UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación. Facultad de Ciencias.

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Segmentation of prostate cancer in T2W MRI images using deep learning models

Presentado por:
Pilar Navarro Ramírez

Tutor:
Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Julián Luengo Martín
Departamento de Ciencias de la Computación e Inteligencia Artificial

Curso académico 2022-2023

Segmentation of prostate cancer in T2W MRI images using deep learning models

Pilar Navarro Ramírez

Pilar Navarro Ramírez *Segmentation of prostate cancer in T2W MRI images using deep learning models*.

Trabajo de fin de Grado. Curso académico 2022-2023.

Responsable de tutorización

Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Julián Luengo Martín
Departamento de Ciencias de la Computación e Inteligencia Artificial

Doble Grado en Ingeniería Informática y Matemáticas

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación.
Facultad de Ciencias.

Universidad de Granada

Declaración de originalidad

D./Dña. Pilar Navarro Ramírez

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2022-2023, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 31 de agosto, 2022

Fdo: Pilar Navarro Ramírez

To my parents and fellow students

Contents

Abstract

Prostate Cancer (PCa) is the most frequently diagnosed type of cancer in males in most countries around the world, including Spain, and the fifth in the number of deaths (2020). Early detection of prostate cancer could reduce the mortality rate and make the disease treatable. In the last years, Magnetic Resonance Imaging (MRI) has shown excellent results in early diagnosis, monitoring and treatment planning for prostate cancer. In fact, international guidelines recommend performing a mpMRI (multiparametric magnetic resonance imaging) of the prostate to identify clinically significant PCa lesions, prior to any biopsy. However, interpreting these images is a complex and time-consuming task even for experienced radiologists. Additionally, it is subject to the observer's experience level, as well as human mistakes, and, as a result, it suffers from inter-observer and intra-observer variability.

Computer-aided diagnosis (CAD) systems that use machine learning and deep learning approaches have been shown to improve diagnostic accuracy and reduce inter-observer variability. In particular, the automatic segmentation of suspicious PCa lesions can provide a significant reference to the radiologists, allowing for a more focused interpretation, reducing the time needed, the efforts required by the radiologists, the diagnostic errors and the dependence on expert knowledge.

In the theoretical part of this work, we study in detail the fundamentals of the learning problem. More precisely, we focus on the statistical learning theory, which is based on probability theory and optimization. Additionally, we study the fundamentals of the particular case of deep learning, together with the two main models needed for our work: feedforward neural networks and convolutional neural networks.

In the most practical part, we developed a deep learning based software for fully automatic segmentation of prostate cancer lesions, both clinically significant and not clinically significant, in T2W MRI images, that we called **AutoPCaSeg**[1]. Several deep learning models based on the U-Resnet architecture were trained for different purposes using prostate T2W MRI images, previously preprocessed to homogenize the differences between images, and augmented to reduce overfitting.

All our models have the joint objective of segmenting PCa lesions in the T2W MRI images given as input. To this end, two strategies were followed. The first strategy divides the task of segmenting PCa lesions into three subtasks of increasing complexity: first, a model is trained to segment the entire prostate; the information learned is transferred to another model, which is further trained to segment the prostate zones; then, the learned weights are given to a third model, which learns to segment PCa lesions together with prostatic zones. In the second strategy, a new model is trained from scratch to segment only PCa lesions, using as input both preprocessed T2W images and prostate zonal probabilistic segmentations predicted by the model trained in the first strategy. The segmentation masks predicted by the models were further processed in a last step to improve the segmentation results, using prior information about the task at hand.

To improve the generalization ability of our models to images obtained with different MRI scanners and annotated by different radiologists, several public databases have been used for

[1]This software is publicly available at https://github.com/pilarnavarro/AutoPCaSeg.

training and evaluation: ProstateX, I2CVB, Prostate158, PROMISE12, NCI-ISBI and Medical Segmentation Decathlon.

This is the first work that makes use of all the public databases containing the segmentation masks of PCa lesions to train and evaluate the deep learning models. Additionally, we propose the first deep learning system to segment not clinically significant PCa lesions using a single MRI modality.

Our models achieved state-of-the-art results for the segmentation of the entire prostate gland and the prostatic zones, while the segmentation performance for the PCa lesions is slightly below the state-of-the-art. Nevertheless, we have to bear in mind that recent studies trained their models with different MRI modalities, but we used a single modality. Moreover, in most cases, only one dataset is used for training and evaluation, so the optimistic performance reported may be dataset-specific with no guarantee of good generalization ability. Anyway, a fair comparison with previous studies is challenging, especially for this task, since the data used for training and evaluation, as well as the evaluation metrics, differ between studies.

Keywords: probability, optimization, statistical learning, deep learning, convolutional neural networks, U-Net, Magnetic Resonance Imaging, semantic segmentation, prostate, prostate cancer, computer-aided diagnosis.

Resumen extendido

El cáncer de próstata (CaP) es el tipo de cáncer diagnosticado con más frecuencia entre los hombres en la mayoría de países alrededor del mundo, incluido España, y es el quinto en número de muertes. Más de 1.4 millones de nuevos casos fueron diagnosticados en 2020 con 375.000 muertes alrededor del mundo. Una detección temprana del cáncer de próstata puede hacer que la enfermedad se pueda tratar y disminuir la tasa de mortalidad. En los últimos años, el uso de imagen por resonancia magnética (IRM) ha demostrado ser muy eficaz para diagnosticar el cáncer en una etapa temprana, vigilar la evolución del mismo y planear un tratamiento adecuado en cada caso. De hecho, es el tipo de imagen recomendado internacionalmente para detectar tumores prostáticos antes de realizar una biopsia.

La interpretación de imágenes de próstata de resonancia magnética requiere de un radiólogo experto y es una tarea compleja en la que es necesario invertir mucho tiempo. Además, la precisión de los resultados depende de la experiencia del radiólogo, y la interpretación está sujeta a errores debidos a las limitaciones humanas, tales como el cansancio mental, una percepción visual limitada o distracciones, y a la complejidad de la propia tarea. Como consecuencia, existen variaciones significativas entre interpretaciones de varios radiólogos y entre interpretaciones realizadas por el mismo radiólogo.

El diagnóstico asistido por ordenador (DAO) se define como *el uso de algoritmos computacionales para ayudar a los médicos en el proceso de interpretación de imágenes médicas*. Se ha demostrado que los sistemas DAO ayudan a mejorar la precisión de los diagnósticos de los radiólogos, a la vez que reducen el tiempo y esfuerzo requerido para la interpretación de imágenes, así como los errores y las diferencias entre interpretaciones de distintos radiólogos. En particular, la segmentación automática de tumores en la próstata, que básicamente consiste en definir los bordes de las lesiones, puede suponer una referencia significativa para los radiólogos, de manera que al centrarse en la interpretación de las regiones señaladas mejora la eficiencia y precisión del diagnóstico realizado. Por otro lado, también permite disminuir el tiempo empleado en la detección de lesiones sospechosas y la dependencia de la interpretación con la experiencia del radiólogo.

Los primeros sistemas DAO se basaban en la extracción manual de características de las imágenes médicas y en su clasificación por medio de cualquier algoritmo clásico de aprendizaje automático (máquinas de soporte vectorial, regresión logística, random forest, etc.). Desde el auge del aprendizaje profundo en 2012, las redes neuronales convoluciones han dominado todas las aplicaciones de análisis de imágenes, incluidas las imágenes médicas. A diferencia de los métodos clásicos de aprendizaje automático, con este tipo de redes neuronales ya no es necesario un paso previo de extracción de características, para el cual se necesita tener conocimiento médico de la tarea en cuestión, sino que es el propio algoritmo el que se encarga de aprender y extraer el conjunto óptimo de características de las imágenes proporcionadas durante su entrenamiento.

Varios estudios previos han usado redes neuronales convolucionales para detectar lesiones malignas en la próstata, determinar si una lesión específica es agresiva o benigna, y para segmentar la próstata y sus zonas. Sin embargo, muy pocos trabajos han investigado la aplicación de redes neuronales convolucionales a la tarea de segmentar tumores prostáticos,

lo cual es un problema bastante más complejo de abordar, y los pocos trabajos existentes son muy recientes. Nuestro trabajo contribuirá a llenar este vacío en la medida de lo posible.

En este trabajo hemos desarrollado una aplicación a la que hemos llamado **AutoPCaSeg**[1] para segmentar de manera completamente automática, usando modelos de aprendizaje profundo, lesiones prostáticas consideradas malignas y benignas en imágenes por resonancia magnética. Es decir, dada una imagen, nuestra aplicación determinará para cada píxel si este se corresponde con una lesión maligna o si se trata de tejido prostático sano, y, en tal caso, incluso identificará a qué parte de la próstata pertenece dicho tejido o si se corresponde con una lesión benigna.

En la parte teórica del trabajo estudiaremos en profundidad los fundamentos del problema de aprendizaje. En particular, nos centraremos en el paradigma del aprendizaje estadístico clásico, el cual se basa en la teoría de la probabilidad y la optimización. Además, estudiaremos las bases del caso particular del aprendizaje profundo, junto con los principales modelos necesarios para nuestro trabajo: las redes neuronales prealimentadas y las redes neuronales convolucionales.

Fundamentos matemáticos

Para entender los principales resultados de la teoría del aprendizaje estadístico es necesario realizar una pequeña introducción a la **teoría de la probabilidad**. Esta, a su vez, está estrechamente relacionada con la teoría de la medida, por lo que la parte matemática de nuestro trabajo comienza con los principales conceptos de esta teoría, definiendo lo que es una medida (una medida de probabilidad es un caso particular de medida) y aplicación medible, junto con sus propiedades. También se estudiarán aquí algunos resultados sobre integración, que serán necesarios para el cálculo de la esperanza de una variable aleatoria. Supondremos conocida la noción de integral con respecto a una medida y su construcción, por lo que nos centraremos simplemente en presentar los resultados necesarios. Una vez explicadas las bases de la teoría de la medida, pasaremos a la parte central de la teoría de la probabilidad: las variables aleatorias y distribuciones de probabilidad asociadas a estas. Una variable aleatoria no es más que una aplicación medible, mientras que su distribución de probabilidad se obtiene como composición de su inversa y una medida de probabilidad. Se introducirán las nociones de función de distribución, función masa de probabilidad y función de densidad además de sus principales propiedades. Así mismo, hablaremos de la independencia entre variables aleatorias, de la probabilidad condicionada y de los momentos de una variable aleatoria, prestando especial atención en este último caso a la esperanza y a su cálculo en función de la distribución de probabilidad. Finalmente, se definirá la convergencia en probabilidad, un concepto clave en la teoría de la generalización del aprendizaje estadístico.

Explicado todo lo anterior, nos centraremos entonces en la **teoría del aprendizaje estadístico**. Empezaremos definiendo el problema del aprendizaje y sus principales componentes y presentaremos el paradigma de aprendizaje usado para resolver este problema, conocido como *minimización del riesgo empírico (ERM)*. Este consiste básicamente en minimizar el error del algoritmo de aprendizaje sobre una muestra dada. A continuación, explicaremos el concepto de *aprendizaje PAC*, por el cual se requiere que un algoritmo de aprendizaje aprenda una función con una cierta probabilidad y un determinado margen de error.

[1] El software desarrollado se puede encontrar en el repositorio de Github https://github.com/pilarnavarro/AutoPCaSeg.

Después pasaremos a enunciar y explicar los principales resultados de la teoría de la generalización siguiendo su formulación original propuesta por Vladimir Vapnik [1]. Para ello, en primer lugar hablaremos de la teoría de la consistencia y se enunciará un teorema (el *teorema fundamental del aprendizaje estadístico*) que determina las condiciones bajo las cuales la regla de minimización del riesgo empírico es consistente. También se verá entonces la relación entre el aprendizaje PAC y la consistencia. En segundo lugar, estudiaremos cotas para el error de generalización centrándonos en el caso de la clasificación binaria. Es decir, se acotará la diferencia entre el error obtenido por el algoritmo de aprendizaje en la muestra de entrenamiento dada y el error real. Este análisis se dividirá en dos casos, según si el conjunto de funciones hipótesis es finito o no. En el segundo caso surge el concepto de *dimensión de Vapnik-Chervonenkis* (dimensión VC), un valor entero que permite medir la complejidad de una clase de hipótesis, y, asociada a esta, la *desigualdad de Vapnik-Chervonenkis*, la cual supone uno de los resultados más importantes de la teoría del aprendizaje. Concluiremos la sección sobre teoría de la generalización con el *teorema fundamental del aprendizaje PAC*, el cual relacionará casi todos los conceptos mencionados hasta ese momento. Este teorema asegura, entre otras cosas, que una clase de hipótesis es PAC aprendible si, y sólo si, su dimensión VC es finita.

Finalmente, para terminar el capítulo sobre teoría del aprendizaje, se presentarán algunos retos subyacentes a todo problema de aprendizaje, como son el compromiso entre complejidad y generalización (*approximation-generalization tradeoff*) de una clase de hipótesis o el sobreajuste (*overfitting*), junto con algunas posibles soluciones (por ejemplo, para el sobreajuste la regularización).

Para minimizar el error del algoritmo de aprendizaje sobre la muestra de entrenamiento se necesitan herramientas de **optimización**. Por tanto, enunciaremos el problema de optimización convexa sin restricciones (en particular el problema de minimización), tras definir el concepto de función convexa y algunas de sus propiedades. Aunque las funciones a minimizar en el marco del aprendizaje no son casi nunca convexas, la optimización convexa supone la base para definir algoritmos que resuelvan los problemas de optimización no convexa y es por ello por lo que nos centraremos en la optimización convexa. Presentaremos *el algoritmo de gradiente descendente*, en el cual se basan la mayoría de algoritmos de optimización usados en el aprendizaje, y que consiste en actualizar iterativamente el óptimo alcanzado hasta ese momento según la dirección opuesta a la indicada por el gradiente de la función a optimizar en ese punto. Demostraremos que, para funciones convexas, diferenciables y con gradiente Lipschitziano, está garantizado que el algoritmo de gradiente descendente encuentra el mínimo de la función. Además, veremos cómo aplicar este algoritmo para resolver el problema del aprendizaje, así como algunas de sus variantes comúnmente usadas por los algoritmos de aprendizaje automático. Entre ellas, destacamos el algoritmo *Adam*, que es el que nosotros usaremos para entrenar nuestros modelos.

Aprendizaje Profundo

En la segunda parte de este trabajo explicaremos los fundamentos del aprendizaje profundo, o *deep learning* en inglés. El aprendizaje profundo es la rama del aprendizaje automático que se centra en estudiar las **redes neuronales artificiales**, las cuales constituyen un modelo muy flexible con un gran potencial de aproximación, tal y como demuestra el *teorema universal de aproximación* de las redes neuronales, enunciado y demostrado por primera vez por G.Cybenko [2]. En efecto, este teorema afirma que cualquier función continua en un conjunto

compacto se puede aproximar por una red neuronal prealimentada. Empezaremos esta parte presentando las redes neuronales prealimentadas, que no son más que una colección de neuronas organizadas en capas. Estas últimas aplican una operación lineal a sus entradas, seguida de una aplicación no lineal conocida como función de activación. Fijaremos la notación a utilizar y definiremos la clase de funciones implementada por estas redes neuronales. Después se describirá el proceso por el cual se entrena este modelo, o, lo que es lo mismo, el proceso por el cual se minimiza su error sobre un conjunto de entrenamiento. Para ello, se describirán dos algoritmos: el algoritmo de propagación hacia delante, que calcula la función implementada por una red neuronal, y el algoritmo de propagación hacia atrás, o *backpropagation*. Este último permite calcular el gradiente de la función de pérdida para usarlo en el algoritmo de gradiente descendente, y se basa en aplicar repetidamente la regla de la cadena. Presentaremos también las funciones de activación más comunes, junto con algunas de sus ventajas e inconvenientes, y un par de técnicas de regularización típicas en las redes neuronales. Para cerrar este capítulo se enuncian algunos resultados que demuestran el gran poder de aproximación de las redes neuronales, entre ellos el teorema de aproximación universal ya mencionado, y se estudia su dimensión VC en función de la función de activación usada por las neuronas de la red, centrándonos en el caso de la clasificación binaria.

Continuaremos esta parte con un tipo particular de redes neuronales prealimentadas, las **redes neuronales convolucionales**, cuyo principal objetivo es procesar señales discretas, como sonidos o imágenes. Estas redes se caracterizan por usar una operación de convolución discreta o, mejor dicho, *correlación cruzada discreta* en algunas de sus capas. Estas capas reciben el nombre de capas convolucionales. Así, pues, definiremos la operación de convolución y correlación cruzada tanto en el caso continuo como el discreto, señalando las diferenias entre ambas operaciones. Presentaremos las capas de convolución junto con algunas de sus propiedades, como la conectividad local y la compartición de parámetros, y los hiperparámetros a especificar para este tipo de capas. Las capas convolucionales suelen ir seguidas por otro tipo de capas, como las *capas de poooling* y *normalización*, de manera que también explicaremos el funcionamiento de estas capas, así como algunos de sus distintos tipos y diferencias entre ellos. Además, como las imágenes IRM con las que vamos a trabajar son tridimensionales, comentaremos la generalización de todas estas capas al caso tridimensional. Después explicaremos lo que se conoce (erróneamente) como la "inversa" de la convolución, llamada convolución traspuesta, que se usa en la arquitectura U-Net, por la cual la salida de una operación de convolución se devuelve al tamaño original que tenía antes de aplicar dicha operación. Finalmente, se detallará el funcionamiento de los *bloques residuales*, que surgen como solución al problema de degradación observado en redes neuronales convolucionales profundas.

Aplicación

Las redes neuronales convolucionales han demostrado resultados excelentes en la resolución de la mayoría de problemas de visión por computador, superando incluso a las personas expertas en muchos casos. En particular, este tipo de redes han resultado ser bastante adecuadas para la tarea de *segmentación semántica*, la cual consiste en etiquetar cada píxel de una imagen con una cierta clase, según lo que se muestre. En particular, la *arquitectura U-Net*, presentada en 2015 por Ronnenberger et al. [3], ha demostrado ser la arquitectura con el mejor desempeño para esta tarea en imágenes médicas. Dicha red está constituida por dos partes: un *codificador*, que se encarga de reducir la resolución de la imagen de entrada a la vez

que extrae las características más importantes de la misma, y un *decodificador*, que produce un mapa de segmentación (en el que cada píxel tiene asociada la probabilidad de pertenecer a cada clase) con la misma resolución de la imagen de entrada mediante el aumento sucesivo de la resolución de la salida producida por el codificador. La U-Net fue propuesta originalmente para tratar con imágenes 2D pero, puesto que la mayoría de imágenes biomédicas tienen forma de volumen, posteriormente se adaptó para el caso 3D. Además, más adelante se fue probando a usar distintos tipos de bloques convolucionales en su estructura, entre ellos los bloques residuales. Así, surgió la *U-Resnet*, la cual describiremos en detalle. Para el problema de segmentación semántica las dos funciones de pérdida más usadas son la pérdida Dice y la entropía cruzada. Presentaremos ambas funciones, junto con su expresión matemática, en el capítulo sobre segmentación.

Con el objetivo de segmentar lesiones de cancer de próstata en imágenes IRM entrenaremos varios modelos que usan la arquitectura U-Resnet en 3D, para explotar la información espacial contextual en todas las dimensiones. Para entrenar y evaluar los modelos usaremos todos los conjuntos de datos actualmente disponibles públicamente de imágenes de próstata IRM que contienen las máscaras de segmentación de los tumores (ProstateX, I2CVB y Prostate158), así como otras bases de datos con solo las máscaras de segmentación correspondientes a las zonas de la próstata (Medical Segmentation Decathlon y NCI-ISBI) y solo la segmentación de la próstata completa (PROMISE12). Con ello, pretendemos mejorar la capacidad de generalización de nuestros modelos a imágenes IRM obtenidas con escáneres y protocolos diferentes y anotadas (segmentadas) por radiólogos con distinta formación. No obstante, esto puede dificultar el proceso de aprendizaje de los modelos y dar lugar a segmentaciones de peor calidad, pero nos aseguramos de que estos presentarán aproximadamente el mismo desempeño en conjuntos de datos muy distintos. Por lo que sabemos, presentamos el primer trabajo que emplea todas las bases de datos públicas disponibles para esta tarea.

