convolutional layer applied to the input 7x7 HSI patches, which are distributed among all training batches. This 2D convolutional layer does not apply any padding but uses a kernel size of 2 with a stride of 1 to extract features from the patches. It outputs 16 feature maps of size 7x7. Then, a

2D max pooling is applied to the feature maps with a kernel size of 2. With this pooling, the feature maps have been subsampled to speed computation and to output another 16 of size 3x3. Afterwards, the 16x3x3 feature maps pass through a ReLU activation function before they are flattened to connect the convolutional phase with the linear one. The linear phase of the Conv2DNet has two fully connected layers. The first one contains 144 neurons, where another ReLU activation function is used, and the second has 64 neurons. Finally, a softmax has been utilized in the latest linear layer to output probabilities for all 5 possible brain tissue classes. Furthermore, the Adam [23] optimization algorithm has been used for gradient-based optimization of stochastic objective functions. Besides, the mini-batch optimization technique [19] has been performed during training by gathering the input patches in mini-batches of size 16, using 100 epochs to run through the whole set of mini-batches. Finally, the criterion used has been the cross-entropy loss implementation from PyTorch [51], used for solving classification problems with a discrete number of classes.

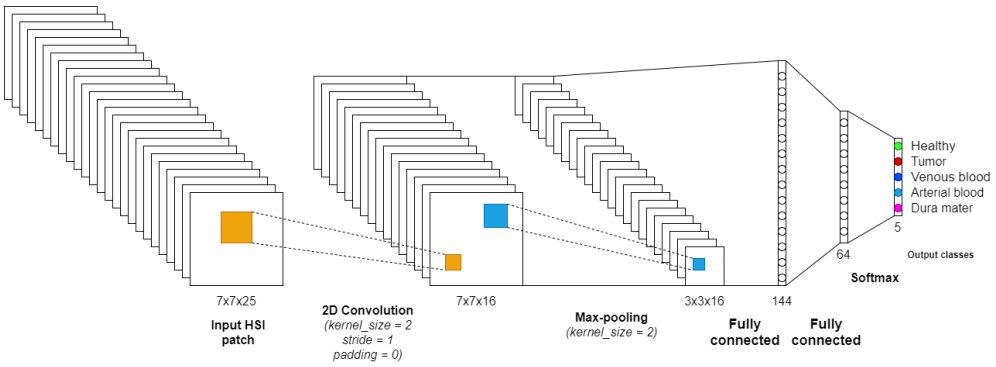

FIGURE 3.5: Conv2D architecture including a 2D convolution layer, max pooling and 2 fully connected layers with a softmax classifier for healthy, tumor, venous, arterial and dura mater classes.

Chapter 4

Microsoft Azure Machine Learning set-up

Before training, manage, and deploy machine learning experiments, a machine learning workspace needs to be created, which is a top-level resource that provides a centralized place to manage everything created within Azure Machine Learning. The workspace itself is stored in a resource group, which is a container holding the resources for an Azure solution. Workspaces run in virtual machines in specific regions to compute targets. In our case, the selected region is west europe since it is the closest to where this final master thesis has been developed. To store everything created in the machine learning workspace, a storage account with a datastore is also needed.

Once the necessary Azure resources have been prepared, the development shown in Figure 4.1 could be started. This image represents the implementation done regarding the machine learning model building, testing, and deployment, presented in Figure 1.1 from the Introduction Chapter 1. As illustrated in Figure 4.1, the cloud provider used to provide a hyperspectral brain classification service has been Microsoft Azure. Within Azure, multiple services have been used. To train PyTorch CNN models, Azure Machine Learning service has been used. This service has allowed to package, validate, and deploy the trained models. Once the desired models have been registered within the Azure Machine Learning workspace, they have been deployed as a web service using AKS. The aim of this web service would be to provide the brain classification service to multiple hospitals at the same time. This way they could use it during brain surgery. In the following sections, in-depth discussions have been provided regarding how the implementation has been done.

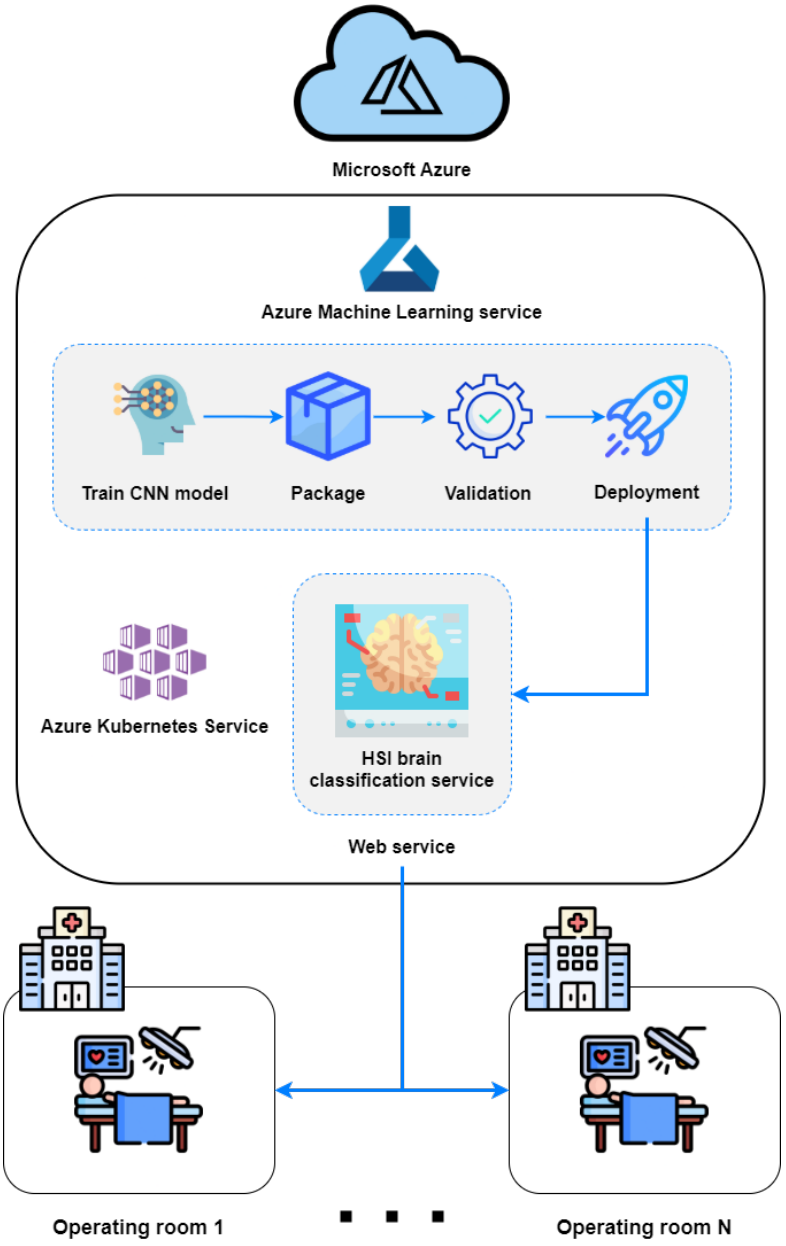

FIGURE 4.1: Microsoft Azure implementation for the NEMESIS-3D-CM IoT approach from Figure 1.1.

4.1 Compute cluster

Inside the workspace, a single compute cluster has been created to distribute the training processes across two GPU compute nodes in the cloud. This cluster is stored in a dedicated virtual machine with the name 'Standard_NC6'. It has as resource properties 6 cores, 56 GB RAM, 380 GB disk, and 1 NVIDIA Tesla K80 GPU. Minimum and maximum number of nodes have been set to 0 and 2, respectively, while the idle seconds before scale down have been set to 1800 since it was the default option. More information

regarding virtual machines powered by the NVIDIA Tesla k80 GPU can be found inside the Azure Virtual Machine documentation [52].