Se suelen usar diferentes modalidades de imagen por resonancia magnética para realizar un diagnóstico de cáncer de próstata (T2W, DWI, mapa ADC, DCE), que en conjunto forman lo que se conoce como imagen por resonancia magnética multiparamétrica. El uso de varias de estas modalidades en conjunto para diagnosticar CaP ofrece mejores resultados que el diagnóstico con una sola de las modalidades, por lo que la mayoría de sistemas de diagnóstico asistido por ordenador previamente diseñados utilizan una combinación de diferentes modalidades. Sin embargo, el tiempo de adquisición de varias modalidades es bastante superior al tiempo requerido para obtener una sola de las modalidades. Además, los ligeros movimientos del paciente producidos entre la adquisición de distintas modalidades, hacen que las imágenes de estas diferentes modalidades no estén alineadas, por lo que es necesario realizar un proceso posterior de alineación para poder usarlas para el entrenamiento de modelos de aprendizaje automático. Este proceso de alineación necesita que las imágenes de todas las modalidades sean anotadas una a una, lo cual es una tarea tediosa que necesita de mucho tiempo. Así pues, en este trabajo analizaremos los resultados obtenidos usando solo la modalidad T2W de IRM, que es la de mayor resolución, para entrenar nuestros modelos, aún sabiendo que esto puede conllevar un empeoramiento del desempeño de los mismos. De hecho, hasta donde sabemos, este es el primer trabajo que trata de segmentar tanto lesiones malignas como benignas usando solo una modalidad de IRM.

Puesto que la apariencia y la probabilidad con la que se manifiestan los tumores prostáticos es diferente según la zona de la próstata en la que se encuentren, decidimos incluir información sobre las zonas prostáticas en nuestros modelos para mejorar su capacidad de detección de lesiones. Para ello seguiremos dos estrategias. La primera divide la tarea de segmentar las lesiones de cancer de próstata en tres subtareas de complejidad creciente: primero

entrenamos un modelo, P-Seg, para segmentar la próstata completa; la información aprendida por este se transfiere entonces a otro modelo, Z-Seg, que se entrena para segmentar las zonas prostáticas; después usamos los pesos aprendidos por Z-Seg para inicializar un tercer modelo, PCa-Seg modelo 1, que aprende a segmentar las lesiones y las zonas prostáticas. Con la segunda estrategia entrenamos un modelo nuevo, PCa-Seg modelo 2, desde el principio para segmentar solo las lesiones, usando como entrada las imágenes T2W y los mapas de segmentación probabilísticos con información de las zonas de la próstata producidos por Z-Seg. Entrenaremos dos veces tanto PCa-Seg modelo 1 como PCa-Seg modelo 2: la primera vez para que aprendan a segmentar solo las lesiones consideradas agresivas, y la segunda para que aprendan a segmentar las lesiones malignas y las benignas.

Antes de usar los datos para entrenar los modelos, los preprocesamos para homogeneizar las diferencias entre las imágenes, y los aumentamos para paliar el sobreajuste. Por otro lado, también procesamos las salidas producidas por los modelos para mejorar la calidad de las segmentaciones (por ejemplo para disminuir el número de falsos positivos eliminaremos lesiones que se consideran demasiado pequeñas).

Tras evaluar nuestros modelos usando validación cruzada con tres pliegues y testearlos en varios datasets, llegamos a la conclusión de que las diferencias entre los resultados producidos por los distintos modelos en la segmentación de lesiones malignas no son lo suficientemente significativas como para tenerlas en cuenta. Los modelos P-Seg y Z-Seg obtienen resultados cercanos al estado del arte, mientras que el rendimiento de los modelos PCa-Seg queda un poco por debajo del de los trabajos más recientes. Sin embargo, debemos tener en cuenta que usamos una arquitectura muy simple, que sólo utilizamos una modalidad de IRM y que entrenamos nuestros modelos con conjuntos de datos muy hetereogéneos.

En concreto, durante la validación cruzada, P-Seg obtuvo un valor medio del coeficiente de dice (DSC) de 0.9049 para la segmentación de la glándula prostática completa, Z-Seg alcanzó un DSC de 0.8506 y 0.7283 para la glándula central (GC) y la zona periférica de la próstata (ZP), respectivamente, y el mejor DSC obtenido por los modelos PCa-Seg para la segmentación de lesiones de CaP clínicamente significativas fue de 0.16.

En los conjuntos de tests, los valores de DSC obtenidos por el modelo P-Seg oscilan entre 0.858 y 0.917, mientras que los valores de DSC obtenidos por Z-Seg están en el rango de 0.67 a 0.877 para la glándula central, y 0.637 a 0.7731 para la zona periférica. Para la segmentación de lesiones benignas en el conjunto de datos ProstateX, el PCa-Seg modelo 2 alcanzó un DSC de 0.118. En cuanto a las lesiones malignas, en los datos de test del conjunto de datos I2CVB adquirido con el escáner Siemens, el mejor valor DSC alcanzado por los modelos PCa-Seg fue de 0.496 y el mejor AUC 0.737, mientras que en las imágenes de test del mismo conjunto de datos pero adquiridas con el escáner General Electric, el DSC más alto obtenido fue de 0.223 y el AUC más alto de 0.724. En el conjunto de test de Prostate158, los mejores resultados fueron un DSC de 0.1679 y un AUC de 0.771, y para el conjunto de datos ProstateX, los valores más altos fueron 0.116 para el DSC y 0.897 para el AUC.

Palabras clave: probabilidad, optimización, aprendizaje estadístico, aprendizaje profundo, redes neuronales convolucionales, U-Net, imagen por resonancia magnética, segmentación semántica, próstata, cancer de próstata, diagnóstico asistido por ordenador.

Part I.

Introduction and objectives

1. Introduction

1.1. Contextualization

Prostate Cancer (PCa) is the most frequently diagnosed type of cancer in males in most countries around the world, including Spain, with more than 1.4 million new prostate cancer cases diagnosed in 2020 and 375,000 associated deaths worldwide. It is the second most frequent cancer among men, after lung cancer, and the fifth most common cause of cancer death (2020). Early detection of prostate cancer could reduce the mortality rate and make the disease treatable. Accurate diagnosis and localization of tumors in the prostate are critical for targeted biopsy, monitoring the disease, and guiding local treatments. In the last years, Magnetic Resonance Imaging (MRI) has shown excellent results in early diagnosis and tumor localization. International guidelines recommend performing a mpMRI (multiparametric magnetic resonance imaging) of the prostate to identify clinically significant PCa lesions, prior to any biopsy.

Interpreting prostate mpMRI is a complex and time-consuming task that requires an experienced radiologist. Additionally, the diagnosis accuracy of PCa depends very much on the experience of the radiologist, and the interpretation is subject to human mistakes. As a consequence, there exists non-negligible inter-observer and intra-observer variability.

Computer-aided diagnosis (CAD) systems are defined as *the use of computer algorithms to aid the image interpretation process*. CAD systems have been shown to improve the diagnostic accuracy of radiologists, while reducing the time and efforts required for image interpretation, as well as diagnostic errors and inter-observer variability. In particular, the automatic segmentation of PCa, which basically consists in determining the boundaries of the lesions, can provide a significant reference for radiologists so that by focusing on the highlighted regions, the efficiency and diagnostic accuracy improves. Moreover, it also allows for a reduction in the time spent on the detection of suspicious PCa lesions and the dependence of the interpretation on the radiologist's experience.

The first CAD systems were mostly based on manual extraction of important features from the medical images, followed by a classification process using any traditional machine learning algorithm (support vector machines, logistic regression, random forests, etc.). Since the breakthrough of deep learning in the area of computer vision in 2012, convolutional neural networks have dominated all types of image analysis applications, including medical image analysis, phasing out classical machine learning classification algorithms. Unlike the aforementioned machine learning methods, with this type of neural network, there is no need for an initial step that requires deep domain-specific knowledge and extraction of discriminant features by human researchers. Instead, it is the algorithm itself that learns the optimal set of features directly from the raw images provided during training.

Several previous studies have used convolutional neural networks to detect malignant lesions in the prostate, determine whether a specific lesion is aggressive or benign, as well as for the segmentation of the prostate anatomy and other types of lesions. However, fewer works have investigated the application of convolutional neural networks to the task of segmenting PCa lesions, which is a more complex task, and the few existing publications are

quite recent. This work contributes to filling this gap to the extent possible.

1.2. Description of the problem

In this work, we seek to automatically segment aggressive prostate cancer and indolent cancer on prostate MRI. Image segmentation is a computer vision task that aims at dividing an image into multiple regions or segments. In other words, every pixel in the image is labeled with a certain class according to what it shows. The U-Net architecture, introduced in 2015 by Ronnenberger et al. [3], has proven to be the best deep learning model for the segmentation of medical images. This network consists of two parts: an encoder, which is responsible for downsampling the spatial resolution of the input image while extracting the most important features from it, and a decoder, which generates a segmentation map (where each pixel contains the probability of belonging to each class) with the same resolution of the input image by successively upsampling the resolution of the output produced by the encoder. A modification of this network called U-Resnet, which uses residual blocks, and receives 3D data as input, has been employed in our work.

Different MRI modalities are commonly used for the diagnosis of prostate cancer (T2W, DWI, ADC map, DCE), which together constitute what is known as multiparameter MRI. The use of several of these modalities to diagnose PCa usually offers better results than diagnosis using a single modality, which is why most previously designed CAD systems use a combination of different modalities. However, the acquisition time of multiparametric MRI is notably longer than that of a single modality. In addition, the use of multiparametric MRI to train deep learning models requires a registration process of the different MRI modalities, since they are normally misaligned due to patient movement between the acquisition of different modalities and differences in resolution. This registration process requires manual annotations of a large number of images in all modalities, which is a tedious and time-consuming task. Therefore, in this work, we opt for the use of a single modality, although it may entail a worsening of the segmentation performance. Concretely, we use the T2W MRI modality, as it presents the highest spatial resolution.

We developed a deep learning based software, that we called **AutoPCaSeg**, for fully automatic segmentation of clinically significant and not clinically significant prostate cancer lesions in T2W MRI images. More precisely, several U-Resnet models were trained for different purposes (but with the joint objective of segmenting PCa lesions in the images given as input) using prostate T2W MRI images, previously preprocessed to homogenize the differences between images, and augmented to reduce overfitting.

Prostate cancer lesions vary in frequency, malignancy and appearance in different prostate zones. Thus, to increase lesion detection performance, we decided to include zonal information in the models responsible for segmenting PCa lesions. To this end, two strategies were followed:

- The first strategy divides the task of segmenting PCa lesions into three subtasks of increasing complexity: first, a model is trained to segment the entire prostate; the information learned is transferred to another model, which is further trained to segment the prostate zones; then, the learned weights are given to a third model, which learns to segment PCa lesions together with prostatic zones.
- In the second strategy, a new model is trained from scratch to segment only PCa lesions, using as input both preprocessed T2W images and prostate zonal probabilistic

segmentations predicted by the model trained in the first strategy.

In MRI images, the voxel intensities, spatial resolution and other appearance characteristics can greatly differ between acquisition protocols and scanners. Therefore, segmentation models designed for use in clinical practice need to deal with these issues. If the deep learning segmentation model was trained using data from only one MRI scanner, its generalization ability to data from different scanner manufacturers could be limited. Additionally, training the model with segmentation masks provided by a single radiologist could introduce personal biases. As a result, the performance of the model when compared with other radiologists' segmentations, especially from other institutions, and, thus, with different formations, could be affected. Consequently, we decided to use different datasets publicly available for training and validating our models. Concretely, we used the following datasets: ProstateX, I2CVB, Prostate158, PROMISE12, NCI-ISBI, and Medical Segmentation Decathlon.

1.3. Work structure

The present work is divided into five parts, which, in turn, are composed of several chapters.

1. Part I consists of two chapters, where a brief introduction to the developed work is presented and the main objectives that it should accomplish are listed.

2. In the second part, all the mathematical fundamentals needed to understand the rest of this work are explained. Concretely, in chapter 3 we review some fundamental concepts of probability theory and the main results are stated. In chapter 4, we develop the theory underlying the learning problem. The classical definition of a learning problem is introduced and the main results of the theory of generalization are explained, as well as some challenges that they pose. For its part, chapter 5 describes the convex optimization problem and the main optimization algorithms used to solve the learning problem.

3. Part III is devoted to the study of artificial neural networks. In chapter 6 we define formally the model that feedforward neural networks constitute, their training procedure and some of their characteristics. In chapter 7 convolutional neural networks are described, together with the most common types of layers they include.

4. In the fourth part we present the main problem this work tries to solve and the solution reached. More precisely, chapter 8 is focused on the semantic segmentation problem, where this task is described along with the most common deep learning architecture used to solve it and the two most popular loss functions. In chapter 9 all the particularities of the problem of segmenting PCa lesions in MRI are explained, including a survey of the most recent publications in the field and the databases we employ. The methodology that we followed to solve this problem is described in chapter 10. Finally, in chapter 11 we describe the experimental framework and present the results obtained by our software.

5. The last part contains a unique chapter where possible future research lines and the conclusions reached are exposed.

1.4. Main references

A wide variety of books and articles have been used as an aid to develop the present work. The most important references are listed below.

- ☐ *Probability Theory: A Comprehensive Course* [4], written by Achim Klenke, for the chapter about probability theory chapter 3.
- ☐ *Statistical Learning Theory* [1], by Vladimir Vapnik, where the theory about statistical learning is explained in a formal way. It has been used for the development of chapter 4.
- ☐ *Understanding Machine Learning: From Theory to Algorithms* [5], written by Shai Shalev-Schwarts and Shai Ben-David, and *Learning from Data, a short course* [6], by Y. S. Abu-Mostafa, for chapter 4 and chapter 6.
- ☐ *Convex Optimization* [7], is the main reference followed for the chapter about optimization chapter 5.
- ☐ *Deep learning* [8], by Ian Goodfellow, Yoshua Bengio and Aaron Courville, has been used for the chapter about artificial neural networks chapter 6 and the chapter of convolutional neural networks chapter 7.
- ☐ *Redes Neuronales* & *Deep Learning*, by Fernando Berzal, has been followed to write some parts of chapter 6 and chapter 7.
- ☐ For the development of the application, we got inspired mainly by the studies of
  - Alkadi et al. [9], *A deep learning-based approach for the detection and localization of prostate cancer in T2 magnetic resonance images*,
  - Pellicer-Valero et al. [10], *Deep Learning for fully automatic detection, segmentation, and Gleason Grade estimation of prostate cancer in multiparametric Magnetic Resonance Images*,
  - Mehta et al. [11], *AutoProstate: Towards Automated Reporting of Prostate MRI for Prostate Cancer Assessment using Deep Learning*,
  - Adams et al. [12], *Prostate158 - An expert-annotated 3T MRI dataset and algorithm for prostate cancer detection*.

2. Objectives

The main objectives that this work shall fulfill are listed below.

1. **Mathematics:** the main goal is to describe the mathematical fundamentals of deep learning models. For this purpose, we shall
   (a) Study the main concepts of probability theory.
   (b) Understand the fundamentals of statistical learning theory.
   (c) Study optimization algorithms to solve the learning problem.
   (d) Analyze the fundamentals of artificial neural networks and, in particular, of convolutional neural networks.
2. **Informatics:** the primary objective is to implement a deep learning based software to automatically segment prostate cancer lesions in T2W MRI images and evaluate it on several datasets. To this end, we have to achieve the following points:
   (a) Describe the main deep learning model used for the task of semantic segmentation in medical images, the U-Net architecture.
   (b) Review the state-of-the-art for the task of segmenting prostate cancer lesions in MRI images.
   (c) Develop a software based on deep learning to automatically segment prostate cancer lesions in MRI images.
   (d) Evaluate the segmentation performance of the software by testing it on different public databases.

All these objectives have been successfully achieved throughout the development of the present work:

- 1.(a) is covered in chapter 3.
- 1.(b) is studied in chapter 4.
- 1.(c) is fulfilled with chapter 5, where the gradient descent algorithm and its variants are explained.
- Objective 1.(d) is covered in chapter 6, where the fundamentals of artificial neural networks, and, in particular, feedforward neural networks, are explained, as well as in chapter 7, where convolutional neural networks are introduced.
- 2.(a) is studied in chapter 8, section 8.2. Here, the U-Net architecture is described in detail, as well as some of its variants that we employed.
- Concerning 2.(b), the literature in the task at hand is reviewed in chapter 9, section 9.6.
- Objective 2.(c) is fulfilled with the implementation of AutoPCaSeg, which is described in chapter 10.

- Finally, objective 2.(d) is also attained as explained in chapter 11, where the experimental framework used is described and the results obtained by our software on different test datasets are analyzed.

In the following table we present the main subjects studied during the double Bachelor degree in Mathematics and Computer Science that have most influenced this work:

Mathematics	**Computer Science**
Probability	Artificial Intelligence
Statistical Inference	Computer Vision
Mathematical Analysis	Data Structures
Lineal Algebra	Programming
Topology	Business Intelligence
Numerical Methods	Machine Learning

Table 2.1.: Main subjects related to our work.

Part II.

Mathematics

3. Probability Theory

This chapter presents some fundamental concepts of probability theory that will be needed to understand the rest of this work, and, specially, the part regarding learning theory. Together with the indispensable notions, additional results are included to make this chapter as complete as possible.
The references for this part are [13] and [4], where the proofs of all the results presented here can be found.

3.1. Fundamentals of Measure Theory

Probability theory goes hand in hand with measure theory. As a consequence, we must introduce the fundamental concepts of measure theory before going deeper into probability.

Let us consider a nonempty set Ω and let 2^Ω be the power set of Ω (set of all subsets of Ω). We define a σ-algebra on the set Ω as follows:

Definition 3.1. (σ-algebra) A class of sets $\mathcal{A} \subset 2^\Omega$ is a σ-algebra if it satisfies the following conditions:

1. $\Omega \in \mathcal{A}$.
2. $\mathcal{A}$ is closed under complements, i.e, $A^c := \Omega \setminus A \in \mathcal{A}$ for all $A \in \mathcal{A}$.
3. $\mathcal{A}$ is closed under countable[1] unions, i.e, $\bigcup_{n=1}^{\infty} A_n \in \mathcal{A}$ for any collection of countably many subsets $\{A_n\}_{n\in\mathbb{N}} \in \mathcal{A}$.

Note that, if $\mathcal{A}$ is a σ-algebra, then $\emptyset = \Omega \setminus \Omega$ is in $\mathcal{A}$.
A tuple $(\Omega, \mathcal{A})$ where Ω is a nonempty set and $\mathcal{A} \subset 2^\Omega$ is a σ-algebra is known as a *measurable space*. The sets $A \in \mathcal{A}$ are called *measurable sets*. The measurable space $(\Omega, 2^\Omega)$ is called *discrete*, as long as Ω is at most countably infinite.

Example 3.1. (σ-algebras)

- The trivial examples of σ-algebras are the classes $\mathcal{A} = \{\emptyset, \Omega\}$ and $\mathcal{A} = 2^\Omega$.
- The intersection of σ-algebras is a σ-algebra.
- The restriction of a σ-algebra $\mathcal{A} \subset 2^\Omega$ to a nonempty subset $A \subset \Omega$, defined by $\mathcal{A}\big|_A := \{A \cap B : B \in \mathcal{A}\} \subset 2^A$, is a σ-algebra on A.

Theorem 3.1. *(*Generated σ-algebra*) Let $\mathcal{E} \subset 2^\Omega$ be a class of sets. Then there exists a smallest σ-algebra $\sigma(\mathcal{E})$ with $\mathcal{E} \subset \sigma(\mathcal{E})$ and it is:*

$$\sigma(\mathcal{E}) := \bigcap_{\substack{\mathcal{A}\subset 2^\Omega \text{is a } \sigma\text{-algebra} \\ \mathcal{A}\supset\mathcal{E}}} \mathcal{A}.$$

[1] By countable we mean finite or countably infinite.

$\sigma(\mathcal{E})$ is called the σ-algebra generated by $\mathcal{E}$ and $\mathcal{E}$ is called a generator *of the σ-algebra.*

Proposition 3.1. *Under the conditions of the previous theorem, the following statements are true:*

1. $\mathcal{E} \subset \sigma(\mathcal{E})$.
2. *If* $\mathcal{E}_1 \subset \mathcal{E}_2$, *then* $\sigma(\mathcal{E}_1) \subset \sigma(\mathcal{E}_2)$.
3. $\mathcal{A}$ *is a σ-algebra if, and only if,* $\sigma(\mathcal{A}) = \mathcal{A}$

Definition 3.2. (Topology) Let $\Omega \neq \emptyset$ be an arbitrary set, and $\tau \subset 2^\Omega$ a class of sets. Then τ is a *topology* if it fulfills the following conditions:

1. $\Omega, \emptyset \in \tau$.
2. $\mathcal{A}$ is closed under intersections, i.e, $A \cap B \in \tau$ for all $A, B \in \tau$.
3. $\mathcal{A}$ is closed under arbitrary unions, i.e, $(\bigcup_{A \in \mathcal{F}} A) \in \tau$ for all $\mathcal{F} \subset \tau$.

The tuple (Ω, τ) is called a *topological space* and the sets $A \in \tau$ are called *open*. The sets $A \subset \Omega$ such that $A^c \in \tau$ are known as *closed*.

Example 3.2. (Borel σ-algebra) On any topological space (Ω, τ) we can define the σ-algebra generated by the open sets $A \in \tau$, $\mathcal{B}(\Omega) := \mathcal{B}(\Omega, \tau) := \sigma(\tau)$, which is called the *Borel σ-algebra* on Ω. The elements $A \in \mathcal{B}(\Omega)$ are known as *Borel sets*. Every topological space is a measurable space with the Borel σ-algebra.

In the case of $\mathbb{R}^N$, with $N \in \mathbb{N}$, the Borel σ-algebra $\mathcal{B}(\mathbb{R}^N)$ is generated by any type of intervals in $\mathbb{R}^N$. That is, it is generated by any of the following classes of sets:

$$\{(a,b) : a,b \in \mathbb{Q}^N, a < b\}, \quad \{[a,b) : a,b \in \mathbb{Q}^N, a < b\}, \quad \{(a,b] : a,b \in \mathbb{Q}^N, a < b\},$$
$$\{[a,b] : a,b \in \mathbb{Q}^N, a < b\}, \quad \{[a,\infty) : a \in \mathbb{Q}^N,\}, \quad \{(-\infty,b) : b \in \mathbb{Q}^N\},$$
$$\{(a,\infty) : a \in \mathbb{Q}^N\}, \quad \{(-\infty,b] : b \in \mathbb{Q}^N,\},$$

where $(a,b) = (a_1,b_1) \times \cdots \times (a_N,b_N)$ (Cartesian product) denotes an open rectangle in R^N and analogously for the other types of intervals. We write $a < b$ if $a_i < b_i, \forall i = 1, \cdots, N$.

Definition 3.3. (Measure) Let $(\Omega, \mathcal{A})$ be a measurable space. A measure on $(\Omega, \mathcal{A})$ is a set function $\mu : \mathcal{A} \to [0, \infty]$ fulfilling the following conditions:

1. $\mu(\emptyset) = 0$.
2. $\mu(\biguplus_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} \mu(A_n)$ for any collection of countably many mutually disjoint[2] sets $\{A_n\}_{n \in \mathbb{N}} \in \mathcal{A}$.

If additionally $\mu(\Omega) = 1$ then μ is a *probability measure*.

We call the triplet $(\Omega, \mathcal{A}, \mu)$ a *measure space*, that is, a measurable space $(\Omega, \mathcal{A})$ over which a measure μ is defined. In the case where μ is a probability measure the triplet $(\Omega, \mathcal{A}, \mu)$ is a *probability space* and the sets $A \in \mathcal{A}$ are called *events*.

Definition 3.4. Let $\mathcal{A}$ be a σ-algebra on Ω. A measure μ on $\mathcal{A}$ is said to be

[2] The symbol $\biguplus$ emphasizes the fact that the union is disjoint.

1. *finite* if $\mu(A) < \infty$ for all $A \in \mathcal{A}$,
2. *σ-finite* if there existis a sequence of sets $\{\Omega_n\}_{n\in\mathbb{N}} \in \mathcal{A}$ such that $\Omega = \bigcup_{n=1}^{\infty} \Omega_n$ and such that $\mu(\Omega_n) < \infty, \forall n \in \mathbb{N}$.

Note that, if μ is finite, it is also σ-finite, since we can take $\Omega_1 = \Omega$ and $\Omega_i = \emptyset$, $\forall i \geq 2$. However, the opposite is not necessarily true.

Example 3.3. (Measures)

- We define the *indicator function* on the set $A \subset \Omega$ by

$$\mathbb{1}_A(x) := \begin{cases} 1 & if \quad x \in A \\ \\ 0 & if \quad x \notin A \end{cases}.$$

  Let $\omega \in \Omega$ and $\delta_\omega(A) = \mathbb{1}_A(\omega)$. Then δ_ω is a probability measure on any σ-algebra $\mathcal{A} \subset 2^\Omega$, called the *Dirac measure* for the point ω.

- Let $\{\mu_n\}_{n\in\mathbb{N}}$ be a sequence of measures and $\{\alpha_n\}_{n\in\mathbb{N}}$ a sequence of nonnegative numbers. Then $\sum_{n=1}^{\infty} \alpha_n \mu_n$ is a measure.

- A measure is defined on $(\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$, called the *Lebesgue-Borel measure*, denoted by λ^N. It is characterized by the following property:

$$\lambda^N((a,b]) = \prod_{i=1}^{n}(b_i - a_i), \quad \forall a, b \in \mathbb{R}^N \text{ with } a < b.$$

  The Lebesgue-Borel measure is not finite, but it is σ-finite. Indeed, we can write,

$$\mathbb{R}^N = \bigcup_{k\in\mathbb{N}} (-k,k] \times \overset{N)}{\cdots} \times (-k,k],$$

  and $\lambda^N((-k,k]^N) = 2Nk < \infty$.

- Let $(\Omega, \mathcal{A}, \mu)$ be a measure space and $\Omega' \in \mathcal{A}$. The restriction of μ to Ω', defined by

$$\mu\big|_{\Omega'} := \mu(A) \text{ for } A \in \mathcal{A} \text{ with } A \subset \Omega',$$

  is a measure on the σ-algebra $\mathcal{A}\big|_{\Omega'}$.

- The restriction of the Lebesgue-Borel measure λ on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ to $[0,1]$ is a probability measure on $([0,1], \mathcal{B}(\mathbb{R})\big|_{[0,1]})$.

Definition 3.5. (Null set) Let $(\Omega, \mathcal{A}, \mu)$ be a measure space. A set $A \in \mathcal{A}$ is called a *μ-null set* (or simply null set) if $\mu(A) = 0$. We denote by $\mathcal{N}_\mu$ the class of all subsets of μ-null sets.

Definition 3.6. (Complete measure space) A measure space $(\Omega, \mathcal{A}, \mu)$ is called *complete* if $\mathcal{N}_\mu \subset \mathcal{A}$.

Example 3.4. The Lebesgue-Borel measure λ on $(\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$ can be extended uniquely to a measure λ^* on

$$\mathcal{B}^*(\mathbb{R}^N) = \sigma(\mathcal{B}(\mathbb{R}^N) \cup \mathcal{N}) = \{B \cup N : B \in \mathcal{B}(\mathbb{R}^N), N \in \mathcal{N}\},$$

where $\mathcal{N}$ is the class of subsets of Lebesgue-Borel null sets, such that $\lambda^*(B \cup N) = \lambda(B)$ for any $B \in \mathcal{B}(\mathbb{R}^N)$ and $N \in \mathcal{N}$. $\mathcal{B}^*(\mathbb{R}^N)$ is called the σ-algebra of Lebesgue measurable sets and λ^* is called the *Lebesgue measure*. The new space $(\mathbb{R}^N, \mathcal{B}^*(\mathbb{R}^N), \lambda^*)$ is called the *completion* of $(\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N), \lambda)$.

Definition 3.7. (Almost everywhere) Let $(\Omega, \mathcal{A}, \mu)$ be a measure space. Let $E(\omega)$ be a property that a point $\omega \in \Omega$ could have. We say that E holds *μ-almost everywhere* (a.e.) or for *almost all* (a.a.) ω if there exists a null set N such that E holds for every $\omega \in \Omega \setminus N$. If $A \in \mathcal{A}$ and there exists a null set N such that E holds for every $\omega \in A \setminus N$, we say that E holds μ-almost everywhere on A.

If μ is a probability measure, then we say that E holds *μ-almost surely* (a.s.), respectively almost surely on A.

Definition 3.8. (Measurable maps) Let $(\Omega, \mathcal{A})$ and $(\Omega', \mathcal{A}')$ be two measurable spaces. A map $X : \Omega \to \Omega'$ is measurable (with respect to $\mathcal{A}$ and $\mathcal{A}'$) if $X^{-1}(A') \in \mathcal{A}$, $\forall A' \in \mathcal{A}'$, that is, if $X^{-1}(\mathcal{A}') = \{X^{-1}(A') : A' \in \mathcal{A}'\} \subset \mathcal{A}$. When X is measurable, we can also write $X : (\Omega, \mathcal{A}) \to (\Omega', \mathcal{A}')$.

Example 3.5. (Measurable maps)

- The identity map $id : \Omega \to \Omega$ is measurable.
- Any map $X : (\Omega, 2^\Omega) \to (\Omega', \{\emptyset, \Omega'\})$ is measurable.
- The indicatos function $\mathbb{1}_A : (\Omega, \mathcal{A}) \to (\{0,1\}, 2^{\{0,1\}})$, with $A \subset \Omega$, is measurable if and only if $A \in \mathcal{A}$.
- The composition of measurable maps is measurable.
- Every continuous map $f : \Omega \to \Omega'$ between two topological spaces (Ω, τ) and (Ω', τ') is measurable.
- If $(\Omega, \mathcal{A})$ is a measurable space and we define $f := (f_1, \cdots, f_N) : \Omega \to \mathbb{R}^N$, with $f_1, \cdots, f_N : \Omega \to \mathbb{R}$, then

$$f \text{ is measurable } \iff \text{ each } f_i \text{ is measurable,}$$

  where $\mathbb{R}^N$ and $\mathbb{R}$ are considered measurable spaces with the Borel σ-algebra.
- If $h : (\Omega, \mathcal{A}) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ and $f, g : (\Omega, \mathcal{A}) \to (\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$ are measurable maps, then $f + g, f - g, f \cdot h$ and f/h are measurable[3] .

Theorem 3.2. *Let $(\Omega', \mathcal{A}')$ be a measurable space, Ω a nonempty set and $X : \Omega \to \Omega'$ a map. Then the preimage $X^{-1}(\mathcal{A}') = \{X^{-1}(A') : A' \in \mathcal{A}'\}$ is the smallest σ-algebra with respect to which X is measurable. We call $\sigma(X) := X^{-1}(\mathcal{A}')$ the σ-algebra on Ω generated by X.*

The following theorem states that it is sufficient to check measurability on a generator of $\mathcal{A}'$ instead of checking all preimages $X^{-1}(A')$, $A' \in \mathcal{A}'$.

Theorem 3.3. *Let $\mathcal{E}' \subset \mathcal{A}'$ be a class of measurable sets. Then $\sigma(X^{-1}(\mathcal{E}')) = X^{-1}(\sigma(\mathcal{E}'))$ and*

$$X \text{ is measurable w.r.t } \mathcal{A} \text{ and } \sigma(\mathcal{E}') \iff X^{-1}(E') \in \mathcal{A} \text{ for all } E' \in \mathcal{E}'.$$

[3]Here we consider that $\frac{x}{0} := 0$ for all $x \in \mathbb{R}$.

In particular, if $\sigma(\mathcal{E}') = \mathcal{A}'$, *then*

$$X \text{ is measurable w.r.t } \mathcal{A} \text{ and } \mathcal{A}' \iff X^{-1}(\mathcal{E}') \subset \mathcal{A}.$$

Example 3.6.

- A map $X : (\Omega, \mathcal{A}) \to (\Omega', 2^{\Omega'})$ with Ω' being countable is measurable if and only if $X^{-1}(\omega') \in \mathcal{A}$ for every $\omega' \in \Omega'$.
- Let $(\Omega, \mathcal{A})$ be a measurable space. A map $X : \Omega \to \mathbb{R}^N$ is measurable, with the Borel σ-algebra in $\mathbb{R}^N$, if and only if

  $$X^{-1}((-\infty, b]) \in \mathcal{A} \text{ for any } b \in \mathbb{R}^N.$$

  Indeed, recall that $\mathcal{B}(\mathbb{R}^N) = \sigma((-\infty, b], b \in \mathbb{R}^N)$. The same is true for any of the classes given in Ex. 3.2.

Corollary 3.1. *Let* $\mathcal{E} \subset 2^\Omega$ *and* $\emptyset \neq A \subset \Omega$. *Then* $\sigma(\mathcal{E}\big|_A) = \sigma(\mathcal{E})\big|_A$.
In particular, if (Ω, τ) *is a topological space, then* $\mathcal{B}(\Omega, \tau)\big|_A = \mathcal{B}(A, \tau\big|_A)$.

The following definition shows that a measurable map transports a measure from one measurable space to another. It will be useful in the definition of probability distribution of a random variable in a subsequent section.

Definition 3.9. (Image measure) Let $(\Omega, \mathcal{A})$ and $(\Omega', \mathcal{A}')$ be measurable spaces, μ a measure on $(\Omega, \mathcal{A})$ and $X : \Omega \to \Omega'$ a measurable map. The *image measure* of μ under the map X is the measure $\mu \circ X^{-1}$ on $(\Omega', \mathcal{A}')$ defined by

$$\mu \circ X^{-1} : \mathcal{A}' \to [0, \infty], \quad A' \mapsto \mu(X^{-1}(A')).$$

3.2. Some results about integration

Given a measure space $(\Omega, \mathcal{A}, \mu)$, and a measurable map $f : \Omega \to \mathbb{R} \cup \{-\infty, \infty\}$, we can define the integral of f with respect to μ as

$$\int_\Omega f(\omega)\mu(d\omega) := \int_\Omega f d\mu.$$

The definition and construction of this integral is supposed to be known, and it is not going to be explained in this work. Let us simply say that the construction takes places in several progressive steps: first, the integral is defined for the so-called *simple functions*, then, for nonnegative measurable functions and, finally, for measurable functions in general, by using in each step the results obtained in the previous ones. This way, a measurable function is said to be μ-integrable if $\int_\Omega |f| d\mu < \infty$.

For $A \in \mathcal{A}$, we define

$$\int_A f d\mu := \int_\Omega (f \mathbb{1}_A) d\mu.$$

Proposition 3.2. *Let* $f : \Omega \to [0, \infty]$ *be a measurable map. Then,*

1. $f = 0$ *almost everywhere* $\iff \int_\Omega f d\mu = 0$.

2. *If* $\int_\Omega f d\mu < \infty$, *then* $f < \infty$ *almost everywhere.*

Definition 3.10. (Lebesgue integral) Let λ be the Lebesgue measure on $\mathbb{R}^N$ and $f : (\mathbb{R}^N, \mathcal{B}^*(\mathbb{R}^N)) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ a measurable function and λ-integrable. Then the integral

$$\int_{\mathbb{R}^N} f d\lambda$$

is called the *Lebesgue integral* of f. If $A \in \mathcal{B}(\mathbb{R}^N)$ and $f : (A, \mathcal{B}^*(\mathbb{R}^N)\big|_A) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ is measurable and, therefore, $f\mathbb{1}_A$ is measurable with respect to $\mathcal{B}^*(\mathbb{R}^N))$ and $(\mathbb{R}, \mathcal{B}(\mathbb{R})))$, then

$$\int_A f d\lambda := \int_{\mathbb{R}^N} f\mathbb{1}_A d\lambda.$$

The following theorem will be useful to define the expectation of a random variable.

Theorem 3.4. *Let* $(\Omega, \mathcal{A})$ *and* $(\Omega', \mathcal{A}')$ *be measurable spaces,* μ *a measure on* $(\Omega, \mathcal{A})$ *and* $X : \Omega \to \Omega'$ *a measurable map. Let us define* $\mu' = \mu \circ X^{-1}$, *which is the image measure of* μ *under the map* X, *and assume that* $f : \Omega' \to \mathbb{R} \cup \{-\infty, \infty\}$ *is* μ'*-integrable. Then* $f \circ X$ *is* μ*-integrable and*

$$\int_\Omega (f \circ X) d\mu = \int_{\Omega'} f d(\mu \circ X).$$

Finally, we introduce the concept of *density* of a measurable function with respect to a measure, which will be used in the next sections.

Definition 3.11. (Density) Let μ be a measure on the measurable space $(\Omega, \mathcal{A})$ and $f : \Omega \to [0, \infty)$ be a measurable map. Then the following

$$\nu(A) := \int_A f d\mu = \int_\Omega (f\mathbb{1}_A) d\mu, \quad \forall A \in \mathcal{A}$$

is a measure on $(\Omega, \mathcal{A})$. We write $f\mu := \nu$, $f = \frac{d\nu}{d\mu}$, and say that ν has *density* f with respect to μ.

Proposition 3.3. *(*Uniqueness of the density*) Under the conditions of the previous definition, let* ν *be* σ*-finite. If* f_1 *and* f_2 *are densities of* ν *with respect to* μ, *then* $f_1 = f_2$ μ*-almost everywhere. In other words, the density* $\frac{d\nu}{d\mu}$ *is unique* μ*-a.e.*

Proposition 3.4. *Under the conditions of the previous definition, a function* $g : \Omega \to \mathbb{R} \cup \{-\infty, \infty\}$ *is* $f\mu$*-integrable if and only if* gf *is* μ*-integrable, in which case*

$$\int_\Omega g d(f\mu) = \int_\Omega (gf) d\mu$$

3.3. Random Variables and Probability Distribution

Once the fundamentals of measure theory and the notion of integral have been explained, we can move to the central part of probability theory: random variables and probability distributions associated with them. We will pay special attention to real random variables, since they are closer to actual world applications, and, in particular, to machine learning.

Definition 3.12. (Random Variable) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $(\Omega', \mathcal{A}')$ a measurable space. Then, $X : \Omega \to \Omega'$ is called a *random variable* with values in $(\Omega', \mathcal{A}')$ when X is measurable.

If, in particular, $(\Omega', \mathcal{A}') = (\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$, we say that X is a *real* random variable.

We denote $\{X \in A'\} := X^{-1}(A')$, for $A' \in \mathcal{A}'$, so $P[X \in \mathcal{A}'] := P[X^{-1}(A')]$. In particular, we define $\{X \geq 0\} := X^{-1}([0, \infty))$, and similarly for the other types of intervals in $\mathbb{R}^N$.

Now we can define the concept of probability distribution, which will be of great importance throughout this work.

Definition 3.13. (Probability Distribution) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $(\Omega', \mathcal{A}')$ a measurable space. Let $X : \Omega \to \Omega'$ be a random variable.

1. The image measure of P under X, $P_X := P \circ X^{-1}$, is a probability measure on $(\Omega', \mathcal{A}')$ which is called the *probability distribution* of X. We write $X \sim \mu$ if $P_X = \mu$ and say that X has distribution μ.

2. In the particular case where $(\Omega', \mathcal{A}') = (\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$, we can define the map $F_X : \mathbb{R}^N \to [0, 1]$ by

$$F_X(x) = P[X_i \leq x_i,\ \forall i = 1, \cdots, N], \quad \forall x = (x_1, \cdots, x_N) \in \mathbb{R}^N,$$

   which is called the *(joint) distribution function* of P_X (or simply of X).

   When $N > 1$ we have $X = (X_1, \cdots, X_N)$ and we can consider the *marginal distribution* of each of the random variables X_i, that is, P_{X_i}. Then, the *marginal distribution function* of each X_i is given by

$$F_{X_i}(x_i) = \lim_{x_j \to \infty, j \neq i} F_X(x_1, \cdots, x_i \cdots, x_N).$$

3. A family $\{X_i\}_{i \in I}$ (where I is an arbitrary index set) of random variables is said to be *identically distributed* if $P_{X_i} = P_{X_j}$, for all $i, j \in I$. In that case, we write $X_i \overset{\mathcal{D}}{=} X_j$, for all $i, j \in I$.