4.2 Datastore and datasets

As presented in Figure 4.2, an overview of how data has been used in Azure Machine Learning is described. Data has been uploaded to the default Azure Blob storage datastore, included in the Azure resource group, to train CNN models in the cloud. The Azure SDK for Python [42] has been used to upload the necessary data for training and testing CNN models.

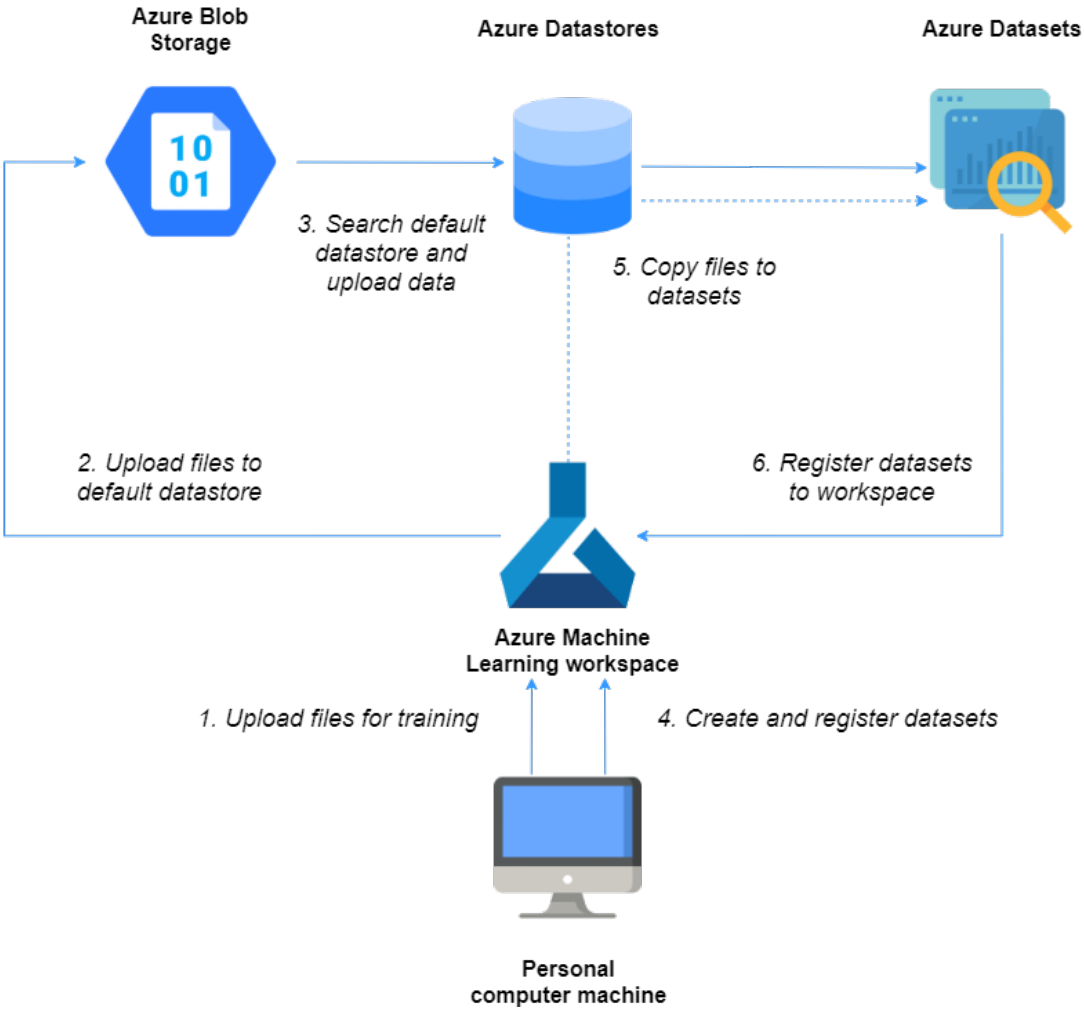

FIGURE 4.2: Schema of data upload and dataset registration in Azure Machine Learning using the Azure SDK for Python.

Once a connection to the workspace has been established and the workspace object has been returned, a call to 'get_default_datastore' has been made to get the default datastore object. Then, using the default datastore object from the connected workspace, a call

to 'upload_files' has been made to upload the necessary files. For that purpose, a new folder has been created in the datastore with two subfolders including all ground truth maps and preprocessed images already described in Subsection 3.1.1.

Azure Machine Learning provides the dataset asset as an abstraction for data. Datasets are versioned references to a specific set of data, which can be tabular or file-based. They are used to reference the uploaded data in Azure Blob storage. For this thesis, file-based data has been used consisting of ground truth maps and preprocessed image. Therefore, two datasets have been created by using the default datastore object. The process of registering these two datasets consists of generating two FileDataset objects using the Dataset.File available in the Azure SDK for Python [42] using the created path folders on the default datastore. Afterwards, these FileDataset objects have been used to call the 'register' method to register these datasets in the working workspace. The reason datasets have been registered is to make them easily accessible to any experiment being run in the workspace.

4.3 Experiment runs

4.3.1 Environment

To run experiments in Azure Machine Learning, a context regarding the execution of the experiment run has to be provided. Experiment contexts are made up of an environment including the necessary packages to run the scripts, and the compute cluster where the script will run. For this thesis, a Python environment has been used with some conda and pip packages. Conda packages used are 'scikit-learn', 'ipykernel', 'matplotlib', 'numpy', 'pillow', and 'pip'. Pip packages used are 'azureml-sdk', 'pyarrow', 'torch', 'scipy', and 'tqdm'. Once an environment has been created or registered in the Azure Machine Learning workspace, it can be used to run scripts as experiments in the defined compute cluster.

4.3.2 Running training scripts in the compute cluster

Before running an experiment in the defined compute cluster inside Azure Machine Learning, a training script has to be programmed. Multiple training scripts have been developed for this thesis to obtain results for the described experiments in Chapter 5. A folder has been created for every experiment to include all necessary Python file dependencies before submitting to Azure Machine Learning. Once all necessary files are ready, the environment has been created as well as the compute cluster, an instance of

the ScriptRunConfig class available in the Azure SDK for Python [42] has been created. This instance is used when submitting the experiment to the workspace, including the parameters in the constructor described in Table 4.1.

By passing these parameters to the training script, a configure can be done for different experiments only by changing any of the parameters passed to the training script. This has been done by using a control script, which is a file preparing the connection to the Azure Machine Learning workspace and datasets, uses or creates environments for the experiment and submits to Azure Machine Learning any experiment. Once the script configuration run has been created and the experiment has been defined, it can be submitted to the workspace as a single run. Figure 4.3 illustrates an overview of how the training procedure has been done in Azure Machine Learning.