The distribution function determines the probability distribution of a real random variable in an unambiguous manner and so it is important to characterize a distribution function, which is the aim of the following theorem.

Theorem 3.5. *(*Characterization of the distribution function*) A function $F : \mathbb{R}^N \to [0, 1]$ is the* joint distribution function *of some N-dimensional random variable X if and only if F is nondecreasing and right continuous with respect to all the arguments $x_1, \cdots, x_N$ and it satisfies the following conditions:*

1. $\forall i = 1, \cdots, N, \exists \lim_{x_i \to -\infty} F(x) = 0$, *with* $x = (x_1, \cdots, x_i, \cdots, x_N) \in \mathbb{R}^N$

2. $\exists \lim_{x_i \to \infty,\ i=1,\cdots,N} F(x) = 1$, *with* $x = (x_1, \cdots, x_i, \cdots, x_N) \in \mathbb{R}^N$

3. *For all* $x = (x_1, \cdots, x_N) \in \mathbb{R}^N$ *and all* $\varepsilon_i \in \mathbb{R}^+ (i = 1, \cdots, N)$*, the inequality*

$$F(x_1 + \varepsilon_1, x_2 + \varepsilon_2, \cdots, x_N + \varepsilon_N) - \sum_{i=1}^{N} F(x_1 + \varepsilon_1, \cdots, x_{i-1} + \varepsilon_{i-1}, x_i, x_{i+1} + \varepsilon_{i+1}, \cdots, x_N + \varepsilon_N)$$
$$+ \sum_{\substack{i,j=1 \\ i<j}}^{N} F(x_1 + \varepsilon_1, \cdots x_{i-1} + \varepsilon_{i-1}, x_i, x_{i+1} + \varepsilon_{i+1}, \cdots, x_{j-1} + \varepsilon_{j-1}, x_j, x_{j+1} + \varepsilon_{j+1}, \cdots, x_N + \varepsilon_N)$$
$$+ \cdots + (-1)^N F(x_1, x_2, \cdots, x_N) \geq 0$$

holds.

In the case of real random variables, we can distinguish between two main types: discrete and continuous random variables.

Definition 3.14. (Discrete random variable) A real random variable X defined over the probability space $(\Omega, \mathcal{A}, P)$ is said to be *discrete* if there exists a countable set $E \subset \mathbb{R}^N$ such that $P[X \in E] = 1$. In this case, we call the function $p_X : E \to [0, 1]$, defined by,

$$p_X(x) = P[X = x] = P\left[X_i = x_i, , \ \forall i = 1, \cdots, N\right], \ \ \forall x = (x_1, \cdots, x_N) \in E,$$

the *joint probability mass function* of X. It is always true that

- $p_X(x) \geq 0, \ \ \forall x = (x_1, \cdots, x_N) \in E,$
- $\sum_{x \in E} P(X = x) = 1.$

Theorem 3.6. *(*Characterization of the probability mass function*) Every non-negative real-valued function, defined over a countable set* $E \subset \mathbb{R}^N$*, such that the addition of all its values is equal to one, is the probability mass function of an n-dimensional real random variable with values in E.*

Definition 3.15. (Continuous random variable) A real random variable X defined over the probability space $(\Omega, \mathcal{A}, P)$ is said to be *continuous* if there exists an integrable function $f_X : \mathbb{R}^N \to \mathbb{R}$ such that the probability distribution P_X has density f_X with respect to the Lebesgue measure. That is,

$$F_X(x) = P[X \leq x] = \int_{-\infty}^{x_N} \cdots \int_{-\infty}^{x_1} f_X(t_1, \cdots, t_N) dt_1, \cdots, dt_N, \ \ \forall x = (x_1, \cdots, x_N) \in \mathbb{R}^N.$$

In this case, the density f_X is called the *probability density function* of X. It is always true that

- $f_X(x) > 0, \ \ \forall x = (x_1, \cdots, x_N) \in \mathbb{R}^N,$
- $\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_X(t_1, \cdots, t_N) dt_1, \cdots, dt_N = 1.$

Theorem 3.7. *(*Characterization of the probability density function*) Every integrable non-negative function* $f_X : \mathbb{R}^N \to \mathbb{R}$ *satisfying that* $\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_X(t_1, \cdots, t_N) dt_1, \cdots, dt_N = 1$ *is the probability density function of a continuous n-dimensional real random variable.*

Let us now show a couple of examples of probability distributions that are common in the real world.

Example 3.7. (Probability distributions)

- *Bernoulli Distribution*. Given $p \in [0,1]$, let us assume that $P[X = 1] = p$ and $P[X = 0] = 1 - p$. Then, the real random variable X is of discrete type with $E = \{0,1\}$ and it is said to have a *Bernoulli distribution* with parameter p. This distribution is denoted by *Bern(p)*, and it can be formally seen as

$$Bern(p) = (1-p)\delta_0 + p\delta_1$$

Its distribution function is given by

$$F_X(x) = \begin{cases} 0 & if \quad x < 0, \\ 1-p & if \quad x \in [0,1), \\ 1 & if \quad x \geq 1. \end{cases}$$

- *Normal Distribution*. Let $\mu \in \mathbb{R}$, $\sigma^2 > 0$ and let X be a real random variable with

$$F_X(x) = P[X \leq x] = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt, \quad \forall x \in \mathbb{R}.$$

Then, the probability distribution P_X is called the *Normal distribution* with parameters μ and σ^2 (which correspond to the expectation and variance of X, that will be defined in an upcoming section), and it is denoted by $\mathcal{N}_{\mu,\sigma^2}$. Note that the probability density function is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad \forall x \in \mathbb{R}.$$

In particular, if $\mu = 0$ and $\sigma^2 = 1$, the distribution is known as the *standard* normal distribution, $\mathcal{N}_{0,1}$.

To finish this section we will introduce an important theorem that motivates the definition of a continuous random variable, or, equivalently, a continuous probability distribution.

Definition 3.16. (Absolutely continuous) Let μ and ν be two measures on a measurable space $(\Omega, \mathcal{A})$. Then ν is said to be *absolutely continuous* with respect to μ and it is denoted by $\nu \ll \mu$, if

$$\nu(A) = 0 \text{ for all } A \in \mathcal{A} \text{ such that } \mu(A) = 0.$$

Theorem 3.8. *(*Radon-Nikodym*) Let μ and ν be two σ-finite measures on a measurable space $(\Omega, \mathcal{A})$. Then*

$$\nu \text{ has density w.r.t } \mu \iff \nu \ll \mu.$$

In this case, the density $\frac{d\nu}{d\mu}$ is called the Radon-Nikodym derivative of ν with respect to μ. It is $\mathcal{A}$-measurable and finite μ-a.e.

Therefore, given a random variable X, as both the probability measure P_X and the Lebesgue measure λ^N on $\mathbb{R}^N$ are σ-finite, saying that the probability distribution P_X is absolutely

continuous with respect to λ^N ($P_X \ll \lambda^N$) is equivalent to the existence of density

$$f_X = \frac{dP_X}{d\lambda^N}$$

of the probability distribution with respect to the Lebesgue measure.

3.4. Independence

Here we will present the concept of independence, both of events and of random variables, as well as conditions for independence.

Definition 3.17. (Independence of events) Let $(\Omega, \mathcal{A}, P)$ be a probability space and let I be an arbitrary index set. A family of events $(A_i)_{i\in I}$ is called *independent* if for any finite subset $J \subset I$ it holds that

$$P\left[\bigcap_{j\in J} A_j\right] = \prod_{j\in J} P[A_j].$$

Intuitively, we can think that in an independent family of events the occurrence of an event A_i does not affect the probability that the other events $A_j, j \neq i$, also occur.

We can extend the definition of independence to classes of events:

Definition 3.18. (Independence of classes of events) Let $(\Omega, \mathcal{A}, P)$ be a probability space and let I be an arbitrary index set. The family $(\mathcal{E}_i)_{i\in I}$, with $\mathcal{E}_i \subset \mathcal{A}$ for all $i \in I$, is called *independent* if for any finite subset $J \subset I$ and any choice of $E_j \in \mathcal{E}_j, j \in J$, it holds that

$$P\left[\bigcap_{j\in J} E_j\right] = \prod_{j\in J} P[E_j].$$

Now we can study the independence of random variables. We will see that it can be defined by means of the independence of the σ-algebras that the random variables generate. That is why we have defined independence of classes of events.

In the following we consider a probability space $(\Omega, \mathcal{A}, P)$ and an arbitrary index set I. For each $i \in I$, we shall assume that $(\Omega_i, \mathcal{A}_i)$ is a measurable space and $X_i : (\Omega, \mathcal{A}) \to (\Omega_i, \mathcal{A}_i)$ a random variable, with $\sigma(X_i) = X_i^{-1}(\mathcal{A}_i)$ the σ-algebra generated by X_i.

Definition 3.19. (Independent random variables) The family of random variables $(X_i)_{i\in I}$ is called *independent* if the family of σ-algebras $(\sigma(X_i))_{i\in I}$ is independent.

Theorem 3.9. *The family of random variables $(X_i)_{i\in I}$ is independent if, and only if, for every subset $J \subset I$ with a finite number $|J|$ of indices and every $x = (x_j)_{j\in J} \in \mathbb{R}^{|J|}$,*

$$F_{(X_j)_{j\in J}}(x) = \prod_{j\in J} F_{X_j}(x_j).$$

Corollary 3.2. *In addition to the assumptions of the above theorem, let us now assume that for any finite subset $J \subset I$ we have a family of continuous random variables $(X_j)_{j\in J}$ with joint probability*

density function $f_{(X_j)_{j\in J}}$. *Then, the family* $(X_i)_{i\in I}$ *is independent if, and only if, for every finite* $J \subset I$

$$f_{(X_j)_{j\in J}}(x) = \prod_{j\in J} f_{X_j}(x_j), \quad \forall x \in \mathbb{R}^{|J|}.$$

We say that a family of random variables $(X_i)_{i\in I}$ is independent and identically distributed, i.i.d for short, if it is independent and $P_{X_i} = P_{X_j}$, for all $i, j \in I$. Based on these properties, the notion of simple random sample arises in statistical inference.

Definition 3.20. (Simple Random Sample) Given a random variable X with probability distribution P_X, a *simple random sample*, of size n, of X is a random vector $(X_1, \cdots, X_n)$ where $(X_i)_{i\in\{1,\cdots,n\}}$ is a family of independent and identically distributed random variables with $P_{X_i} = P_X, \forall i \in \{1, \cdots, n\}$.

3.5. Moments of a random variable

Definition 3.21. (Moments) Let $(\Omega, \mathcal{A}, P)$ be a probability space and X a real-valued random variable.

1. If X is P-integrable in Ω, then the *expectation* or *mean* of X is defined by

   $$\mathbb{E}[X] := \int_\Omega X dP.$$

   If $\mathbb{E}[X] = 0$ we say that the random variable X is *centered*. The mean is also sometimes referred to as μ.

   For a real random vector $X = (X_1, \cdots, X_N)$, the *mean vector* or *expected value vector* is defined by

   $$\mathbb{E}[X] := (\mathbb{E}[X_1], \cdots, \mathbb{E}[X_N]),$$

   whenever the individual expectations of each random variable $X_i, i = 1, \cdots, N$, exist.

2. For any $k \in \mathbb{N}$, if X^k is P-integrable in Ω, we can define the *kth-moment* and *kth-central moment* of X as

   $$m_k := \mathbb{E}[X^k] \text{ and } \mu_k := \mathbb{E}[(X - \mathbb{E}[X])^k],$$

   respectively.

   In the case of a real random vector $X = (X_1, \cdots, X_N)$ we can define the N-dimensional *moment* and *central moment* of order $(k_1, \cdots, k_N)$, with $k_i \in \mathbb{N}, i = 1, \cdots, N$ of X, whenever $\prod_{i=1}^N X_i^{k_i}$ is P-integrable, as

   $$m_{k_1,\cdots,k_N} := \mathbb{E}[X^{k_1} \cdots X^{k_n}] \text{ and } \mu_{k_1,\cdots,k_N} := \mathbb{E}[(X - \mathbb{E}[X])^{k_1} \cdots (X - \mathbb{E}[X])^{k_N}],$$

   respectively.

3. If X^2 is P-integrable in Ω, then the *variance* of X is defined as the second central moment of X, that is,

   $$Var(X) := \mu_2 = \mathbb{E}[(X - \mathbb{E}[X])^2].$$

   Based on the variance, the *standard deviation* of X is defined as $\sigma := \sqrt{Var(X)}$. The variance is usually also noted by σ^2.

4. Given two real-valued random variables X, Y, such that X^2, Y^2 are P-integrable, the *covariance* of X and Y is defined by

$$Cov(X, Y) := \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

In particular, $Cov(X, X) = Var(X)$.

If $Cov(X, Y) = 0$, we say that te variables are *uncorrelated*. Otherwise they are *correlated*.

Theorem 3.10. *Let $(\Omega, \mathcal{A}, P)$ be a probability space and X, Y independent real-valued P-integrable random variables. Then XY is integrable and $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$. Therefore, independent random variables are uncorrelated.*

We will now present some important properties of the expectation of a random variable, which are a direct consequence of the properties of the integral.

Proposition 3.5. *(*Properties of the expectation*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $X, Y, X_n, n \in \mathbb{N}$, real-valued P-integrable random variables.*

1. *If $P_X = P_Y$, then $\mathbb{E}[X] = \mathbb{E}[Y]$.*
2. (Linearity). *Let $c \in \mathbb{R}$. It is true that cX and $X + Y$ are P-integrable and*

$$\mathbb{E}[cX] = c\mathbb{E}[X] \text{ and } \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y].$$

3. *If $X \geq 0$ almost surely, then*

$$\mathbb{E}[X] = 0 \iff X = 0 \text{ almost surely}$$

.

4. (Monotonicity). *If $X \leq Y$ almost surely, then $\mathbb{E}[X] \leq \mathbb{E}[Y]$. The equality holds if, and only if, $X = Y$ a.s.*
5. (Triangle inequality) $|\mathbb{E}[X]| \leq \mathbb{E}[|X|]$.
6. *If $X_n \geq 0$ a.s. for all $n \in \mathbb{N}$, then $\mathbb{E}[\sum_{n=1}^{\infty} X_n] = \sum_{n=1}^{\infty} \mathbb{E}[X_n]$.*

The linearity property of the expectation leads to the following result:

Proposition 3.6. *Let $(\Omega, \mathcal{A}, P)$ be a probability space and X, Y real-valued random variable, such that X^2, Y^2 are P-integrable in Ω, then*

$$Cov(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

In particular,

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

Proof.

$$\begin{aligned} Cov(X, Y) &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY - X\mathbb{E}[Y] - \mathbb{E}[X]Y + \mathbb{E}[X]\mathbb{E}[Y]] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[X]\mathbb{E}[Y] + \mathbb{E}[X]\mathbb{E}[Y] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]. \end{aligned}$$

□

Proposition 3.7. *(*Properties of the variance*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and X a real-valued random variable, such that X^2 is P-integrable in Ω. Then $Var(X) \geq 0$ and $Var(X) = 0 \iff X = \mathbb{E}[X]$ a.s.*

Proposition 3.8. *(*Properties of the covariance*) Let $(\Omega, \mathcal{A}, P)$ be a probability space, $X_1, \cdots, X_n, Y_1, \cdots, Y_m$ real-valued random variable whose squares are integrable and $\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_m, e, d \in \mathbb{R}$. Then,*

$$Cov\left[d + \sum_{i=1}^{n} \alpha_i X_i, e + \sum_{j=1}^{m} \beta_j Y_i j\right] = \sum_{i,j} \alpha_i \beta_j Cov[X_i, Y_j].$$

In particular, $Var(\alpha X) = \alpha^2 Var(X)$ and

$$Var\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} Var[X_i] + \sum_{\substack{i,j=1 \\ i \neq j}}^{n} Cov[X_i, X_j].$$

For uncorralated random variables, it is true that $Var\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} Var[X_i]$

3.5.1. Computing expectation

The definition of expectation given above is difficult to use in practice. Therefore, a formulation of the expectation closer to a real application is needed. We will see here how to easily compute the expectation of a function of a real random variable based on the probability mass function or the probability density function, depending on the type of the random variable.

Let us fix a probability space $(\Omega, \mathcal{A}, P)$ and a random variable X defined over it. From Theorem 3.4 we deduce, for any P_X-integrable function $f : X(\Omega) \to \mathbb{R}$, that

$$\int_{\Omega} (f \circ X) dP = \int_{X(\Omega)} f dP_X.$$

This result, together with Proposition 3.4, leads us to the following:

Proposition 3.9. *(*Change of variables*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $(\Omega', \mathcal{A}')$ a measurable space. Let $X : \Omega \to \Omega'$ be a random variable and $g : \Omega' \to \mathbb{R}$ a P_X-integrable function. Then, $(g \circ X)$ is a P-integrable random variable with*

$$\mathbb{E}[g(X)] = \int_{\Omega'} g dP_X.$$

Additionally, when $(\Omega', \mathcal{A}') = (\mathbb{R}^N, \mathcal{B}(\mathbb{R}^N))$:

- *If X is of discrete type, with probability mass function p_X,*

$$\mathbb{E}[g(X)] = \sum_{x \in E} p_X(x) g(x),$$

where $E \subset \mathbb{R}^N$ is a countable set such that $P[X \in E] = 1$.

- *If X is of continuous type with probability density function f_X, then*

$$\mathbb{E}[g(X)] = \int_{\mathbb{R}^N} f_X g d\lambda^N = \int_{\mathbb{R}^N} f_X(x_1, \cdots, x_N) g(x_1, \cdots, x_N) dx_1 \cdots, dx_N.$$

This way, we can calculate the expectation of a real random variable by simple considering g as the identity function. The above formulations depend on the probability distribution of the random variable at hand, so we sometimes write

$$\mathbb{E}_{X \sim P_X}[X] = \int_{\Omega'} x dP_X(x),$$

to explicitly indicate the probability distribution of the considered random variable.

3.6. Conditional Probability

Let us begin this section by defining the classical concept of conditional probability.

Definition 3.22. (Conditional probability) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $B \in \mathcal{A}$ with $P[B] > 0$. We define the conditional probability given B for any $A \in \mathcal{A}$ by

$$P[A|B] = \frac{P[A \cap B]}{P[B]}.$$

$P[\cdot|B]$ is a probability measure on $(\Omega, \mathcal{A})$.

Observation 3.1. Note that conditional probability is not defined when $P[B] = 0$. In this case we can assume the value 0 for the conditional probability, for example.

From the definition of conditional probability, the following result can be easily obtain by induction.

Proposition 3.10. *(*Multiplication Rule*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $A_1, \cdots, A_n \in \mathcal{A}$, with $P\left[\bigcap_{j=1}^{n-1} A_j\right] > 0$. Then,*

$$P\left[\bigcap_{j=1}^{n} A_j\right] = P(A_1)P[A_2|A_1]P[A_3|A_1 \cap A_2] \cdots P\left[A_n \middle| \bigcap_{j=1}^{n-1} A_j\right].$$

Proposition 3.11. *(*Summation formula*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and I be a countable set of indices. Let $(B_i)_{i \in I}$ be a family of pairwise disjoint events such that $P\left[\biguplus_{i \in I} B_i\right] = 1$ and $P[B_i] > 0, \forall i \in I$. Then, for any $A \in \mathcal{A}$,*

$$P[A] = \sum_{i \in I} P[A|B_i]P[B_i].$$

By combining the multiplication and summation formulas and applying the definition of conditional probability, we can obtain the well-known Bayes Rule.