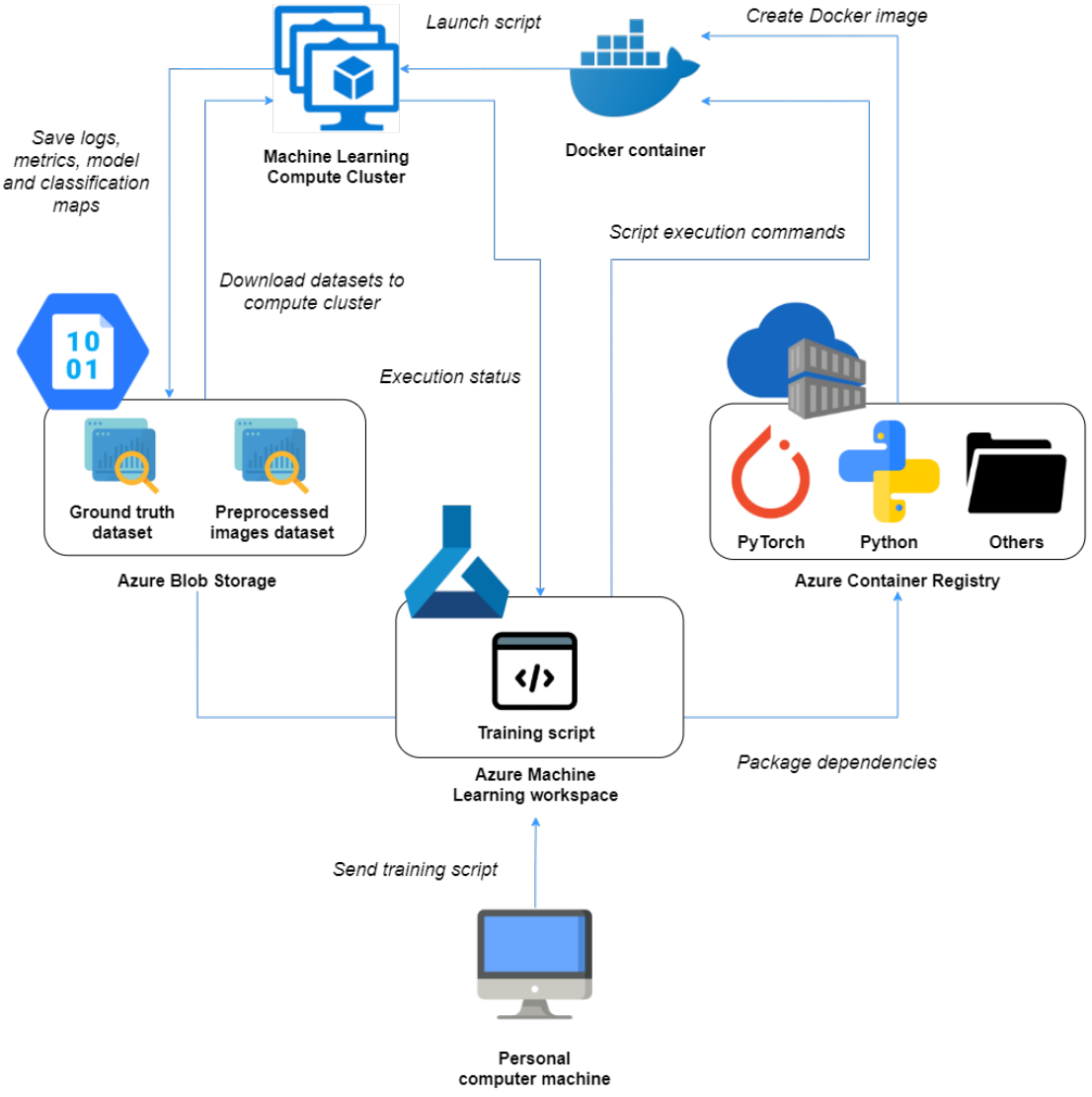

FIGURE 4.3: Schema of training PyTorch CNN models in Azure Machine Learning.

TABLE 4.1: Description of every parameter used in the ScripRunConfig class constructor to train models in Azure Machine Learning.

Paramater	Description
arguments	All passed arguments to the training script. These arguments are parsed inside the training script and are the following ones: • Temporary path location on the compute cluster where the ground truth maps from the dataset have been downloaded. A call to 'as_download' method on FileDataset object is done to cause the files in the file dataset to be downloaded to a temporary location on the compute where the script is being run. • Temporary path location on the compute cluster where the preprocessed images from the dataset have been downloaded. A call to 'as_download' method on FileDataset object is done to cause the files in the file dataset to be downloaded to a temporary location on the compute where the script is being run. • String with all patient IDs for training separated by comas. The training script parse the string and splits it to get all patient IDs. These are used when accessing the necessary files from temporary path locations. • String with the patient ID used for testing. Used when accessing the necessary files from the temporary path locations. • Number of epochs for the CNN model to run over the training batches. • Batch size. Indicates the number of patches to include each batch. • Patch size. Indicates the squared spatial dimension of the HSI patches. • Number of K folds. Used to split the data K times during the double cross-validation. • Learning rate when training the CNN model. • Model name. Used when registering trained models in Azure Machine Learning to properly identify them.

Continued on next page

Table 4.1 – continued from previous page

Parameter	Description
source_directory	The folder where the training script is and all other necessary scripts.
script	The name of the training Python script included in 'source_directory'.
environment	The defined environment object described in Subsection 4.3.1 with all necessary packages for the training script to run properly.
compute_target	The defined compute cluster object described in Subsection 4.1 with the necessary resources for the training script.

Once the training scripts are sent to the Azure Machine Learning workspace, it generates all necessary dependencies in an Azure Container registry. Then it creates a Docker image to send all packages as well as the scripts to proceed with the training. After establishing connection with the compute cluster, it downloads the datasets to the cluster itself. Then the training script is launched, which saves logs, metrics, the trained model, and the classification maps to the Azure Blob Storage. Meanwhile, it retrieves information regarding the execution status to the Azure Machine Learning workspace.

4.3.3 Metrics

During the execution of the experiment run, the training script is obtaining multiple metrics. It does not only store in the experiment log the classification metrics described in Subsection 3.2.1, the computed classification maps, and the passed arguments from the control script, but also some execution time metrics. Every metric is stored within the experiment run in the Azure Machine Learning workspace. These time metrics have been described in Table 4.2 and have been computed using the default timer in timeit library from Python.

TABLE 4.2: Description of every metric computed when training CNN models in Azure Machine Learning.

Paramater	Description
Time loading arguments	Time spent loading arguments from the control script.

Continued on next page

Table 4.2 – continued from previous page

Parameter	Description
Time preparing train data	Time preparing the train data and converting the batches to PyTorch tensors before the training procedure. Note that this execution time will be longer when more patient images are used.
Time training CNN	Time training CNN models, either when using the double cross-validation or training a single model. Please note that this execution time depends on the CNN architecture due to the update of every weight in every layer.
Time preparing test data	Time preparing test data and converting the batches to PyTorch tensors for the CNN model to classify. This procedure is needed to compute the classification metrics of the model.
Time predicting GT test image	Time predicting the ground truth pixels from the test image.
Time generating classification maps	Time generating classification maps, including ground truth maps and the entire preprocessed test cube.

4.3.4 Model registration

Once the training script has been executed, the CNN model is saved with the '.pt' extension in the output folder of the compute cluster. Then, it is loaded again with PyTorch to map the storage to the central processing unit (CPU). This has to be done since Azure Kubernetes Service has been used on a CPU-only machine. Otherwise, a runtime error will appear, indicating that the model could not be deserialized on a CUDA device, since a GPU in the AKS container is not available. Finally, the loaded

model with the CPU map location is uploaded to the output folder of the experiment and registered to the current run experiment.

4.4 Azure Kubernetes Service deployment

Although Azure provides the ACI containers [44], they have not been used for this thesis. The reason is that ACI containers have a timeout of 60 seconds which could not be configured for more time. Since more time is needed, Azure Kubernetes Service [45] has been used as an alternative to provide a model as a classification service for hospitals. Before preparing the model for the container, a Kubernetes service resource has been created inside the Azure portal. Region has been set to West Europe, availability zones 1, 2, and 3, Kubernetes version 1.19.11, node size 'Standard_DS2_v2' with 3 nodes and manual scale, since they have been the default options. Then, an AKS inference cluster has been deployed inside Azure Machine Learning studio using the Kubernetes service resource.

4.4.1 Deploy a CNN model as a web service

After connecting to the workspace with all necessary resources and the AKS inference cluster, the desired registered PyTorch CNN model in the workspace is loaded to be used in a web service. Then, to host this model in a container, a folder has been created to store all necessary Python files as well as the scoring script. This ensures that the model can provide the classification service for the hospitals correctly. An overview of how the deployment has been done as well as how it has been consumed is presented in Figure 4.4, which is a schema based on an image available in the Microsoft Azure documentation regarding real-time scoring models [53]. Since the container needs to install Python dependencies when it gets initialized, a '.yml' environment configuration file has been created to include conda and pip packages. Conda packages used are 'scikit-learn', 'ipykernel', 'matplotlib', 'numpy', 'pillow', and 'pip'. Pip packages used are 'azureml-sdk', 'azureml-defaults', 'pyarrow' version 3, 'torch', 'scipy', and 'tqdm'. Additionally, the Python version 3.8.10 has been included since at the time this thesis has been developed, higher versions did not work with the 'azureml-sdk' package.

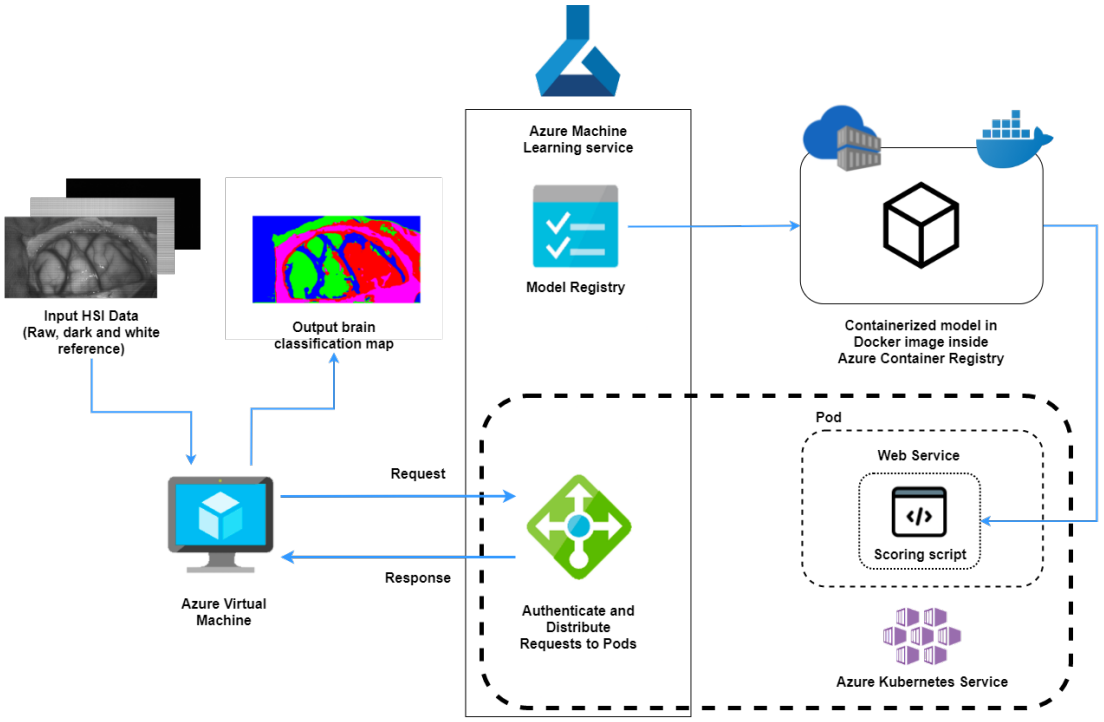

FIGURE 4.4: Schema of model deployment and consumption using Azure Kubernetes Service and Azure Machine Learning. Based on the architecture presented in the Microsoft documentation regarding real-time scoring models [53]

Once the environment and the files are ready to be used, the AKS inference cluster object in the workspace is retrieved using the AksCompute class available in the Azure SDK for Python [42]. Then, a deployment configuration object has been created using the AksEndpoint class, with 0.1 CPU cores, 0.5 GB of memory, and a timeout of 240 seconds (4 minutes) for deploying to an AKS compute target. Afterwards, the scoring environment has been prepared including the parameters described in Table 4.3, explained in the InferenceConfig class constructor, available in the Azure SDK for Python:

TABLE 4.3: Description of every parameter used in the InferenceConfig class constructor to deploy a model to AKS.

Paramater	Description
source_directory	The folder where the scoring script is and all other necessary scripts.
entry_script	The name of the training Python script included in 'source_directory'.
conda_file	The path to the local '.yml' file which contains the custom conda environment definition to use for the container image.

Continued on next page

Table 4.3 – continued from previous page

Parameter	Description
runtime	Specified as Python runtime to use for the container image.

Finally, the model has been deployed to a container with the workspace object, the custom name for the service, the loaded model, the inference configuration, the deployment configuration, and the AKS compute object. For this end, a call to the 'deploy' method from the Model class had to be done including all mentioned parameters. All web service deployments can be seen inside the Azure Machine Learning page, which are available as endpoints in the assets tab.

4.4.2 Scoring script

The scoring script is a Python file which gets the deployed model from the workspace, loads the input data to generate and return a classification map prediction. It needs to have two defined methods, 'init' and 'run'. The 'init' method is called when the service is loaded. It gets the path to the deployed model file and loads it. On the other hand, the 'run' method is called when a request is received. It receives a JavaScript Object Notation (JSON) object containing all sent files. When the script deserialize them, it preprocess the input HSI raw images (brain image, dark and white references) using the preprocessing chain described in Subsection 3.1.2. Afterwards, it converts the hyperspectral cube to PyTorch batches to feed the deployed CNN model. Patch coordinates are also obtained to properly compute the classification map for the input image. Then, the CNN model predicts all patches and the scoring script computes the classification map, which is finally encoded in a JSON object with some time metrics. These metrics are described in Table 4.4, which also includes the time measure regarding the deployment of a registered model to an AKS container.

These time metrics have been computed using the default timer in timeit library from Python. Furthermore, once the JSON object is ready, it is returned whenever a request is done to the service.

TABLE 4.4: Description of every time execution measure when deploying and consuming trained models.

Time measure name	Description
Deployment to AKS	Time spent between requesting AKS to deploy a registered model until it becomes available as an endpoint.
Data transmission	Time spent between sending the request data to the web service and when it has sent a response back to the client.
Parsing data	Time parsing the data whenever a request is done to the service.
Image preprocessing	Time preprocessing the raw brain hyperspectral images using the preprocessing chain described in subsection 3.1.2.
Data preparation	Time spent preparing the hyperspectral cube to PyTorch tensors and batches.
Map prediction	Time spent predicting and generating the classification map.
Scoring script run	Time elapsed when the run method from the scoring script is called until it returns a response to the client.

4.4.3 Web service usage with the Azure SDK for Python

Once the web service is up and running on an AKS container, it can be consumed by a client application through the Azure SDK. However, when working in production environments, businesses usually work with Hypertext Transfer Protocol (HTTP) requests to consume the web service. In this thesis, the SDK has been used to classify hyperspectral brain images.

In a local Python script, raw tif images from the brain and its white and dark references are loaded and then added to a Python list. Other variables like the batch and patch sizes as well as the patient identifier are included in the dictionary. Since the raw images are loaded as numpy arrays, a numpy array encoder has been used to properly serialize the images into a JSON object. Integer and string variables can be serialized without any problem into a JSON format. Once all variables and images have been dumped as a JSON object, a call to the 'run' method is done to the web service object including the JSON. This 'run' method retrieves a response which has been loaded as a JSON,

since the scoring script described in Subsection 4.4.2 returns a JSON object including a Python dictionary. This dictionary has been used to obtain the classification map and the execution time for every process in the scoring script. Finally, the local Python script plots the classification map as a figure, so that neurosurgeons can visualize the different classified brain tissues.

Chapter 5

Experiments and results

5.1 Training and testing models in Azure Machine Learning

Training times need to be taken into consideration when handling large datasets. Otherwise, deploying new trained models might take longer than desired. To analyze the time Azure Machine Learning takes to train the Conv2DNet described in subsection 3.3.1, multiple experiments have been deployed. Using the double cross-validation described in subsection 3.2.2 as well as the number of images during training has been taken into consideration. Since 13 hyperspectral brain images have been provided by NEMESIS-3D-CM, a maximum of 12 images have been used to train Conv2DNet models, while the remaining image has been utilized for testing and classification purposes. Nonetheless, other Conv2DNet models have been trained using 6 and even 3 images, while only testing 1 single image. Therefore, a total of six experiments have been deployed to analyze Azure Machine Learning training times. Half of them have used the double cross-validation while the other half have omitted the cross-validation step. Thus, in every experiment, a total of 12 Conv2DNet models have been trained to estimate the mean and standard deviation training times.