Theorem 3.11. *(*Bayes Rule*) Let $(\Omega, \mathcal{A}, P)$ be a probability space and I be a countable set of indices. Let $(B_i)_{i \in I}$ be a family of pairwise disjoint events such that $P\left[\biguplus_{i \in I} B_i\right] = 1$ and $P[B_i] > 0, \forall i \in I$.*

Then, for any $A \in \mathcal{A}$, with $P[A] > 0$ and any $k \in I$,

$$P[B_k|A] = \frac{P[A|B_k]P[B_k]}{\sum_{i\in I} P[A|B_i]P[B_i]}.$$

Now let $(\Omega, \mathcal{A}, P)$ be a probability space and X a P-integrable random variable defined over it. Clearly, $\mathbb{1}_A X$ is P-integrable for any $A \in \mathcal{A}$, so X is integrable with respect to the probability measure $P[\cdot|A]$ and we can define the expectation of X with respect to $P[\cdot|A]$:

Definition 3.23. Let X be a P-integrable random variable and $A \in \mathcal{A}$ with $P[A] > 0$. Then, the *conditional expectation* of X given A is defined as

$$E[X|A] := \int_{X(\Omega)} X(\omega) P[d\omega|A] = \frac{\mathbb{E}[\mathbb{1}_A X]}{P[A]}.$$

It holds that $P[B|A] = \mathbb{E}[\mathbb{1}_B|A]$, for all $B \in \mathcal{A}$.

We would like to define now the concept of conditional probability distribution for random variables. In order to do that, let us consider a probability space $(\Omega, \mathcal{A}, P)$ and two real random variables $X, Y : \Omega \to \mathbb{R}$. For each $\varepsilon > 0$ and $y \in Y(\Omega)$, we suppose that $(y-\varepsilon, y+\varepsilon] \subset Y(\Omega)$ and $P[y-\varepsilon < Y \leq y+\varepsilon] > 0$. For each $x \in X(\Omega)$, we consider the conditional probability of the event $\{X \leq x\}$ given that $Y \in (y-\varepsilon, y+\varepsilon]$, that is,

$$P[X \leq x | y-\varepsilon < Y \leq y+\varepsilon] = \frac{P[X \leq x, y-\varepsilon < Y \leq y+\varepsilon]}{P[Y \in (y-\varepsilon, y+\varepsilon]]},$$ [4]

By taking the limit when ε approaches zero from the right, we obtain the following definition:

Definition 3.24. (Conditional distribution function) The *conditional distribution function* of a real random variable X, given $Y = y$, is defined as the limit

$$F_{X|Y}(x|y) = \lim_{\varepsilon \to 0^+} P[X \leq x | y-\varepsilon < Y \leq y+\varepsilon],$$

provided that such limit exists.

Since the distribution function determines in an unambiguous manner the probability distribution of a random variable, we can also talk about the *conditional probability distribution* of the randon variable X given $Y = y$, and note it by $P_{X|Y}$.

For a discrete real random variable, we can define the conditional probability mass function as follows:

Definition 3.25. (Conditional probability mass function) Let (X, Y) be a real random vector of the discrete type. Given a fixed $y \in Y(\Omega)$, with $P[Y = y] > 0$, the function

$$P[X = x | Y = y] = \frac{P[X = x, Y = y]}{P[Y = y]}$$

is known as the *conditional probability mass function* of X, given $Y = y$.

Since both members of the definition, $P[X = x, Y = y]$ and $P[Y = y]$ are probability mass functions, it turns out that $P[X = x|Y = y]$ is also a probability mass function, i.e., it is well-defined.

[4] We denote by $P[A, B]$ the probability $P[A \cap B]$, for any events A and B.

In the case of continuous real random variables, we can consider the following concept:

Definition 3.26. (Conditional probability density function) Let (X, Y) be a real random vector of the continuous type. Let us assume that the limit given in Def. 3.24 exists. Given $y \in Y(\Omega)$, we can define the *conditional density function* of X, given $Y = y$, as an integrable non-negative function $f_{X|Y}(x|y)$ satisfying that

$$F_{X|Y}(x|y) = \int_{-\infty}^{x} f_{X|Y}(t|y)dt, \quad \forall x \in X(\Omega).$$

An easier way to calculate the conditional probability density function based on the joint and marginal density functions is given by the following result.

Proposition 3.12. *Let (X, Y) be a real random vector of the continuous type. Let $f_{(X,Y)}$ be its probability density function and f_Y the marginal density function of Y. At every point $(x, y) \in \mathbb{R}^2$ at which $f_{(X,Y)}$ and f_Y are continuous and $f_Y(y) > 0$, the conditional density function of X, given $Y = y$, exists and can be expressed as*

$$f_{X|Y}(x|y) = \frac{f_{(X,Y)}(x, y)}{f_Y(y)}.$$

Proof. At every point $(x, y) \in \mathbb{R}^2$ at which $f_{(X,Y)}$ and f_Y are continuous and $f_Y(y) > 0$, we have

$$F_{X|Y}(x|y) = \lim_{\varepsilon \to 0^+} \frac{P[X \leq x, y - \varepsilon < Y \leq y + \varepsilon]}{P[Y \in (y - \varepsilon, y + \varepsilon]]} = \lim_{\varepsilon \to 0^+} \frac{\int_{-\infty}^{x} \left[\int_{y-\varepsilon}^{y+\varepsilon} f_{X,Y}(u, v)dv\right] du}{\int_{y-\varepsilon}^{y+\varepsilon} f_Y(v)dv}$$

Dividing numerator and denominator by 2ε and computing the limit, thanks to the continuity we get that

$$F_{X|Y}(x|y) = \frac{\int_{-\infty}^{x} f_{X,Y}(u, y)du}{f_Y(y)} = \int_{-\infty}^{x} \left[\frac{f_{X,Y}(u, y)}{f_Y(y)}\right] du,$$

as we wanted. □

Going back to the conditional expectation, for real random variables it can be calculated based on their probability distributions.

Definition 3.27. Let X and Y be real random variables defined on a probability space $(\Omega, \mathcal{A}, P)$ and $h : \mathbb{R} \to \mathbb{R}$ a measurable function. Then, the conditional expectation of $h(X)$ given Y, $\mathbb{E}[h(X)|Y]$, is a real random variable that takes the value

$$\mathbb{E}[h(X)|Y = y] := \begin{cases} \sum_x h(x)P[X = x|Y = y] & if \quad (X, Y) \text{ is of discrete type and } P[Y = y] > 0 \\ \int_{-\infty}^{\infty} h(x)f_{X|Y}(x|y)dx & if \quad (X, Y) \text{ is of continuous type and } f_Y(y) > 0 \end{cases},$$

when the random variable Y takes the value y.

All the results given above for real random variables can be easily extended to N-dimensional real random vectors.

3.7. Convergence

This brief final section shall describe different notions of convergence, paying special attention to convergence in probability, which will be necessary to understand the concepts of the theory of consistency in the learning theory chapter.

In the following, we fix a σ-finite measure space $(\Omega, \mathcal{A}, \mu)$ and a separable metric space (E, d) with the Borel σ-algebra $\mathcal{B}(E)$, where d is a distance function. By separable we mean that there exists a countable subset of E that is dense in E. The reader is assumed to understand those concepts (dense subset, metric space, etc.) and they will not be further explained.

Lemma 3.1. *Let $f, g : \Omega \to E$ be measurable with respect to $\mathcal{A}$ and $\mathcal{B}(E)$. Then, the map*

$$H : \Omega \to R_0^+, \quad \omega \mapsto d(f(\omega), g(\omega))$$

is measurable with respect to $\mathcal{A}$ and $\mathcal{B}(\mathbb{R}_0^+)$.

Thanks to this lemma, the following definitions make sense.

Definition 3.28. Let $f, f_1, f_2, \cdots : \Omega \to E$ be measurable with respect to $\mathcal{A}$ and $\mathcal{B}(E)$. We say that the sequence $(f_n)_{n\in\mathbb{N}}$ converges to f

1. in *μ-measure* (or simple in measure), which is denoted by $f_n \xrightarrow{meas} f$, if

$$\mu\left(\{d(f, f_n) > \varepsilon\} \cap A\right) \xrightarrow{n\to\infty} 0$$

   for all $\varepsilon > 0$ and all $A \in \mathcal{A}$ with $\mu(A) < \infty$, and

2. *μ-almost everywhere*, $f_n \xrightarrow{a.e.} f$, if there exists a μ-null set $N \in \mathcal{A}$ such that

$$d(f(\omega), f_n(\omega)) \xrightarrow{n\to\infty} 0$$

   for all $\omega \in \Omega \setminus N$.

When μ is a probability measure, the convergence in measure is called *convergence in probability* and it is denoted by $f_n \xrightarrow{p} f$. If $(f_n)_{n\in\mathbb{N}}$ converges a.e. we also say that it converges almost surely (a.s).

Observation 3.2. Almost everywhere convergence implies convergence in measure, but the opposite is not true.

Observation 3.3. Convergence in measure determines the limit a.e. That is, if $f_n \xrightarrow{meas} f$ and $f_n \xrightarrow{meas} g$, then $\mu(\{d(f, g) > 0\}) = 0$.

In the particular case of real random variables, convergence in probability is defined as follows.

Definition 3.29. (Convergence in probability) Let $(\Omega, \mathcal{A}, P)$ be a probability space and $X, X_1, X_2, \cdots : \Omega \to \mathbb{R}$ real random variables. The sequence $(X_n)_{n\in\mathbb{N}}$ converges in probability to the random variable X if

$$P\left[|X_n - X| > \varepsilon\right] \xrightarrow{n\to\infty} 0,$$

for all $\varepsilon > 0$.

4. Learning Theory

In this chapter the main ideas behind machine learning are presented in a formal way. The definition of a learning problem is introduced together with the theory that accompanies it. Additionally, the main results of the theory of generalization are explained following its original formulation by V.N.Vapnik. Finally, the underlying challenges of most learning problems are discussed and some possible solutions are exposed.
The books used for this chapter have been [14], [5], [15], [1], [6], [16].

4.1. The Learning Problem

Let us begin this section by defining the learning problem. Classically, in the Artificial Intelligence field, learning a task has been considered as the process by which a computer program is able to improve its performance at that specific task through experience. More precisely, in 1997, Tom Mitchell [14] stated: *a computer program learns from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*.

For his part, Vladimir N. Vapnik [15] describes the learning model as a combination of three components:

- A generator G of random vectors $x \in \mathcal{X}$, where $\mathcal{X}$ is the *input or instance space*, drawn independently from a probability distribution $P(x)$[1], that is fixed (the distribution does not change over time) but unknown.

- A supervisor S which returns a value (random variable) $y \in \mathcal{Y}$ for each input vector x, being $\mathcal{Y}$ the *output* or *label space*, according to a conditional probability distribution $P(y|x)$[2], fixed and unknown, called the *target distribution*.

- A learning machine (learning algorithm or learner) $\mathcal{A}$ which is able to implement a set of functions $\{h(x, \alpha), \alpha \in \Lambda\}$, called the hypothesis set, $\mathcal{H}(\Lambda)$, being Λ any set of parameters (scalar values, vectors or even abstract elements).

In that way, the learning problem consists in choosing the hypothesis function from the given set $h(x, \alpha)$, $\alpha \in \Lambda$[3], that bests approximates the supervisor's response for each vector x. It is the learning algorithm who is responsible for choosing the best hypothesis from $\mathcal{H}(\Lambda)$. In order to do so, a finite set of independent and identically distributed (i.i.d) observations drawn from the joint probability distribution $P(x, y) = P(x)P(y|x)$, which is defined over

[1] For greater clarity, in this section we denote the probability distribution of a random variable X by $P(X)$ instead of P_X, or simply P when the random variable is clear. Additionally, we denote by $\mathbb{P}$ any probability measure over a measurable space $(\Omega, \mathcal{F})$.

[2] This includes the case where the supervisor response is determined by a function $y = f(x)$. We can think of $P(y|x)$ being zero for all y except when $y = f(x)$.

[3] We shall omit the braces when referring to a set of functions, since it will be specified when it is a set and when it is a fixed function.

the space $\mathcal{X} \times \mathcal{Y}$, is used. This set is called the training set[4], $\mathcal{D}$, and it is the only data that the learning machine has access to:

$$(x_1, y_1), \dots (x_m, y_m)$$

The joint probability distribution $P(x, y)$ is composed of two parts: a marginal distribution $P(x)$ over the unlabeled random vectors in $\mathcal{X}$ and a conditional probability distribution over the labels in $\mathcal{Y}$, $P(y|x)$. The conditional probability distribution $P(y|x)$ is what must be learned, while the input distribution $P(x)$ determines how likely it is to observe a point $x \in \mathcal{X}$.

With a view to choosing the best approximation function to the supervisor's response from the hypothesis set, a measure of the loss or discrepancy between the response y of the supervisor to an input x and the function $h(x, \alpha)$ provided by $\mathcal{A}$ must be defined. The *loss function*, $\ell : \Lambda \times \mathcal{Z} \to \mathbb{R}^+$, measures that discrepancy, where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. That is, each function $\ell(\alpha, \cdot) : \mathcal{Z} \to \mathbb{R}^+$ measures the loss corresponding to each vector $z \in \mathcal{Z}$, for a fixed value of $\alpha \in \Lambda$.

To measure how likely it is that the hypothesis function $h(x, \alpha)$ provided by the learning algorithm predicts a different value y that the one given by the supervisor for a random input x, the *true error* (or *true risk*) is defined as the expected loss of $h(x, \alpha)$, $\alpha \in \Lambda$, with respect to the probability distribution $P(x, y)$ over $\mathcal{X} \times \mathcal{Y}$, which is given by the following risk functional:

$$L_P(\alpha) \stackrel{\text{def}}{=} \int_{\mathcal{X} \times \mathcal{Y}} \ell(\alpha, (x, y)) dP(x, y) = \mathbb{E}_{(x,y)\sim P}[\ell(\alpha, (x, y))], \quad \alpha \in \Lambda$$

Thus, the goal of the learning task is to find the function $h \in \mathcal{H}(\Lambda)$ which minimizes the previous risk functional over the class of functions $\mathcal{H}(\Lambda) = \{h(x, \alpha) : \alpha \in \Lambda\}$ using only the available information of the training set, since the joint probability distribution $P(x, y)$ is unknown. Finding the best hypothesis function is equivalent to selecting the corresponding value α of the parameter that determines that hypothesis. We shall assume that the minimum of the functional exists for a parameter $\alpha \in \Lambda$.

It is worth mentioning that the choice of the loss function (sometimes also called the *error measure*) affects the outcome of the learning process, i.e., different loss functions may lead to different choices of the best hypothesis h, since the value of a particular loss may produce a small true error while the value of another loss may result in a large error, even for the same training set and probability distribution. Indeed, the same learning task in different situations may require the use of different loss functions.

There exist different learning paradigms regarding the philosophy used in the acquisition of the training data:

- **Supervised learning:** Each training example is labeled by the correct output that the hypothesis function selected by the learner should be able to produce. In this way, a training example can be seen as a tuple (x, y) where x is a random vector of features given as input to the learning machine and y is the supervisor's response for x that the learner should reproduce. It is the most used and studied type of learning and, in fact, it is the learning paradigm that we have been discussing so far as well as in which we will focus through this work.

[4]Even though it is called a "set", actually it is a sequence. The same example may appear more than once in $\mathcal{D}$ and some learning machines may consider the order of the elements in $\mathcal{D}$.

The general setting of the supervised learning problem is depicted in Figure 4.1, obtained from the book *Learning from Data* [6].

- **Unsupervised learning:** In this setting the training data does not contain any output information at all. Each training example is composed solely by a random input vector of features x without any supervisor's response. The unsupervised paradigm can be seen as the task of looking for patterns and structures in the input data or, in other words, the task of creating a high-level representation of the data. Unsupervised learning can be used in a previous step to supervised learning as a data exploratory technique or for data preprocessing.
- **Reinforcement learning:** A training example does not contain the correct output, but instead it comprises a possible output together with a measure of the grade of goodness of that output. That is, a training example in this setting is a triplet $(x, \text{some } y, \text{grade for } y)$. It should be noted that in an example there is no information of the grade for a different output for the same input x. In this way, the learner uses the examples to reinforce the better actions. An application example of this paradigm is learning how to play a game.

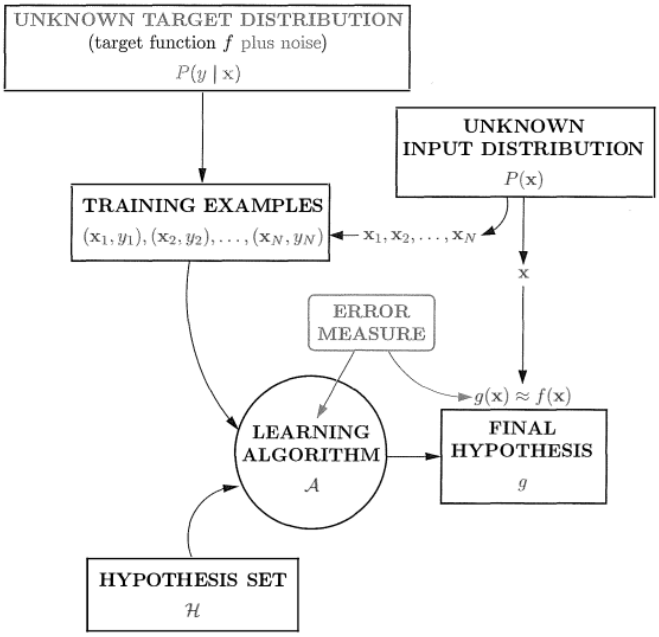


Figure 4.1.: Components of the general supervised learning problem

On the whole, the general setting of the learning problem can be described as follows:

Definition 4.1. (Learning Problem) Let P be any probability distribution defined on a space of examples $\mathcal{Z} \subset \mathbb{R}^N$, Λ a set of parameters, and $\ell : \Lambda \times \mathcal{Z} \to \mathbb{R}^+$ a function such that for each fixed $\alpha \in \Lambda$, $\ell(\alpha, \cdot) : \mathcal{Z} \to \mathbb{R}^+$ is integrable with respect to P in $\mathcal{Z}$. Let us consider the functional:

$$L_P(\alpha) \stackrel{\text{def}}{=} \int_{\mathcal{Z}} \ell(\alpha, z) dP(z) = \mathbb{E}_{z \sim P}[\ell(\alpha, z)], \quad \alpha \in \Lambda \tag{4.1}$$

The goal is to minimize the functional (4.1) in the case where the probability distribution $P(z)$ is unknown but an i.i.d sample from P, $z_1, \ldots, z_m$, is given, supposing that there exists the minimum of (4.1) in Λ.

It should be noted that in the case of supervised learning it is $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, while for unsupervised learning $\mathcal{Z} = \mathcal{X}$.

Observation 4.1. In the above definition, we implicitly consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and the probability distribution P is assumed to be of any real random vector $Z : (\Omega, \mathcal{F}) \to (\mathcal{Z}, \mathcal{B}(\mathcal{Z}))$, where $\mathcal{B}(\mathcal{Z})$ is the Borel σ-algebra restricted to $\mathcal{Z}$. Then, the probability measure P is considered to be defined over $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}))$. Similarly, the function $\ell(\alpha, \cdot) : (\mathcal{Z}, \mathcal{B}(\mathcal{Z})) \to (\mathbb{R}^+, \mathcal{B}(\mathbb{R}^+))$ is a real random variable for each $\alpha \in \Lambda$ and the risk functional is the expected value of that random variable.

For the sake of simplicity, from now on we will refer to a learning problem as a triplet $(\mathcal{Z}, \Lambda, \ell)$ fulfilling the conditions of Def. 4.1.

4.1.1. The Empirical Risk Minimization (ERM) Rule

As mentioned before, the learning algorithm does not know the probability distribution P that generates the data points, the only available data are the training examples, $\mathcal{D} = \{z_1, \ldots, z_m\} \subset \mathcal{Z}^m$, and, thus, the true error is unknown to the learner. Since the training set is the snapshot of the world that the learner has access to, it makes sense to define an error that is based on the training data and to search for a hypothesis that works well on that data. The *training error*, *empirical error* or *empirical risk functional* is defined as the expected loss over the sample $\mathcal{D}$:

$$L_{\mathcal{D}}(\alpha) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} \ell(\alpha, z_i) \tag{4.2}$$

The subscript $\mathcal{D}$ emphasizes the fact that the error is calculated over the training set and depends on it.