However, it is important to mention that these time results strongly depend on the CNN architecture. More complex architectures including further hidden layers might take longer to train. This is due to the additional parameters that have to be computed on every backpropagation pass. Moreover, using a different epoch number can highly influence the training times, considering that the models have been trained with 100 epochs for these experiments.

5.1.1 Training times

Training time executions have been counted between the end of the training data preparation and when the model has been finally computed. Therefore, other time execution processes done in the training scripts have not been taken into consideration. After computing the execution time spent by Azure Machine Learning to train models, the mean and standard deviation values have been plotted in Figure 5.1. Each bar indicates the time spent training 12 Conv2DNet. Blue bars express that the models have been trained without any cross-validation technique, while red bars imply that the double cross-validation described in subsection 3.2.2 has been used.

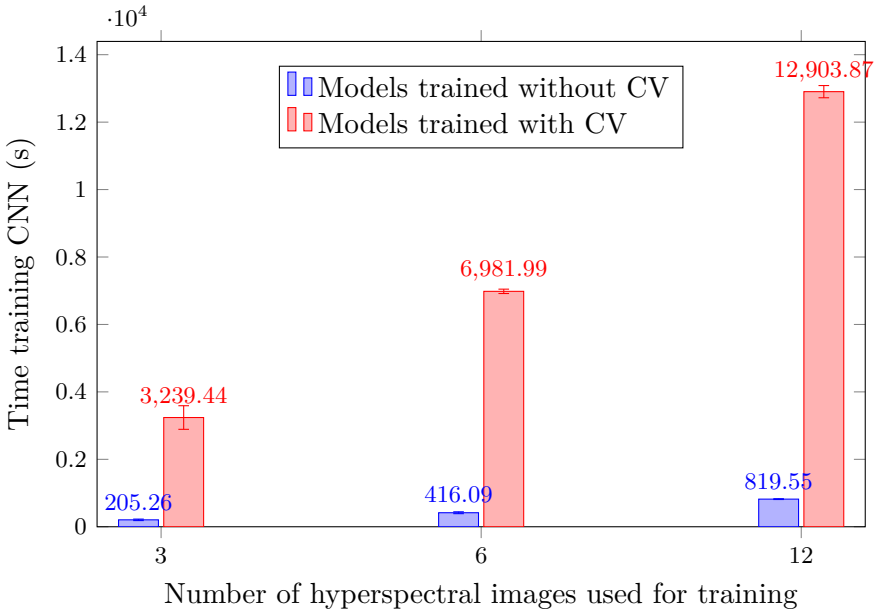

FIGURE 5.1: Mean and standard deviation of the time spent training all 12 Conv2DNet models for every experiment in Azure Machine Learning.

As shown in Figure 5.1, using the double cross-validation is 16 times slower than not using it, independently of the images used during training. This is evident since the proposed 5-fold double cross-validation in Subsection 3.2.2 trains 25 Conv2DNet models instead of 1. Besides, the figure shows how doubling the number of images almost implies doubling the time spent during training. Furthermore, to facilitate the reader how much time Azure Machine Learning has taken to train the models, Table 5.1 has been added. This table includes the mean time in hours, minutes, and seconds spent in each experiment. As presented in the table, using the double cross-validation implies around 1 hour of training time when using 3 hyperspectral images and 100 epochs. In case of not using the cross-validation technique, only 3 minutes are needed to train. Likewise, training with 12 images while computing the double cross-validation involves 3 and a half hours to train the model, which does not happen when the model has been

trained without the cross-validation. In this case, only 14 minutes have been used during training.

TABLE 5.1: Mean value time spent during training CNN models in Azure Machine Learning. Measured seconds from Figure 5.1 have been included as well as their conversion to hour, minute, and second format.

Training images	**Cross-validation**	**Time (s)**	**Time (HH:MM:SS)**
3	No	205.26	00:03:25
3	Yes	3239.44	00:53:59
6	No	416.09	00:06:56
6	Yes	6681.99	01:51:22
12	No	819.55	00:13:40
12	Yes	12903.87	03:35:04

Although cross-validation techniques are recommended when training ML models, it is necessary to evaluate how the models have classified the test image not used during training for these experiments.

5.1.2 Classification metrics

Is it worth it to spend more time on the double cross-validation when training models? Would it make a considerable difference to the $OACC$ and SEN of the models? To answer these questions, Figures 5.2 and 5.3 are presented regarding the $OAAC$ and the SEN over the tumor class respectively. Both figures describe these two classification metrics when Conv2DNet models have been trained using 12 hyperspectral brain images. This has been done to compare Conv2DNet models with the 3D-CNN model proposed by Urbanos et al. [8]. Thus, the metrics shown in Figures 5.2 and 5.3 represent how each model has classified the patient indicated in the x axis, which has not been used during training. Additionally, both figures include models trained with and without double cross-validation.

Looking at the $OACC$ in Figure 5.2, Conv2Net trained models without cross-validation have been the best classifiers in 5 out of the 13 cases (ID29, ID30, ID33, ID34, and ID35). When these models have been trained with the double cross-validation, they have performed best in 4 out of the 13 cases (ID38, ID47C1, ID51, and ID56). Moreover, 3D-CNN model had performed best in the remaining 4 out of the 13 cases (ID18, ID25, ID47C2, and ID50). However, when comparing the two ways Conv2DNet models have been trained, non cross-validation models have performed better than cross-validation models in 7 out of the 13 cases, which have performed better only in 4 out of the 13 cases.

Above all, *OACC* values vary between 0.25 and 0.92, possibly showing inconsistency with the data used for training.

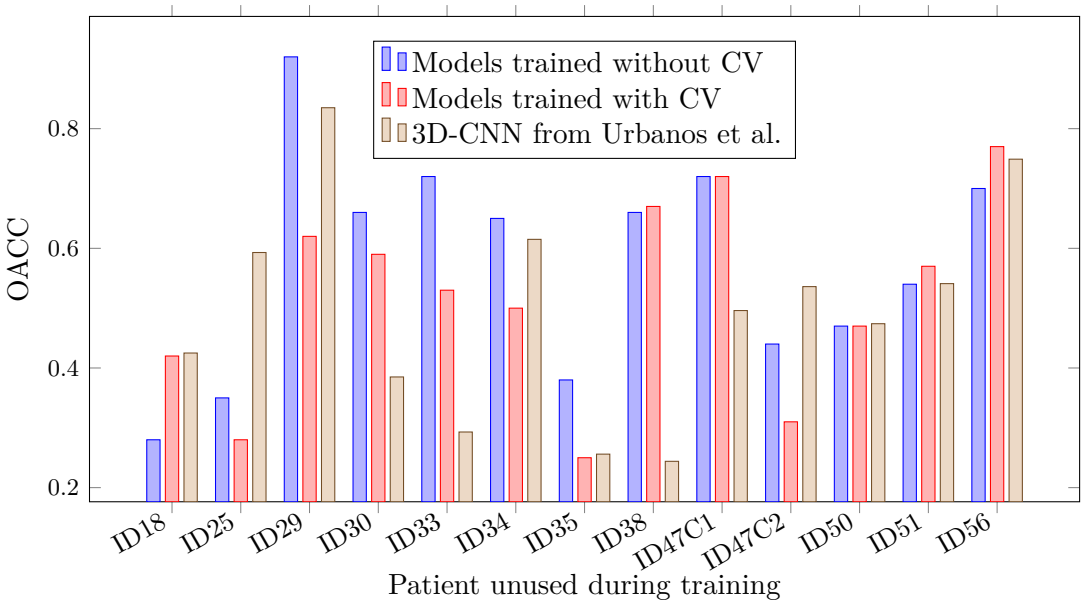

FIGURE 5.2: Overall accuracy values. Comparison between Conv2DNet and 3D-CNN from Urbanos et al. [8]. Models trained with 12 hyperspectral images.

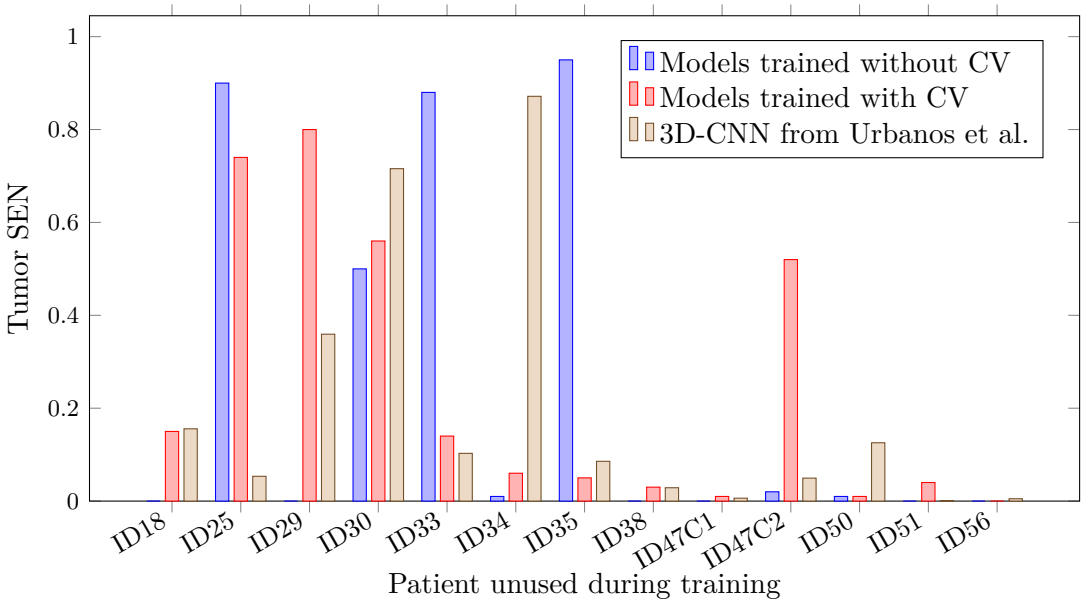

FIGURE 5.3: Sensitivity values for the tumor class. Comparison between Conv2DNet and 3D-CNN from Urbanos et al. [8]. Models trained with 12 hyperspectral images.

Nonetheless, further research needs to be done, since as Urbano et al. [8] mentioned in their work, these classification fluctuations among experiments might happen because of the biological differences in the patients. Then, no strong evidence could be provided disfavoring the usage of double cross-validation in relation to classification performance.

This can also be seen in Figure 5.3, where non cross-validation models have only performed better in 3 out of the 13 cases (ID25, ID33, and ID35). On the other hand, Models trained with the double cross-validation have performed best in 5 out of the 13 cases (ID29, ID38, ID47C1, ID47C2, and ID51), while the 3D-CNN model proposed by Urbano et al. has performed better in the other 5 out of 13 cases left (ID18, ID30, ID34, ID50, and ID56). In this figure, tumor SEN values highly fluctuate between 0 and 0.97 values. This could be explained due to the similarities in the labeled spectral signatures presented in Figure 3.2, where some tissues appear to be very similar in the measured spectrum. As mentioned in the work performed by Urbanos et al., this could lead to inaccuracies in the trained models.

5.2 Model deployment and consumption using AKS and Azure SDK

Once a model has been trained and registered in the cloud, some time executions are important to acknowledge. Since the purpose of the thesis is to provide a classification service during brain tumor surgery, the response of it should be the minimum possible for the neurosurgeons to continue their labor. Nevertheless, another important aspect to consider is the time spent deploying the model as a classification service for multiple hospitals.

With this in scope, two experiments have been done to estimate how much time it will take for Azure Machine Learning and Azure Kubernetes Service to deploy and consume trained models. To this effect, some models trained during the experiments described in Section 5.1 have been deployed on AKS containers and consumed using the Azure SDK for Python. In this case, only models trained on 12 hyperspectral images have been used, independently if the double cross-validation has been used or not.

5.2.1 Web service time latency

Web service deployment and consumption times have been measured using a client laptop connected to ethernet, reducing any latency via wi-fi connection. These time execution measures are presented in Figure 5.4, where blue bars indicate all 13 deployed models that have been trained without cross-validation, and red bars the other 13 deployed models trained with the double cross-validation. The figure illustrates seven different measures, which have been already described in Table 4.4.

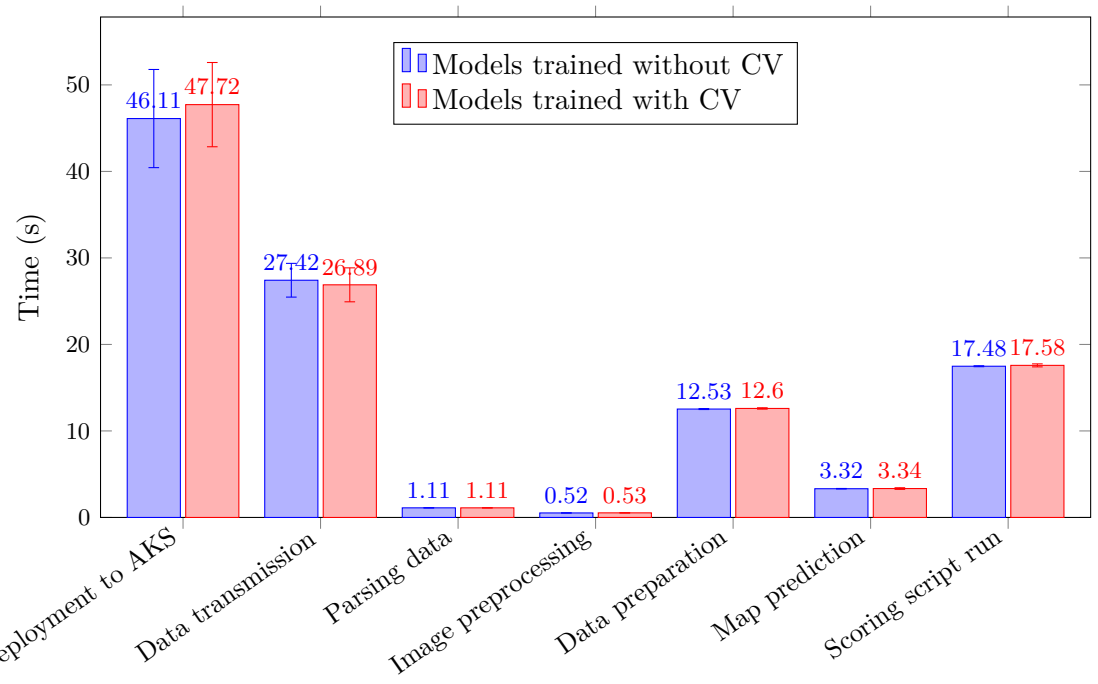

FIGURE 5.4: Web service time measures when deploying and consuming models trained on 12 hyperspectral images with and without the double cross-validation. Values correspond to the mean and standard deviation measured times for every 12 models in both experiments.

As shown in Figure 5.4, deploying a trained PyTorch model in Azure Machine learning takes almost 47 seconds, independently if the models have been trained using the cross-validation or not. Measures may vary depending on the internet connection and network congestion, but when measuring the time when data has been transmitted to the AKS container and it has returned back the classification map, almost 27 seconds have passed. This amount of time could be acceptable for the neurosurgeons to wait for the classification service response. However, when analyzing the remaining time execution metrics, it can be seen that preparing the preprocessed cube to PyTorch tensor batches has been the most time consuming task. This data preparation measure has taken almost 12.5 seconds to execute, while the image preprocessing task using the preprocessing chain described in Subsection 3.1.2 have proved to be very efficient, lasting almost half a second to complete. Predicting the preprocessed cube to generate a classification map has lasted around 3.3 seconds, while parsing the data from the client request takes approximately 1 second to be completed. The last measure, the scoring script run, is the sum of the four previous measures, which starts when the data from the client is received until the classification map is returned to the hospital client. This would indicate that by optimizing the data preparation task, the brain classification service could became quicker. Nonetheless, Figure 5.4 demonstrates that the time spent when deploying or consuming PyTorch models has no influence if cross-validation methods have been used, since the mean values for both experiments have almost been the same.