Therefore, the objective is to minimize the empirical risk functional (4.2) instead of the functional (4.1). In this sense, the function $\ell(\alpha, z)$ that minimizes (4.1) is approximated by the funcion $\ell(\alpha_m, z)$ that minimizes (4.2). This learning principle is called the *Empirical Risk Minimization* and it plays a crucial role in learning theory.

It is worth noting that, following this learning paradigm, the final hypothesis selected by $\mathcal{A}$ depends very much on the training data $\mathcal{D}$.

It is common to apply the ERM rule over a restricted search space, i.e., the learner choose before seeing the training data a hypothesis class $\mathcal{H}(\Lambda)$ to apply the ERM rule over it. More specifically, for a given training set $\mathcal{D}$ and a hypothesis class $\mathcal{H}(\Lambda)$, the learning machine uses the ERM rule to choose a hypothesis $h_{\mathcal{D}} \in \mathcal{H}(\Lambda)$ with the lowest possible error over $\mathcal{D}$:

$$h_{\mathcal{D}} \in argmin_{h \in \mathcal{H}(\Lambda)} L_{\mathcal{D}}(\alpha), \quad \alpha \in \Lambda$$

where *argmin* denotes the subset of hypothesis from $\mathcal{H}(\Lambda)$ that achieve the minimum value of $L_{\mathcal{D}}$ over Λ. Since the learner is restricted to choosing a function from a previously fixed hypothesis set, a bias is introduced toward that particular set of hypothesis, which is called an *inductive bias*. The choice of the hypothesis class should be ideally based on prior knowledge of the problem to be learned.

4.2. The Probably Approximately Correct (PAC) learning model

From a logical point of view, one might try to learn a hypothesis $h(\cdot, \alpha)$, $\alpha \in \Lambda$, for which $L_P(\alpha) = 0$. Nevertheless, it is useless in the learning problem setting we are considering for two reasons. On the one hand, unless the learner have access to training examples corresponding to every possible vector from $\mathcal{Z}$ (which is an unrealistic scenario), there may be several hypothesis in the given class $\mathcal{H}(\Lambda)$ which are consistent with the provided training examples, and the learning machine cannot be sure that the selected one approximates best the target distribution. In other words, there may always be some fine details of P[5] that the training data does not represent due to its finite nature. On the other hand, since the training points are generated randomly according to P, there will always be some non-zero probability that the given examples are misleading, that is, non-informative of the distribution P (for example, the training set contains the same point sampled multiple times).

In order to alleviate those problems, two new parameters are introduced in the learning problem setting. First, the true error of the hypothesis selected by the learner is bounded by a constant ε (the accuracy parameter), that can be made arbitrarily small, so that the error of the hypothesis is not required to be zero. Second, the probability of failure of the learner given a sequence of randomly drawn training examples will be bounded by a constant δ (confidence parameter), which can also be as small as desired, i.e., the learning algorithm is not required to succeed for every training sequence. To put it briefly, it is only required that the learning machine *probably* learns a hypothesis that is *approximately correct* (hence the PAC terminology).

Moreover, if the chosen set $\mathcal{H}(\Lambda)$ does not contain a hypothesis $h(\cdot, \alpha)$, $\alpha \in \Lambda$, such that $L_{\mathcal{D}}(\alpha) = 0$, it is not possible to find a function in that hypothesis class such that $L_P(\alpha) = 0$. In this case, the most that can be learned is a hypothesis from $\mathcal{H}(\Lambda)$ that has the minimum empirical error.

In [5] the agnostic PAC learnability is defined formally as follows:

Definition 4.2. (Agnostic PAC learnability) Given a learning task $(\mathcal{Z}, \Lambda, \ell)$, the hypothesis class $\mathcal{H}(\Lambda)$ is agnostic PAC learnable with respect to $\mathcal{Z}$ and the loss function ℓ, if there exists a function $m_\Lambda : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property:

- For every $\varepsilon, \delta \in (0,1)$ and for every probability distribution P defined over $\mathcal{Z}$, when applying $\mathcal{A}$ on $m \geq m_\Lambda(\varepsilon, \delta)$ examples i.i.d generated by P, $\mathcal{A}$ returns $h(\cdot, \alpha)$, $\alpha \in \Lambda$, such that
$$\mathbb{P}_{\mathcal{D} \sim P^m}[L_P(\alpha) - \inf_{\alpha' \in \Lambda} L_P(\alpha') \leq \varepsilon] \geq 1 - \delta$$

When such an algorithm $\mathcal{A}$ exists, it is called an *agnostic* PAC-learning algorithm (or *agnostic* PAC learner) for $\mathcal{H}(\Lambda)$.

The function $m_\Lambda : (0,1)^2 \to \mathbb{N}$ determines the sample complexity (that is, the required number of examples) of learning a *probably approximately correct* hypothesis from $\mathcal{H}(\Lambda)$. It is a function of the accuracy, ε, and the confidence, δ, parameters and also depends on some properties of the hypothesis class. As there are many functions m_Λ satisfying the requirements given in the definition of PAC learnability, provided that $\mathcal{H}(\Lambda)$ is PAC-learnable, the sample complexity function is defined as the "minimal function", i.e., for any $\varepsilon, \delta \in (0,1)$, $m_\Lambda(\varepsilon, \delta)$ is the minimal integer that satisfies the requirements of PAC learnability with accuracy ε and confidence δ.

[5] P is the joint probability distribution, $P(x, y) = P(x)P(y|x)$, that generates the training examples.

Observation 4.2. In the original definition by Valiant (1984) the sample complexity is required to be polynomial in $1/\varepsilon$, $1/\delta$ as well as in the computational representation size of the hypothesis in the class, but we will see later that it is always the case whenever a class is PAC-learnable.

In the particular case where the target distribution is a deterministic function, that is, the supervisor response to a point $x \in \mathcal{X}$ is uniquely determined by a measurable function $f : \mathcal{X} \to \mathcal{Y}$, it suffices to consider a distribution P over the input space $\mathcal{X}$, being the training examples obtained by drawing $(x_1, \ldots, x_m)$ from P and labeling them with f. Additionally, if there exists a hypothesis $h(\alpha, \cdot)$, $\alpha \in \Lambda$, in $\mathcal{H}(\Lambda)$ satisfying that $L_{\mathcal{D}}(\alpha) = 0$, the definition of PAC learnability can be formulated as follows:

Definition 4.3. (PAC learnability) Let P be any probability distribution defined on a space $\mathcal{X} \subset \mathbb{R}^N$, $\mathcal{Y}$ a label space and Λ a set of parameters. The hypothesis class $\mathcal{H}(\Lambda)$ is PAC learnable if there exists a function $m_\Lambda : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property:

- For every $\varepsilon, \delta \in (0,1)$, for every probability distribution P over $\mathcal{X}$, and for every labeling measurable function $f : \mathcal{X} \to \mathcal{Y}$, if there exists a hypothesis in $\mathcal{H}(\Lambda)$ which attains zero error over the training set, then when applying $\mathcal{A}$ on $m \geq m_\Lambda(\varepsilon, \delta)$ examples i.i.d generated by P and labeled by f, $\mathcal{A}$ returns $h(\cdot, \alpha)$, $\alpha \in \Lambda$, such that

$$\mathbb{P}_{\mathcal{D} \sim P^m}[L_{P,f}(\alpha) \leq \varepsilon] \geq 1 - \delta$$

The above definition is considered in many books, so it is worth contemplating it in this work. It is a particular case of the agnostic PAC learnability: while in a PAC leaning scenario the learner is required to achieve a small error in absolute terms, in the case of agnostic PAC learnability a learning algorithm can still be successful if its true error is not far from the best error attained by a hypothesis from $\mathcal{H}(\Lambda)$.

4.3. Theory of generalization

We have seen before that under the ERM rule the learner evaluates the empirical risk of every hypothesis $h \in \mathcal{H}(\Lambda)$, with Λ a given set of parameters, over the training set $\mathcal{D}$ and returns a hypothesis $h(\cdot, \alpha)$, $\alpha \in \Lambda$, which minimizes that empirical risk. In this section we discuss the question of whether a hypothesis that minimizes the empirical risk with respect to a given finite set of examples $\mathcal{D}$ is also a minimizer of the true risk with respect to the real probability distribution P. In order to obtain an affirmative answer to that question, we will try to bound the discrepancy between the true error and the empirical error (called *generalization error*), i.e., the value $|L_{\mathcal{D}}(\alpha) - L_P(\alpha)|$, for all $\alpha \in \Lambda$. In other words, we will need that uniformly over all the hypothesis in the class $\mathcal{H}(\Lambda)$, the generalization error is small. In that case, we say that the learning machine generalizes well.

4.3.1. Theory of consistency

We start this section by introducing some conceptual asymptotic results. In particular, the definition of *consistency* of the ERM principle is presented, as well as the key theorem of learning theory that establishes necessary and sufficient conditions for consistency. Additionally, a relation between the concepts of consistency and PAC learnability is given.

Proofs of all the results in this part can be found in [1]. We omit them here because they are tedious and are out of the scope of this work.

Definition 4.4. (*consistency* of the ERM principle) Let $(\mathcal{Z}, \Lambda, \ell)$ be a learning task, P a probability distribution over $\mathcal{Z}$ and $\mathcal{D} = \{z_1, \dots, z_m\} \subset \mathcal{Z}^m$ a set of examples i.i.d. generated by P. Assume that $\alpha_m \in \Lambda$ is a parameter such that $\ell(\alpha_m, z)$ minimizes the empirical risk functional over $\mathcal{D}$. The ERM principle is said to be **consistent** (w.r.t domain $\mathcal{Z}$, set Λ, loss function ℓ and probability distribution P) if the following two sequences converge in probability to the same limit:

$$L_P(\alpha_m) \xrightarrow[m\to\infty]{p} \inf_{\alpha\in\Lambda} L_P(\alpha) \tag{4.3}$$

$$L_{\mathcal{D}}(\alpha_m) \xrightarrow[m\to\infty]{p} \inf_{\alpha\in\Lambda} L_P(\alpha) \tag{4.4}$$

Therefore, the ERM method is consistent if it returns a sequence of functions $h(\cdot, \alpha_m) : \mathcal{Z} \to \mathbb{R}^+$, $m = 1, 2, ...$, whose empirical error and true error converge to the minimal possible error over the hypothesis class $\mathcal{H}(\Lambda)$.

It is possible that the given set of functions $\mathcal{H}(\Lambda)$ contains a minorizing hypothesis of the true risk, in which case the consistency property is fulfilled trivially: the minimum of the empirical error will be attained at that minorizing function as well as the minimum of the true error, for all probability distributions over $\mathcal{Z}$ and any number of observations m. In other words, there exist cases of trivial consistency that depend on whether the hypothesis class contains a minimizing function. Thus, it is necessary to define a new consistency property that takes those cases into account, so that the consistency of the ERM principle does not depend on the individual properties of the elements $h(\cdot, \alpha)$, $\alpha \in \Lambda$, but on the general properties of the hypothesis class.

Definition 4.5. (*strict (non-trivial) consistency* of the ERM principle) Under the same conditions of Def. 4.4, the ERM principle is said to be **strictly (non-trivially) consistent** (w.r.t domain $\mathcal{Z}$, set Λ, loss function ℓ and probability distribution P) if for any non-empty subset $\Lambda(c)$, $c \in (-\infty, \infty)$, of the set of functions $\mathcal{H}(\Lambda)$ defined as:

$$\Lambda(c) = \{\alpha : L_P(\alpha) > c, \alpha \in \Lambda\}$$

the following convergence holds

$$\inf_{\alpha\in\Lambda(c)} L_{\mathcal{D}}(\alpha_m) \xrightarrow[m\to\infty]{p} \inf_{\alpha\in\Lambda(c)} L_P(\alpha) \tag{4.5}$$

That is, the ERM method is strictly consistent if the convergence (4.5) is true for all the subsets $\Lambda(c)$ of hypothesis functions that remain after the ones with a smaller risk than c are removed from the hypothesis class.

Note that the convergence condition (4.3) of Def. 4.4 does not appear in the above definition. The reason is that (4.3) is automatically satisfied under the condition of Def. 4.5:

Lemma 4.1. *Under the conditions of Def. 4.4, if the ERM method is strictly consistent, then the following convergence in probability is true:*

$$L_P(\alpha_m) \xrightarrow[m\to\infty]{p} \inf_{\alpha\in\Lambda} L_P(\alpha)$$

4.3.1.1. Relation between PAC learnability and consistency

Let us assume the same context of Def. 4.4. If we rewrite (4.3) using the definition of convergence in probability and limit we have:

$$\begin{aligned} L_P(\alpha_m) \xrightarrow[m\to\infty]{p} \inf_{\alpha\in\Lambda} L_P(\alpha) &\overset{def}{\Leftrightarrow} \lim_{m\to\infty} \mathbb{P}_{\mathcal{D}\sim P^m}[|L_P(\alpha_m) - \inf_{\alpha\in\Lambda} L_P(\alpha)| > \varepsilon] = 0\,, \forall \varepsilon > 0 \\ &\overset{def}{\Leftrightarrow} \forall \delta > 0, \exists M > 0 : \forall m \geq M, \mathbb{P}_{\mathcal{D}\sim P^m}[|L_P(\alpha_m) - \inf_{\alpha\in\Lambda} L_P(\alpha)| > \varepsilon] < \delta, \forall \varepsilon > 0 \end{aligned} \tag{4.6}$$

Going back to Def. 4.2, it is clear that (4.6) is similar to the definition of agnostic PAC-learnability with sample complexity M. In the above expression we have the absolute value of the difference between risks but not in Def. 4.2. However, it is always true that $L_P(\alpha_m) \geq \inf_{\alpha\in\Lambda} L_P(\alpha)$, and the following result follows immediately:

Proposition 4.1. *Let $(\mathcal{Z}, \Lambda, \ell)$ be a learning task. If the ERM method is strictly consistent (w.r.t domain $\mathcal{Z}$, set Λ and loss function ℓ) for all probability distributions P over $\mathcal{Z}$, then the ERM method is a successful agnostic PAC-learner for $\mathcal{H}(\Lambda)$ (equivalently the hypothesis class $\mathcal{H}(\Lambda)$ is agnostic PAC learnable w.r.t $\mathcal{Z}$ and ℓ).*

Note that the other implication is not necessarily true since PAC learnability says nothing about the convergence condition (4.4).

4.3.1.2. The key theorem of learning theory

Theorem 4.1. *Let $(\mathcal{Z}, \Lambda, \ell)$ be a learning task, P a probability distribution over $\mathcal{Z}$ and $\mathcal{D} = \{z_1, \ldots, z_m\} \subset \mathcal{Z}^m$ a set of examples i.i.d. generated by P. Suppose that there exist constants A and B such that all the functions in the set $\ell(\alpha, z)$, $\alpha \in \Lambda$, satisfy the condition*

$$A \leq L_P(\alpha) \leq B, \ \forall \alpha \in \Lambda$$

Then the ERM principle is strictly consistent (w.r.t domain $\mathcal{Z}$, set Λ, loss function ℓ and probability distribution P) if, and only if, the empirical risk $L_{\mathcal{D}}$ converge uniformly to the true risk L_P over the set of functions $\ell(\alpha, z)$, $\alpha \in \Lambda$, for the given distribution P in the following sense:

$$\lim_{m\to\infty} \mathbb{P}_{\mathcal{D}\sim P^m}[\sup_{\alpha\in\Lambda}(L_P(\alpha) - L_{\mathcal{D}}(\alpha)) > \varepsilon] = 0\,, \forall \varepsilon > 0 \tag{4.7}$$

Briefly, the theorem states that consistency of the ERM principle is equivalent to the existence of uniform convergence in the sense of (4.7).

The theorem is formulated for a fixed probability distribution, but the interesting case for learning theory is when the ERM principle is consistent for any probability distribution over the domain $\mathcal{Z}$. Thus, the theorem can be reformulated as follows:

Corollary 4.1. *Let $(\mathcal{Z}, \Lambda, \ell)$ be a learning task. Suppose that there exist constants A and B such that all the functions in the set $\ell(\alpha, z)$, $\alpha \in \Lambda$, satisfy the condition*

$$A \leq L_P(\alpha) \leq B, \ \forall \alpha \in \Lambda$$

for all probability distributions P over $\mathcal{Z}$. Then the ERM principle is strictly consistent (w.r.t domain $\mathcal{Z}$, set Λ, loss function ℓ) for any probability distribution P if, and only if, the empirical risk $L_{\mathcal{D}}$

converge uniformly to the true risk L_P over the set of functions $\ell(\alpha, z)$, $\alpha \in \Lambda$, for any distribution P in the following sense:

$$\lim_{m\to\infty} \mathbb{P}_{\mathcal{D}\sim P^m}[\sup_{\alpha\in\Lambda}(L_P(\alpha) - L_{\mathcal{D}}(\alpha)) > \varepsilon] = 0 \,, \forall \varepsilon > 0$$

If we consider the sequence of random variables

$$\xi_m^+ = \sup_{\alpha\in\Lambda}\left(\int_{\mathcal{Z}} \ell(\alpha, z) dP(z) - \frac{1}{m}\sum_{i=1}^{m} \ell(\alpha, z_i)\right)_+ , \quad m = 1, 2, \ldots$$

with $u_+ = \begin{cases} u & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases}$, the problem of determining consistency of the ERM method is reduced to describing conditions under which the following holds for any $\varepsilon > 0$:

$$\mathbb{P}_{\mathcal{D}\sim P^m}[\xi_m^+ > \varepsilon] \xrightarrow[m\to\infty]{} 0 \tag{4.8}$$

We can further define the random variables

$$\xi_m = \sup_{\alpha\in\Lambda}\left|\int_{\mathcal{Z}} \ell(\alpha, z) dP(z) - \frac{1}{m}\sum_{i=1}^{m} \ell(\alpha, z_i)\right| , \quad m = 1, 2, \ldots$$

and consider the convergence

$$\mathbb{P}_{\mathcal{D}\sim P^m}[\xi_m > \varepsilon] \xrightarrow[m\to\infty]{} 0, \quad \forall \varepsilon > 0 \tag{4.9}$$

which means that, for a large enough number of training examples m, the empirical risk approximates the true risk well uniformly over all the functions in the hypothesis class $\mathcal{H}(\Lambda)$.

Observation 4.3. The key theorem includes only the condition (4.8) because the concern here is to obtain consistency in *minimizing* the empirical risk but not in *maximazing* it. Since (4.9) is stronger than (4.8), the uniform convergence (4.9) is *sufficient* for the consistency of the ERM method.

4.3.2. Bounds on the rate of convergence

The goal of this part is to obtain non-asymptotic bounds on the rate of uniform converge of means to their mathematical expectations over a given set of hypothesis functions in the sense of (4.9). This way, we do not only determine how fast the asymptotic rate of convergence is but also whether the ERM method is consistent for that class of hypothesis thanks to Theorem 4.1. In other words, the generalization error will be bounded uniformly for all hypothesis and the obtained bounds will indicate the generalization ability of the learning machines operating under the ERM rule.

The analysis carried out is focused on the binary classification problem, but it can be extended to other type of problems and real-valued functions, as it is done in [1]. The added difficulty in the analysis of those cases is not worth the minor additions it provides to the analysis of binary functions.

Binary classification Problem

Binary classification, also known as pattern recognition, is a supervised learning problem in which the supervisor's response y takes only two values, that is, $y \in \{0,1\}$, and the hypothesis functions $h(x,\alpha)$, with $\alpha \in \Lambda$ and x a given input, are a set of indicator functions. Then, the loss function is the following:

$$\ell(\alpha,(x,y)) = \begin{cases} 0 & if \quad y = h(x,\alpha) \\ \\ 1 & if \quad y \neq h(x,\alpha) \end{cases} \tag{4.10}$$

The true error in this case is called *classification error*.

All in all, for the analysis conducted in this section, we shall consider an input space $\mathcal{X} \subset \mathbb{R}^N$, the output space $\mathcal{Y} = \{0,1\}$ (now $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$) and the hypothesis class $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and $(X,Y) : \Omega \to \mathcal{X} \times \mathcal{Y}$ a random vector with probability distribution P. We shall assume that our training set $\mathcal{D} = \{(x_1,y_1),\cdots,(x_m,y_m)\}$ is a realization of a simple random sample of (X,Y).