5.2.2 Classification maps

To visually evaluate how the brain classification service has performed, some computed classification maps have been presented in Figure 5.5 as well as their corresponding ground-truth maps and synthetic RGB images. First two row images have been cropped from the original raw capture. These raw images have been used when sending requests to the service provided by AKS containers. For that reason, the classification map dimensions are bigger than the synthetic RGB and ground-truth images. Additionally, the models used to classify images ID33 and ID51 have been trained on 12 images using the double cross-validation described in subsection 3.2.2 without the image destined to classify. Since tissues have similar spectral signatures as shown in the mean spectral signatures of Figure 3.2, models have had trouble properly identifying tissues. This has already been reported in the work done by Urbanos et al. [8]. Additionally, the model deployed to classify image ID35 has been trained using 3 images (ID30, ID33, and ID35). Therefore, the obtained classification map is clearer identifying different brain tissues when comparing it with its ground-truth map. Although the classified image has been used during the training of the model, it has been presented in this figure to illustrate the reader the possibility of using HSI as a non-invasive and non-ionizing tool, requiring no contact with the patient, to identify brain tumors.

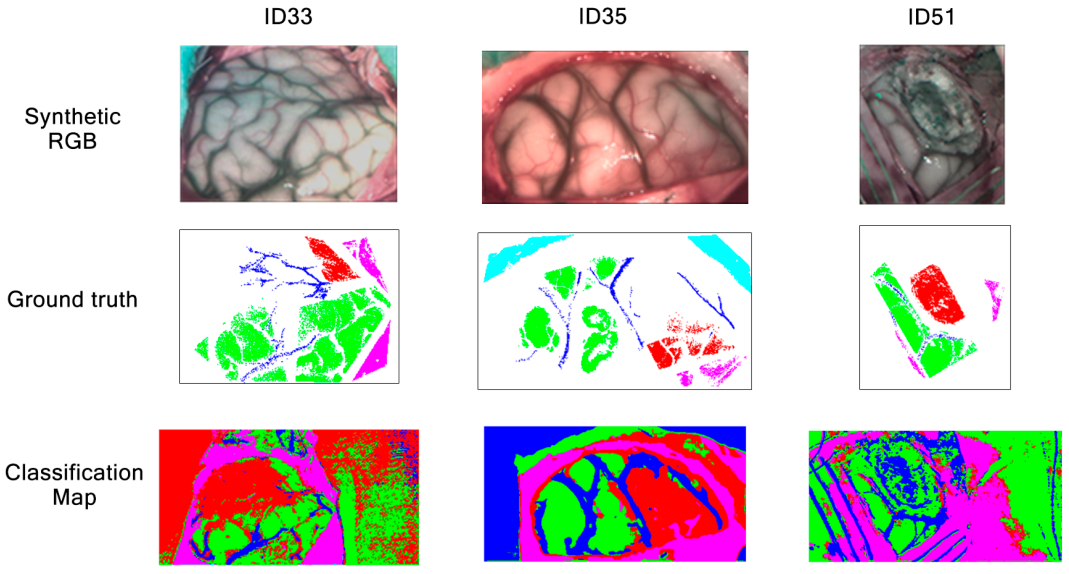

FIGURE 5.5: Classification maps from brain hyperspectral images computed by deployed models in AKS containers.

Chapter 6

Budget

For the development of this thesis, the estimated spent budget is shown in Table 6.1. Additionally, the following resources have been utilized:

- Computer station with 16 GB RAM memory.
- Python, PyTorch, Azure SDK for Python among other necessary package dependencies to manage HSI data, train CNN models, and communicate with Azure Machine Learning.
- Microsoft Visual Studio Code with all necessary extensions to work with Python and communicate with Azure Machine Learning.
- Microsoft Azure credit subscription to use Azure services.
- NEMESIS-3D-CM hyperspectral image database to do the experiments.
- Ximea snapshot hyperspectral camera for capturing brain images inside the operating theatre.

Table 6.1: Estimated budget for the development of the thesis.

Concept	**Price/hour**	**Subtotal**
320 hours of enginering development	30€	9.600€
30 hours of tutor consultancy	50€	1.500€
Computer station with required software	-	1.315€
Microsoft Azure subscription	-	500€
Ximea snapshot hyperspectral camera usage for 6 months	-	5.000€
	Total	17.915€

Chapter 7

Conclusions

Although HSI has already been proved to be a viable tool for brain tumor classification [8], this thesis has analyzed its use from an IoT perspective, training CNN models in the cloud provider Microsoft Azure and providing a decentralized service for multiple hospitals using AKS.

Results in Chapter 5 have shown how much time Azure Machine Learning has taken to train a simple CNN model on 12 hyperspectral brain images, using the proposed 5-fold double cross-validation in Subsection 3.2.2. As presented in Figure 5.1, almost three hours and a half have been required to obtain the best CNN model. Considering that multiple hospitals would send their brain images to the cloud week after week, 12 images are a relative low number to train models regarding their quality. Increasingly adding more images would take more time to train a CNN model, which implies waiting extra time before consuming it during surgery. Besides, experiments without the 5-fold double cross-validation have been done to train CNN models. Since only 1 model has been trained in contrast to all 25 that the 5-fold double cross-validation has trained, the training times are 16 times faster, needing 13 minutes when 12 hyperspectral images have been used. However, as presented in the $OACC$ and SEN in Figures 5.2 and 5.3 respectively, the classification results have had a high variance. Figures have shown that the results have been highly influenced by the patients biology and the spectral signature similarity along the measured spectrum, presented in Figure 3.2. This indicates that the experiments have no evidence at all to suggest the non-usage of cross-validation to speed up the training procedure.

Regarding model deployment and web service consumption times, the results presented in Figure 5.4 have shown that almost 47 seconds have been needed to deploy any model as a web service with an AKS. No significant differences have been encountered between the deployed cross-validated and non-cross-validated models. Nonetheless, spending

47 seconds seems a reasonable amount of time to deploy a trained model as a web service. Furthermore, when analyzing the time these web services take to provide a response when a request has been made, only 27 seconds have been needed to provide a classification map from the raw hyperspectral brain image sent. Although 27 seconds can be an acceptable amount of time for neurosurgeons to wait during surgery, the results presented in Figure 5.4 have also shown that by optimizing the data preparation task, response times could even be reduced. Additionally, three classification maps computed by the three deployed models in AKS as web services have been presented in Figure 5.5, which serve the reader as visual examples similar to what a neurosurgeon will observe during surgery.

As future work and to further improve times, analyzing which cloud resources, such as virtual machines and GPU models, could provide a faster training procedure. Additionally, using methods to re-train registered models could be used to reduce training times before deployment. Furthermore, to reduce the latency response of the web service, the data preparation task could be optimized so neurosurgeons would wait less time for a classification map during surgery. Finally, to obtain better classification results, 3D-CNN models could be trained as well as conduct further research over the HSI images to help reduce inaccuracies on CNN models.

Bibliography

[1] SEOM. Las cifras del cáncer en españa 2020. Available at https://seom.org/seomcms/images/stories/recursos/Cifras_del_cancer_2020.pdf (2020/01), 2020.

[2] Jigisha P Thakkar, Therese A Dolecek, Craig Horbinski, Quinn T Ostrom, Donita D Lightner, Jill S Barnholtz-Sloan, and John L Villano. Epidemiologic and molecular prognostic review of glioblastoma. *Cancer Epidemiology and Prevention Biomarkers*, 23(10):1985–1996, 2014.

[3] Nader Sanai, Mei-Yin Polley, Michael W McDermott, Andrew T Parsa, and Mitchel S Berger. An extent of resection threshold for newly diagnosed glioblastomas. *Journal of neurosurgery*, 115(1):3–8, 2011.

[4] Daniel A Orringer, Alexandra Golby, and Ferenc Jolesz. Neuronavigation in the surgical management of brain tumors: current and future trends. *Expert review of medical devices*, 9(5):491–500, 2012.

[5] B Albayrak, AF Samdani, and PM Black. Intra-operative magnetic resonance imaging in neurosurgery. *Acta neurochirurgica*, 146(6):543–557, 2004.

[6] Rahul Sastry, Wenya Linda Bi, Steve Pieper, Sarah Frisken, Tina Kapur, William Wells III, and Alexandra J Golby. Applications of ultrasound in the resection of brain tumors. *Journal of Neuroimaging*, 27(1):5–15, 2017.

[7] Nicholas Ferraro, Eric Barbarite, Trevine R Albert, Emmanuel Berchmans, Ashish H Shah, Amade Bregy, Michael E Ivan, Tyler Brown, and Ricardo J Komotar. The role of 5-aminolevulinic acid in brain tumor surgery: a systematic review. *Neurosurgical review*, 39(4):545–555, 2016.

[8] Gemma Urbanos, Alberto Martín, Guillermo Vázquez, Marta Villanueva, Manuel Villa, Luis Jimenez-Roldan, Miguel Chavarrías, Alfonso Lagares, Eduardo Juárez, and César Sanz. Supervised machine learning methods and hyperspectral imaging techniques jointly applied for brain cancer classification. *Sensors*, 21(11), 2021.

ISSN 1424-8220. doi: 10.3390/s21113827. URL https://www.mdpi.com/1424-8220/21/11/3827.

[9] Phung and Rhee. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, 9:4500, 10 2019.

[10] Wikipedia. Convolutional layers, 2021. URL https://en.wikipedia.org/wiki/Convolutional_neural_network#Convolutional_layers.

[11] Wikipedia. Activation functions, 2021. URL https://en.wikipedia.org/wiki/Activation_function.

[12] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[13] Wikipedia. Pooling layers, 2021. URL https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layers.

[14] Wikipedia. Fully connected layers, 2021. URL https://en.wikipedia.org/wiki/Convolutional_neural_network#Fully_connected_layer.

[15] Corinna Cortes, M. Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. In *UAI*, 2009.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[17] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122, 2018. doi: 10.1109/IIPHDW.2018.8388338.

[18] David C Plaut et al. Experiments on learning by back propagation., 1986.

[19] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8): 2, 2012.

[20] Bo-Kyeong Kim, Hwaran Lee, Jihyeon Roh, and Soo-Young Lee. Hierarchical committee of deep cnns with exponentially-weighted decision fusion for static facial

expression recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 427–434, 2015.

[21] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[22] Vitaly Bushaev. Understanding rmsprop - faster neural network learning, Sep 2018. URL https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Facebook AI Research lab. Pytorch, 2021. URL https://pytorch.org/.

[25] Google Brain Team. Tensorflow, 2021. URL https://www.tensorflow.org/.

[26] François Chollet. Keras, 2021. URL https://keras.io/.

[27] Nishir Mehta, Shahensha Shaik, R. Devireddy, and M. Gartia. Single-cell analysis using hyperspectral imaging modalities. *Journal of biomechanical engineering*, 140 2, 2018.

[28] Cecie Starr, Christine Evers, and Lisa Starr. *Biology: concepts and applications*. Nelson Education, 2010.

[29] Megandhren Govender, Kershani Chetty, and Hartley Bulcock. A review of hyperspectral remote sensing and its application in vegetation and water resource studies. *Water Sa*, 33(2):145–151, 2007.

[30] Gaspar Cano, Jose Garcia-Rodriguez, Alberto Garcia-Garcia, Horacio Perez-Sanchez, Jón Atli Benediktsson, Anil Thapa, and Alastair Barr. Automatic selection of molecular descriptors using random forest: Application to drug discovery. *Expert Systems with Applications*, 72:151–159, 2017.

[31] Yao-Ze Feng and Da-Wen Sun. Application of hyperspectral imaging in food safety inspection and control: a review. *Critical reviews in food science and nutrition*, 52 (11):1039–1058, 2012.

[32] Yao-Ze Feng and Da-Wen Sun. Application of hyperspectral imaging in food safety inspection and control: a review. *Critical reviews in food science and nutrition*, 52 (11):1039–1058, 2012.

[33] Yibing Hou, Zhong Ren, Guodong Liu, Lvming Zeng, and Zhen Huang. Design of a novel ld-induced hyper-spectral imager for breast cancer diagnosis based on vht

grating. In *2011 Symposium on Photonics and Optoelectronics (SOPO)*, pages 1–4. IEEE, 2011.

[34] David T Dicker, Jeremy Lerner, Pat Van Belle, DuPont Guerry, 4th, Meenhard Herlyn, David E Elder, and Wafik S El-Deiry. Differentiation of normal skin and melanoma using high resolution hyperspectral imaging. *Cancer biology & therapy*, 5(8):1033–1038, 2006.

[35] Baowei Fei, Hamed Akbari, and Luma V Halig. Hyperspectral imaging and spectral-spatial classification for cancer detection. In *2012 5th International Conference on BioMedical Engineering and Informatics*, pages 62–64. IEEE, 2012.

[36] Liu Zhi, David Zhang, Jing-qi Yan, Qing-Li Li, and Qun-lin Tang. Classification of hyperspectral medical tongue images for tongue diagnosis. *Computerized Medical Imaging and Graphics*, 31(8):672–678, 2007.

[37] Nader Sanai and Mitchel S Berger. Glioma extent of resection and its impact on patient outcome. *Neurosurgery*, 62(4):753–766, 2008.

[38] Himar Fabelo, Samuel Ortega, Raquel Lazcano, Daniel Madroñal, Gustavo M Callicó, Eduardo Juárez, Rubén Salvador, Diederik Bulters, Harry Bulstrode, Adam Szolna, et al. An intraoperative visualization system using hyperspectral imaging to aid in brain tumor delineation. *Sensors*, 18(2):430, 2018.

[39] Microsoft. Microsoft azure, 2021. URL https://azure.microsoft.com/.

[40] Synergy Research Group. Incremental growth in cloud spending hits a new high while amazon and microsoft maintain a clear lead, 2020. URL https://www.srgresearch.com/articles/incremental-growth-cloud-spending-hits-new-high-while-amazon-and-microsoft-maintain-clear-lead-reno-nv-february-4-2020.

[41] Microsoft. What is azure machine learning?, 2021. URL https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-ml.

[42] Microsoft. Azure sdk for python, 2021. URL https://azure.github.io/azure-sdk-for-python/.

[43] Inc. Docker. Docker overview, 2021. URL https://docs.docker.com/get-started/overview/.

[44] Microsoft. Azure container instances, 2021. URL https://azure.microsoft.com/en-us/services/container-instances/.

[45] Microsoft. Azure kubernetes service, 2021. URL https://docs.microsoft.com/en-us/azure/aks/.

[46] XIMEA GmbH. Mq022hg-im-sm5x5-nir - ximea. Available at https://www.ximea.com/en/products/hyperspectral-cameras-based-on-usb3-xispec/mq022hg-im-sm5x5-nir (2020/07/04).

[47] Adam Hayes. Stratified random sampling, May 2021. URL https://www.investopedia.com/terms/stratified_random_sampling.asp.

[48] Wikipedia. Cross-validation (statistics), Jun 2021. URL https://en.wikipedia.org/wiki/Cross-validation_(statistics).

[49] scikit. 3.1. cross-validation: evaluating estimator performance, 2021. URL https://scikit-learn.org/stable/modules/cross_validation.html.

[50] Jason Brownlee. A gentle introduction to k-fold cross-validation, Aug 2020. URL https://machinelearningmastery.com/k-fold-cross-validation/.

[51] PyTorch. Crossentropyloss, 2021. URL https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html?highlight=cross+entropy#torch.nn.CrossEntropyLoss.

[52] vikancha MSFT. Nc-series - azure virtual machines, 2020. URL https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series.

[53] Microsoft Azure. Real-time scoring of machine learning models - azure reference architectures, 2020. URL https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/ai/real-time-scoring-machine-learning-models.