4.3.2.1. Finite Hypothesis Classes

In the case where the hypothesis class has a finite number of functions it is easy to determine a bound for the generalization error by means of the *Hoeffding's Inequality* that is presented below.

Lemma 4.2. (Hoeffding's Inequality) *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and Z a random variable defined over it with probability distribution P such that $\mathbb{P}[a \leq Z \leq b] = 1$, for some constants a and b. Let $Z_1,\cdots,Z_m$ be a simple random sample of Z with $\mathbb{E}[Z_i] = \mu$, $\forall i = 1,\cdots,m$. Then, for any $\varepsilon > 0$*

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m} Z_i - \mu\right| > \varepsilon\right] \leq 2\exp\left(\frac{-2m\varepsilon^2}{(b-a)^2}\right)$$

Proposition 4.2. *Under the assumption that the set $\Lambda = \{\alpha_1 \cdots, \alpha_N\}$ is finite, it holds that, for all $\varepsilon > 0$,*

$$\mathbb{P}_{\mathcal{D}\sim P^m}[\max_{\alpha\in\Lambda}|L_P(\alpha) - L_{\mathcal{D}}(\alpha)| > \varepsilon] \leq 2N\exp(-2m\varepsilon^2) \tag{4.11}$$

Proof. We start by considering the loss function $\ell(\alpha_i,\cdot) : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, for each fixed $\alpha_i \in \Lambda, i = 1,\cdots,N$, defined as in (4.10). Since (X,Y) is a random vector and α_i is fixed, it is clear that $\ell(\alpha_i,(X,Y))$ is a random variable defined over the space $(\Omega,\mathcal{F},\mathbb{P})$ which follows a Bernoulli probability distribution, so $\mathbb{P}[0 \leq \ell(\alpha_i,(X,Y)) \leq 1] = 1$. Let us denote $\ell_j(\alpha_i) = \ell(\alpha_i,(x_j,y_j))$, $j = 1,\cdots,m$, so that $\ell_1(\alpha_i),\cdots,\ell_m(\alpha_i)$ is a simple random sample of the random variable $\ell(\alpha_i,(X,Y))$ for being $(x_1,y_1),\cdots,(x_m,y_m)$ a simple random sample of (X,Y), with $\mathbb{E}_{(x_j,y_j)\sim P}[\ell_j(\alpha_i)] = L_P(\alpha_i)$, $\forall j = 1,\cdots,m$. Since $L_{\mathcal{D}}(\alpha_i) = \frac{1}{m}\sum_{j=1}^{m}\ell_j(\alpha_i)$, by applying Lemma 4.2 we get for any $\varepsilon > 0$

$$\mathbb{P}_{\mathcal{D}\sim P^m}[|L_P(\alpha_i) - L_{\mathcal{D}}(\alpha_i)| > \varepsilon] \leq 2\exp(-2m\varepsilon^2). \tag{4.12}$$

From the following chain of inequalities

$$\begin{aligned}\mathbb{P}[\max_{\alpha\in\Lambda}|L_P(\alpha)-L_{\mathcal{D}}(\alpha)|>\varepsilon] &\leq \mathbb{P}[|L_P(\alpha_1)-L_{\mathcal{D}}(\alpha_1)|>\varepsilon \\ &\quad or\ \ |L_P(\alpha_2)-L_{\mathcal{D}}(\alpha_2)|>\varepsilon \\ &\quad \dots \\ &\quad or\ \ |L_P(\alpha_N)-L_{\mathcal{D}}(\alpha_N)|>\varepsilon] \\ &= \mathbb{P}\left[\bigcup_{i=1}^{N}|L_P(\alpha_i)-L_{\mathcal{D}}(\alpha_i)|>\varepsilon\right] \\ &\leq \sum_{i=1}^{N}\mathbb{P}\left[|L_P(\alpha_i)-L_{\mathcal{D}}(\alpha_i)|>\varepsilon\right] \\ &\leq 2N\exp(-2m\varepsilon^2)\end{aligned}$$

the desired result is obtained, where all the probabilities are calculated over the choice of the m training examples. In the first inequality we have taken into account the fact that the event $\{\max_{\alpha\in\Lambda}|L_P(\alpha)-L_{\mathcal{D}}(\alpha)|>\varepsilon\}$ is included in $\left\{\bigcup_{i=1}^{N}|L_P(\alpha_i)-L_{\mathcal{D}}(\alpha_i)|>\varepsilon\right\}$ and in the last inequality we have applied (4.12) to each term of the addition. □

From the above proposition it is clear that the uniform convergence (4.9) holds because $2N\exp(-2m\varepsilon^2)\xrightarrow[m\to\infty]{}0$ and the rate of convergence is exponential in the number of training examples. Moreover, as the probability distribution P is arbitrary, the uniform convergence is valid for any probability distribution over $\mathcal{Z}$. As a consequence, the ERM principle is consistent for any finite class of hypothesis for any probability distribution over $\mathcal{Z}$ and, therefore, it is a successful agnostic PAC-learner.

Let us fix a confidence level $\delta\in(0,1)$, so that $\delta=2N\exp(-2m\varepsilon^2)$. We can then solve with respect to ε to obtain $\varepsilon=\sqrt{\frac{1}{2m}\ln(\frac{2N}{\delta})}$ and assert that, with probability of at least $1-\delta$,

$$|L_P(\alpha)-L_{\mathcal{D}}(\alpha)|\leq\sqrt{\frac{1}{2m}\ln\left(\frac{2N}{\delta}\right)},\ \ \forall\alpha\in\Lambda. \tag{4.13}$$

Additionally, we get that for any number of examples $m\geq\frac{\ln(2N/\delta)}{2\varepsilon^2}$, it is true that

$$\mathbb{P}_{\mathcal{D}\sim P^m}[|L_P(\alpha)-L_{\mathcal{D}}(\alpha)|\leq\varepsilon]\geq 1-\delta,\ \ \forall\alpha\in\Lambda. \tag{4.14}$$

Let us assume that $\alpha_m\in\Lambda$ is a parameter such that $\ell(\alpha_m,(x,y))$ minimizes the empirical risk over $\mathcal{D}$. Then it holds with probability at least $1-\delta$ that, for all $m\geq\frac{\ln(2N/\delta)}{2\varepsilon^2}$,

$$L_P(\alpha_m)\leq L_{\mathcal{D}}(\alpha_m)+\varepsilon\leq L_{\mathcal{D}}(\alpha)+\varepsilon\leq L_P(\alpha)+\varepsilon+\varepsilon=L_P(\alpha)+2\varepsilon,\ \ \forall\alpha\in\Lambda,$$

where in the first and third inequalities we have applied (4.14) and the second one is due to the fact that for α_m the empirical risk is minimum. Therefore, as the above chain of inequalities is true for any $\alpha\in\Lambda$, in particular,

$$\mathbb{P}_{\mathcal{D}\sim P^m}[L_P(\alpha_m)-\min_{\alpha\in\Lambda}L_P(\alpha)\leq\varepsilon]\geq 1-\delta,\ \ \forall m\geq\frac{2\ln(2N/\delta)}{\varepsilon^2}$$

where the value of ε has been substituted by $\varepsilon/2$.

Corollary 4.2. *Under the assumption that the set $\Lambda = \{\alpha_1 \cdots, \alpha_N\}$ is finite, the following statements are true:*

1. *For any $\delta \in (0,1)$, with probability of at least $1-\delta$, it holds that*

$$L_P(\alpha) \leq L_{\mathcal{D}}(\alpha) + \sqrt{\frac{1}{2m} \ln\left(\frac{2N}{\delta}\right)}, \quad \forall \alpha \in \Lambda. \tag{4.15}$$

2. *The ERM method is strictly consistent for any probability distribution P over the domain $\mathcal{X} \times \mathcal{Y}$.*

3. *The hypothesis class $\mathcal{H}(\Lambda)$ is agnostic PAC learnable for any learning algorithm using the ERM method with sample complexity*

$$m_\Lambda(\varepsilon, \delta) \leq \left\lceil \frac{2\ln(2N/\delta)}{\varepsilon^2} \right\rceil \tag{4.16}$$

Inequality (4.15) provides an upper bound for the true error in terms of the empirical error. The important fact is that it decreases as the size of the training set increases. Note that (4.13) can be used to obtain a lower bound as well, but it is not as important for learning theory as the upper bound. Both (4.15) and (4.16) increase logarithmically with the number of parameters in the set Λ and the inverse of the desired confidence level δ. The sample complexity (4.16) grows in addition with the square of $1/\varepsilon$.

4.3.2.2. Infinite Hypothesis Classes and the VC dimension

The obtained bound in the previous section depends on the number of parameters in the set Λ, which presents two main drawbacks: first, it can lead to a quite loose bound and, second, when the hypothesis class is not finite (which is the most usual case), the bound goes to infinity and it is meaningless. Therefore, a new concept which measures the capacity or complexity of a class of hypothesis must be introduced to replace N in (4.11). This new concept is called the Vapnik and Chervonenkis (VC) dimension and determines the number of different elements of $\mathcal{X}$ that can be completely discriminated using the functions in the class $\mathcal{H}(\Lambda)$.

Definition 4.6. (Dichotomies generated by a hypothesis class) Let $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ be a class of hypothesis functions and $C = \{x_1, \cdots, x_m\} \subset \mathcal{X}$. The dichotomies generated by $\mathcal{H}(\Lambda)$ on the subset of points C are defined by:

$$\mathcal{H}_C(\Lambda) = \{(h(x_1,\alpha), \cdots, h(x_m,\alpha)) : \alpha \in \Lambda\}$$

where each element of $\mathcal{H}_C(\Lambda)$ is a vector in $\{0,1\}^m$.

In other words, the dichotomies are the different ways in which the points of the subset C can be separated using the functions in $\mathcal{H}(\Lambda)$.

For each subset $C \subset \mathcal{X}$ there are $2^{|C|}$ possible dichotomies. We say that the class $\mathcal{H}(\Lambda)$ *shatters* the subset C if every possible dichotomy can be represented by a function of $\mathcal{H}(\Lambda)$. The ability of the hypothesis class to shatter a set of points can be seen, therefore, as a measure of its capacity.

Definition 4.7. (Growth function) Let $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ be a class of hypothesis functions. The growth function of $\mathcal{H}(\Lambda)$, denoted by $\Pi_\Lambda : \mathbb{N} \to \mathbb{N}$, is defined by

$$\Pi_\Lambda(m) = \max_{C \subset \mathcal{X}: |C| = m} |\mathcal{H}_C(\Lambda)|, \quad \forall m \in \mathbb{N},$$

where $|\cdot|$ denotes the number of elements of a set.

That is, $\Pi_\Lambda(m)$ indicates the maximum number of dichotomies that the hypothesis class $\mathcal{H}(\Lambda)$ is able to generate on a subset of $\mathcal{X}$ with m points. It is a purely combinatorial measure and does not depend on the underlying distribution P. It is always true that $\Pi_\Lambda(m) \leq 2^m$, $\forall m \in \mathbb{N}$.

Definition 4.8. (VC-dimension) The Vapnik-Chervonenkis (VC) dimension of a hypothesis class $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$, denoted by $d_{VC}(\Lambda)$, is the largest value of m for which $\Pi_\Lambda(m) = 2^m$. When $d_{VC}(\Lambda) = 2^m$ for all $m \in \mathbb{N}$ we say that $\mathcal{H}(\Lambda)$ has infinite VC-dimension.

Therefore, the VC-dimension is the size of the largest subset of $\mathcal{X}$ that can be completely shattered by $\mathcal{H}(\Lambda)$. We can say, informally, that the VC-dimension of a hypothesis class characterizes its diversity or richness. It is also a purely combinatorial notion. Note that, if $d_{VC}(\Lambda) = d$, there exists a subset $C \subset \mathcal{X}$ of size d that can be fully shattered, but this does not mean that every subset of size d (or less) can be shattered by $\mathcal{H}(\Lambda)$.

Theorem 4.2. *(*Sauer's Lemma*) Let $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ be a hypothesis class with $d_{VC}(\Lambda) = d < \infty$. Then, for all $m \in \mathbb{N}$,*

$$\Pi_\Lambda(m) \leq \sum_{i=0}^{d} \binom{m}{i}.$$

Corollary 4.3. *Let $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ be a hypothesis class with $d_{VC}(\Lambda) = d < \infty$. Then, for all $m \geq d$,*

$$\Pi_\Lambda(m) \leq \left(\frac{em}{d}\right)^d = O(m^d).$$

The above corollary shows that the growth function only presents two types of behavior: either $d_{VC}(\Lambda) = \infty$, in which case $\Pi_\Lambda(m) = 2^m \ \forall m \in \mathbb{N}$, or $d_{VC}(\Lambda) = d < \infty$, in which case,

$$\Pi_\Lambda(m) \begin{cases} = 2^m & if \quad m < d \\ \\ \leq \left(\frac{em}{d}\right)^d & if \quad m \geq d \end{cases}.$$

The proofs of both of the above results can be found in [16].

Now we can formulate the fundamental result of the Vapnik and Chervonenkis theory, a bound for the generalization error in terms of the growth function (or, equivalently, the VC-dimension), whose proof is explained in detail in [6].

Theorem 4.3. *(*VC Inequality*) For all $\varepsilon > 0$ it holds:*

$$\mathbb{P}_{\mathcal{D} \sim P^m}[\sup_{\alpha \in \Lambda} |L_P(\alpha) - L_{\mathcal{D}}(\alpha)| > \varepsilon] \leq 4\Pi_\Lambda(2m) \exp\left(-\frac{1}{8}\varepsilon^2 m\right) \tag{4.17}$$

The above inequality is valid for any target binary function and any input probability distribution.

The fact that the growth function is bounded by a polynomial of order $O(m^d)$, provided that the VC dimension is finite, makes it possible that the term in the right hand side of (4.17) converges to zero as the size of the training sample m increases, in such a way that the uniform convergence (4.9) holds and the rate of convergence is exponential.

By selecting a confidence level $\delta \in (0,1)$, so that $\delta = 4\Pi_\Lambda(2m)\exp\left(-\frac{1}{8}\varepsilon^2 m\right)$, and solving for ε the following corollary is obtained:

Corollary 4.4. *(*VC generalization bound*) For any $\delta \in (0,1)$ it holds, with probability at least $1-\delta$,*

$$L_P(\alpha) \leq L_{\mathcal{D}}(\alpha) + \sqrt{\frac{8}{m}\ln\left(\frac{4\Pi_\Lambda(2m)}{\delta}\right)}, \quad \forall \alpha \in \Lambda. \tag{4.18}$$

As in the case of finite hypothesis classes, the upper bound for the true error depends on the empirical risk, the size of the training set (it decreases as the training set increases in size), and also dependes logarithmically on $1/\delta$. The number of hypothesis functions in the bound (4.15) has been replaced by the new alternative concept measuring the complexity of the hypothesis class, that is, the growth function, so that now the bound depends logarithmically on the growth function.

Thanks to Corollary 4.3, we can also bound the true error in terms of the VC dimension. That is, if the VC dimension d is finite, with probability at least $1-\delta$, it is true that

$$L_P(\alpha) \leq L_{\mathcal{D}}(\alpha) + \sqrt{\frac{8d\ln\frac{2em}{d} + 8\ln\frac{4}{\delta}}{m}} = L_{\mathcal{D}}(\alpha) + O\left(\sqrt{\frac{\ln(m/d)}{(m/d)}}\right), \quad \forall \alpha \in \Lambda.$$

With this reformulation of the bound we notice the importance of the ratio of the number of training examples to the VC dimension, $\frac{m}{d}$, for the generalization ability of a learning machine.

We have seen that the finiteness of the VC dimension of a hypothesis class is *sufficient* for uniform convergence (4.9). The following theorem, whose proof can be found in [1], shows that it is also *necessary*.

Theorem 4.4. *Let $\mathcal{X} \subset \mathbb{R}^N$ be an input space, $\mathcal{Y} = \{0,1\}$ the output space, $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ a hypothesis class and ℓ the 0-1 loss function (4.10). For existence of uniform convergence in the sense of (4.9) with respect to any probability distribution P over $\mathcal{X} \times \mathcal{Y}$ it is necessary and sufficient that the hypothesis class $\mathcal{H}(\Lambda)$ has a finite VC-dimension.*

The above theorem, together with all the previous results, leads immediately to the following:

Theorem 4.5. *(*The fundamental theorem of PAC learning*) Let $\mathcal{X} \subset \mathbb{R}^N$ be an input space, $\mathcal{Y} = \{0,1\}$ the output space, $\mathcal{H}(\Lambda) = \{h(x,\alpha), \alpha \in \Lambda\}$ a hypothesis class and ℓ the 0-1 loss function (4.10). The following statements are equivalent:*

1. *The uniform convergence (4.9) over the set of functions $\ell(\alpha,(x,y))$, $\alpha \in \Lambda$, holds with respect to any probability distribution P over $\mathcal{X} \times \mathcal{Y}$.*
2. *The ERM method is strictly consistent for any probability distribution P over the domain $\mathcal{X} \times \mathcal{Y}$.*

3. *The ERM method is a successful agnostic PAC-learner for* $\mathcal{H}(\Lambda)$.
4. *The hypothesis class* $\mathcal{H}(\Lambda)$ *is agnostic PAC learnable for any learning algorithm using the ERM method.*
5. *The hypothesis class* $\mathcal{H}(\Lambda)$ *is PAC learnable for any learning algorithm using the ERM method.*
6. *The ERM method is a successful PAC-learner for* $\mathcal{H}(\Lambda)$.
7. $\mathcal{H}(\Lambda)$ *has a finite VC dimension.*

All the implications between the statements of the theorem have been discussed so far, except for $5 \to 7$ and $6 \to 7$. These implications are a direct consequence of the following proposition:

Proposition 4.3. *Let* $\mathcal{H}(\Lambda) = \{h(x, \alpha), \alpha \in \Lambda\}$ *be a hypothesis class with infinite VC dimension. Then,* $\mathcal{H}(\Lambda)$ *is not PAC learnable.*

Observation 4.4. The fundamental theorem of PAC learning can be adapted to other learning problems such as regression, but it does not hold for all learning tasks.

Until now we have not mentioned anything about the sample complexity in the case of an infinite hypothesis class. The reason is that in this case it is not as easy to determine as for a finite hypothesis class. The following proposition covers this issue. The proof, which is quite tedious, is presented in [5].

Proposition 4.4. *Let* $\mathcal{X} \subset \mathbb{R}^N$ *be an input space,* $\mathcal{Y} = \{0, 1\}$ *the output space,* $\mathcal{H}(\Lambda) = \{h(x, \alpha), \alpha \in \Lambda\}$ *a hypothesis class and* ℓ *the o-1 loss function* (4.10)*. Assume that* $d_{VC}(\Lambda) = d < \infty$*. Then, there exists a constant* C *such that* $\mathcal{H}(\Lambda)$ *is agnostic PAC learnable with sample complexity*

$$m_\Lambda(\varepsilon, \delta) \leq C \frac{d + \ln(1/\delta)}{\varepsilon^2}$$

Note that, as in the case of finite hypothesis classes, the sample complexity depends logarithmically on $1/\delta$ and grows with $1/\varepsilon^2$. It is also influenced by the VC dimension in such a way that the sample complexity increases with d.

4.4. Approximation-Generalization tradeoff

At this point, we may ask how we should choose a good hypothesis class among all the classes that have finite VC dimension. Of course, we would like that the smallest true error achievable by a hypothesis from this class is as small as possible, but at the same time we would like that the VC dimension of that hypothesis class is small so that the generalization bound (4.18) does not become too big. Here comes a tradeoff known as the *approximation-generalization* tradeoff (or the *bias-variance* tradeoff).

The minimal possible error obtained by any function defined over the given domain $\mathcal{X}$ is called *Bayes error* and it is the minimal inevitable error due to the possible nondeterminism of the problem setting. It is the error associated with the Bayes optimal predictor (see Section 3.2.1 [5]), and we will denote it by L^*.

Let $(\mathcal{Z}, \Lambda, \ell)$ be a learning task, P a probability distribution over $\mathcal{Z}$ and $\mathcal{D} = \{z_1, \ldots, z_m\} \subset \mathcal{Z}^m$ a set of examples i.i.d. generated by P. Then, the difference between the true error of any

hypothesis $h(\cdot, \alpha)$, $\alpha \in \Lambda$, and the Bayes error can be decomposed as follows:

$$L_P(\alpha) - L^* = \underbrace{\left(L_P(\alpha) - \inf_{\alpha' \in \Lambda} L_P(\alpha')\right)}_{\text{estimation error}} + \underbrace{\left(\inf_{\alpha' \in \Lambda} L_P(\alpha') - L^*\right)}_{\text{approximation error}}$$

The second term of this decomposition is called the **approximation error** and it measures the excess over the Bayes error of the minimum risk achievable by a hypothesis of the class $\mathcal{H}(\Lambda)$. It depends on the richness of the class selected, in such a way that it decreases by choosing a more complex hypothesis class. This error comes from the *inductive bias* that appears because the learning machine is restricted to looking for a hypothesis in one specific class.

The first term is the **estimation error** and it determines the quality of the selected hypothesis, which under the ERM rule is that which minimizes the empirical risk, with respect to the best hypothesis in the class in terms of the true error. In the case of the ERM principle, it appears because the empirical risk is just an estimate of the true risk, and so the hypothesis selected minimizing the empirical risk is not necessarily the best in the class with respect to the true risk. The estimation error can be bounded in terms of the generalization error. Indeed, let us assume that $\alpha_m \in \Lambda$ is a parameter such that $\ell(\alpha_m, z)$ minimizes the empirical risk over $\mathcal{D}$ and $\alpha^* = argmin_{\alpha' \in \Lambda} L_P(\alpha')$. Then,

$$\begin{aligned} L_P(\alpha_m) - L_P(\alpha^*) &= L_P(\alpha_m) - L_{\mathcal{D}}(\alpha_m) + L_{\mathcal{D}}(\alpha_m) - L_P(\alpha^*) \\ &\leq L_P(\alpha_m) - L_{\mathcal{D}}(\alpha_m) + L_{\mathcal{D}}(\alpha^*) - L_P(\alpha^*) \\ &\leq 2 \sup_{\alpha \in \Lambda} |L_P(\alpha) - L_{\mathcal{D}}(\alpha)| \end{aligned} \tag{4.19}$$

Going back to the binary classification problem and (4.17), let us suppose that $d_{vc}(\Lambda) = d < \infty$ and denote

$$\Omega(m, d, \varepsilon) = 4\Pi_\Lambda(2m)\exp\left(-\frac{1}{8}\varepsilon^2 m\right) \leq 4\left(\frac{em}{d}\right)^d \exp\left(-\frac{1}{8}\varepsilon^2 m\right).$$

From the above chain of inequalities (4.19), relating the estimation and the generalization error, and (4.17), it is clear that the estimation error is also bounded by $\Omega(m, d, \varepsilon)$. Therefore, the estimation error depends on the sample size, the VC dimension of the hypothesis class and the accuracy parameter ε.

In order to minimize the difference between the Bayes error and the true error of any hypothesis, it is necessary to minimize at the same time the approximation and the estimation error. However, that is not possible and we must find a tradeoff, called the *approximation-generalization* tradeoff:

- On the one hand, if we choose a very complex hypothesis class, the approximation error is likely to decrease considerably. Nonetheless, the value $\Omega(m, d, \varepsilon)$ increases as the VC dimension grows, leading to a high estimation error (and high generalization error).
- On the other hand, selecting a hypothesis class with a low VC dimension would make the bound $\Omega(m, d, \varepsilon)$ small, so the generalization error, and, consequently, the estimation error, would decrease. But at the same time the approximation error would increase as a result of a simpler hypothesis class.

In short, if $\mathcal{H}(\Lambda)$ is too simple, it may fail to approximate the target distribution well, and if it is too complex it may not generalize well. Therefore, to select a good hypothesis class, we must find a balance between those two conflicting goals. As shown in Figure 4.2, the minimum true risk is obtained at some intermediate point d^*_{VC}.

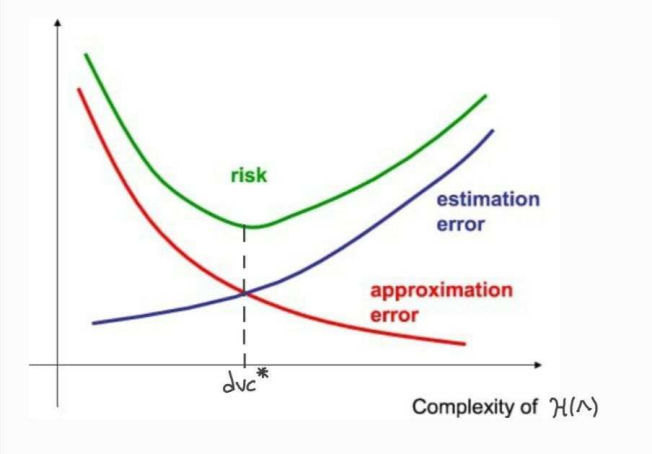


Figure 4.2.: Approximation-Generalization Tradeoff [6].

One possible solution to this problem is the use of some prior knowledge about the learning problem at hand. Thanks to this prior knowledge we might be able to select a hypothesis class with both small approximation and estimation errors. For example, we know that for the segmentation problem considered in this work convolutional neural networks work fine, and, in particular, the Unet architecture provides the best results, so we restrict ourselves to using this model.

4.5. Overfitting

In this section we shall describe the most common problem that the ERM rule entails, namely *overfitting*. This phenomenon takes place when the selected hypothesis fits the training data extremely well, in the sense that the empirical risk associated with that hypothesis is so low (maybe even zero), but the true risk of that hypothesis is high. Therefore, the empirical risk is no longer a good estimate of the true error. In this case, we can intuitively think that the model has memorized the supervisor's response to every example in the training set and it fits even the possible noise in the training data. In other words, when overfitting occurs the generalization error and the estimation error are high.

Overfitting is more likely when the hypothesis class is complex (large VC dimension) and the number or training examples is small, because in that case the term $\Omega(m, d, \varepsilon)$ in the right hand side of the VC inequality (4.18) is high, and, thus, the generalization error is big. Therefore, here we find again the tradeoff discussed in the previous section: a more complex hypothesis class increases the risk of overfitting but leads to a lower empirical error and a less complex class decreases the probability of overfitting but increases the empirical error.

[6] Image obteined from https://www.jobilize.com/course/section/competing-goals-the-bias-variance-tradeoff-by-openstax.

The ability to deal with overfitting is one of the key skills in machine learning. Numerous techniques have been developed to avoid overffiting. Such techniques constrain the capacity of the hypothesis class with the objective of decreasing the generalization (or estimation) error while still providing a low empirical error. They are known as *regularization techniques*. The most popular one consists in minimizing a combination of the empirical risk and a measure of the complexity of the hypothesis at hand, in such a way that the learning algorithm is constrained to fit the training data properly using a simpler hypothesis. That is, instead of minimizing the empirical risk, the learning algorithm has to minimize the following sum:

$$L_{\mathcal{D}}(\alpha) + \lambda\Omega(\alpha), \quad \alpha \in \Lambda,$$

where $\Omega(\alpha)$ is a measure of the complexity of the hypothesis $h(\cdot, \alpha)$ and $\lambda > 0$ is a regularization parameter which controls the tradeoff between the empirical error and the complexity of the hypothesis (the bigger λ the simpler hypothesis we will get). In conjuction, $\lambda\Omega(\alpha)$ is called the *regularization (or penalty) term*. For example, if $\Lambda \subset \mathbb{R}^N$ a measure of the complexity could be any norm in $\mathbb{R}^N$. By choosing $\Omega(\alpha) = \|\alpha\|_2^2$, we obtain the **weight decay** regularization.

Other regularization techniques commonly used for neural networks are **dropout** or **early stopping**.

4.6. The test set

In subsection 4.3.2 we have obtained bounds for the true error of any hypothesis. We can use those bounds as an estimate of the true error of the final hypothesis selected based on the empirical error. However, those bounds are quite loose so the estimate is not accurate enough.

One possible solution is to estimate the true error based on a *test set*, that is, a set of examples that were not used to select the final hypothesis. In this way, the final hypothesis given by the learning algorithm is evaluated on the test set, i.e, its empirical error is calculated over the test set, and that value is taken as an estimate of the true error of the hypothesis. Let us call that value L_{test}.

As far as the test set is concerned there exists only one hypothesis, that is the final hypothesis selected by the learning algorithm and it will not change if the test set changes since this set does not influence in the selection of the hypothesis. Therefore, the simple Hoeffding inequality for one hypothesis (4.12) holds in this case. Assuming the binary classification problem setting, and with $\alpha^* \in \Lambda$ being the final parameter chosen by the learner, for any $\delta \in (0, 1)$ it is true with probability at least $1 - \delta$, that

$$L_P(\alpha^*) \leq L_{\text{test}}(\alpha^*) + \sqrt{\frac{1}{2m} \ln\left(\frac{2}{\delta}\right)}.$$

This bound, which decreases as the number of examples, m, in the test set increases, is much tighter than the VC generalization bound (4.18), and, therefore, it provides a better estimate of the true error. Note that, in the moment when the test set affects the choice of the final hypothesis, the previous bound does not hold any more.

Another positive aspect is that the empirical error over the test set is an unbiased estimate of the true error, since it does not have an optimistic or pessimistic bias, as opposed to the

empirical error over the training set, which has an optimistic bias because the best hypothesis was selected according to a small error over the training data.

Nevertheless, as the examples used for testing cannot be used for training and the available data is limited, setting aside a test set leads to fewer examples for training, and, as a consequence, to a worse final hypothesis. Thus, it is important to find a compromise between the number of examples used for testing and for training so that the final hypothesis is good enough and the estimate of the true error based on the test set is reliable. A rule of thumb is to use the 20% of the avaliable data for testing and the remaining 80% for training.

5. Optimization

The learning problem is originally formulated as an optimization problem. Therefore, in this chapter we will study optimization in detail focusing on convex functions and unconstrained problems, since the minimization problem in the learning framework presents no additional constrains. Although the functions to minimize in learning are frequently not convex, convex optimization is the basis for several heuristics for solving nonconvex problems.
We will proof convergence of the main optimization algorithm used in learning, namely gradient descent, for convex and L-smooth functions, after presenting the definition and properties of these two types of functions. We explained this with the help of [7],[17] a [18].
To finish the chapter we will describe the application of gradient descent to the learning problem, together with some of its important variants common in machine learning, following [8] (Chapter 8) and [19].

5.1. Convex functions

5.1.1. Definition and basic properties

We begin this chapter by presenting the concept of convex functions, as well as some of their properties.

Convex functions must be defined over convex sets.

Definition 5.1. (Convex set) A set $C \subset \mathbb{R}^N$ is *convex* if for any $x, y \in C$ and any $\theta \in [0,1]$ it holds that $\theta x + (1-\theta)y \in C$. In words, a set is convex if the line segment joining any two points of the set lies inside the set.

Definition 5.2. (Convex function) Let $C \subset \mathbb{R}^N$ be a convex set. A function $f : C \to \mathbb{R}$ is *convex* if for all $x, y \in C$ and all $\theta \in [0,1]$, we have that

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{5.1}$$

From a geometrical point of view, when $N = 1$, the condition of a convex function means that the graph of the function f lies below the line segment between the point $(x, f(x))$ and $(y, f(y))$ (see Figure 5.1).

We say that the function f is *strictly convex* if the inequality in Def. 5.2 is strict for any two points $x \neq y$ in C and any $\theta \in (0,1)$.

If the inequality (5.1) holds in the opposite direction for a function f we say that f is a *concave function*. When the inequality is strict, the function f is *strictly concave*. It is clear that f is concave (respectively strictly concave) if, and only if, $-f$ is convex (respectively strictly convex).

The following result allows us to check whether a function is convex by restring it to a line that intersects is domain.

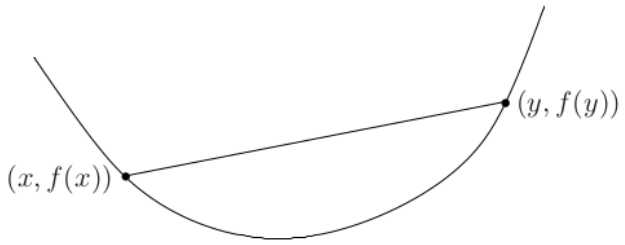


Figure 5.1.: Graph of a convex function in $\mathbb{R}$.

Proposition 5.1. *Let $C \subset \mathbb{R}^N$ be a convex set. A function $f : C \to \mathbb{R}$ is convex if, and only if, for all $x \in C$ and all $v \in \mathbb{R}^N$, the function $\phi : \mathbb{R} \to \mathbb{R}$, $t \mapsto f(x + tv)$, is convex on its domain $\{t : x + tv \in C\}$.*

In the following we will present two conditions for convexity of a function based on its gradient and Hessian matrix.

Theorem 5.1. *(*First order condition*) Let $f : C \to \mathbb{R}$, $C \in \mathbb{R}^N$, be a differentiable function. Then f is convex if, and only if, its domain C is convex and it holds*

$$f(y) \geq f(x) + \langle \nabla f(x), (y - x) \rangle, \quad \forall x, y \in C. \tag{5.2}$$

Proof. (5.1) $\Rightarrow$ (5.2) Dividing the inequality (5.1) by the scalar θ and rearranging the terms we obtain

$$\frac{f(y + \theta(x - y)) - f(y)}{\theta} \leq f(x) - f(y).$$

If we now take the limit when $\theta \to 0$ and apply the definition of the differential, we have the desired result :

$$\langle \nabla f(y), x - y \rangle \leq f(x) - f(y)$$

(5.2) $\Rightarrow$ (5.1) Let us consider the point $x_\theta = \theta x + (1 - \theta)y$, with $\theta \in [0, 1]$. From (5.2) we have that

$$f(x) \geq f(x_\theta) + \langle \nabla f(x_\theta), x - x_\theta \rangle = f(x_\theta) - (1 - \theta)\langle \nabla f(x_\theta), y - x \rangle$$
$$f(y) \geq f(x_\theta) + \langle \nabla f(x_\theta), y - x_\theta \rangle = f(y) - \theta\langle \nabla f(x_\theta), y - x \rangle$$

Multiplying the first inequality by θ and the second one by $(1 - \theta)$ and adding the results we obtain

$$\theta f(x) + (1 - \theta) f(y) \geq f(x_\theta).$$

□

From the first order condition inequality it is easy to deduce the following result:

Corollary 5.1. *Let $C \subset \mathbb{R}^N$ be a convex set and $f : C \to \mathbb{R}$ be a differentiable convex function. Then a point $x \in C$ is a global minimizer of the function f if, and only if, $\nabla f(x) = 0$.*

Theorem 5.2. *(*Second order condition*) Let $f : C \to \mathbb{R}$, $C \in \mathbb{R}^N$, be a twice differentiable function. Then f is convex if, and only if, its domain C is convex and its Hessian matrix is positive semidefinite.*

That is[1],

$$\langle \nabla^2 f(x)v, v\rangle \geq 0, \quad \forall x \in C, v \in \mathbb{R}^N. \tag{5.3}$$

Proof. We will proof this result by showing that the first and second order conditions are equivalent, that is, (5.2) $\Leftrightarrow$ (5.3).

(5.2) $\Rightarrow$ (5.3) Let us first consider the case $N = 1$. Given $x, y \in C, C \subset \mathbb{R}$, with $x < y$, from (5.2) we have

$$f(y) \geq f(x) + f'(x)(y - x)$$
$$f(x) \geq f(y) - f'(y)(x - y),$$

which implies

$$f'(x)(y - x) \leq f(y) - f(x) \leq f'(y)(y - x) \Rightarrow f'(x)(y - x) \leq f'(y)(y - x).$$

Dividing by $(y - x)^2 > 0$, we obtain

$$\frac{f'(y) - f'(x)}{y - x} \geq 0.$$

Since the reasoning also holds for $y < x$, the above inequality is true for all $x, y \in C, x \neq y$. By taking the limit as $y \to x$, we have that $f''(x) > 0$, for all $x \in C$.

We prove now the general case $N \in \mathbb{N}$, using Proposition 5.1. We know that if (5.2) is true, then the function f is convex, and, from Proposition 5.1, we have that the function $\phi(t) = f(x + tv)$, $t \in [0, 1]$, is convex for all $x \in C, v \in \mathbb{R}^N$. By applying the result obtained in dimension 1 to ϕ, we have that

$$\phi''(t) = \langle \nabla^2 f(x + tv)v, v\rangle \geq 0, \quad \forall x \in C, \forall v \in \mathbb{R}^N, \forall t \in \mathbb{R} \text{ such that } x + tv \in C.$$

Therefore, if f is convex, (5.2) is true and $\nabla^2 f(x)$ is positive semidefinite for all $x \in C$.

(5.3)$\Rightarrow$ (5.2) This implication is easy to proof by considering the function $\phi(t) = f(y + t(x - y))$ for each $t \in [0, 1], x, y \in C$, and the fundamental theorem of calculus[2]. The function $\phi : [0, 1] \to \mathbb{R}$ is differentiable for each t, with

$$\phi'(t) = \langle \nabla f(y + t(x - y)), x - y\rangle$$
$$\phi''(t) = \langle \nabla^2 f(y + t(x - y))(x - y), x - y\rangle.$$

Thanks to the fundamental theorem of calculus we can state the following

$$\begin{aligned} f(x) = \phi(1) &= \phi(0) + \int_0^1 \phi'(t)dt \\ &= \phi(0) + \phi'(0) + \int_0^1 (\phi'(t) - \phi'(0))dt \\ &= \phi(0) + \phi'(0) + \int_0^1 \int_0^t \phi''(s)ds. \end{aligned}$$

[1] By $\nabla^2 f(x)$ we denote the Hessian matrix and $\langle \cdot, \cdot \rangle$ denotes the scalar product.
[2] See Fundamental theorem of calculus.

Writing the above expression back in terms of the function f gives

$$f(x) = f(y) + \langle \nabla f(y), x - y \rangle + \int_0^1 \int_0^t \langle \nabla^2 f(y + s(x-y))(x-y), x-y \rangle ds dt \geq f(y) + \langle \nabla f(y), x - y \rangle,$$

where in the last inequality we have taken into account the fact that the Hessian matrix is positive semidefinite. □

Geometrically, the second order condition indicates that the graph of the function has non-negative curvature at every point x in the domain.

5.1.2. Smoothness

Here we describe a special kind of functions that will be needed for the gradient descent algorithm to convergence.

Definition 5.3. (L-smooth function) A differentiable function $f : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^N$, is said to be *L-smooth* if its gradient is Lipschitz continuous, that is, if there exists a real constant $L > 0$, such that

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{5.4}$$

The following two lemmas will be used in the proof of convergence of gradient descent.

Lemma 5.1. *If $f : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^N$, is an L-smooth function, then, for all $x, y \in \Omega$ it is true that*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|_2^2.$$

Proof. Let us consider $x, y \in \Omega$ and the differentiable function $\phi(t) = f(x + t(y - x), t \in [0, 1]$. By applying the fundamental theorem of calculus (FTC) to ϕ, the Cauchy-Schwartz inequality and the definition of L-smoothness, we obtain the expected result:

$$\begin{aligned}
f(y) = \phi(1) &\overset{FTC}{=} \phi(0) + \int_0^1 \phi'(t)dt \\
&= \phi(0) + \phi'(0) + \int_0^1 (\phi'(t) - \phi'(0))dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 \langle \nabla f(x + t(y-x)) - \nabla f(x), y - x \rangle dt \\
&\overset{Cauchy-Schwartz}{\leq} f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 \|\nabla f(x + t(y-x)) - \nabla f(x)\|_2 \|y - x\|_2 dt \\
&\overset{(5.4)}{\leq} f(x) + \langle \nabla f(x), y - x \rangle + L \int_0^1 t\|y - x\|_2^2 dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|_2^2.
\end{aligned}$$

□

Lemma 5.2. *Let $\Omega \subset \mathbb{R}^N$ be a convex set and $f : \Omega \to \mathbb{R}$ an L-smooth and convex function. Then, for all $x, y \in \Omega$ it holds*

1. $f(y) - f(x) \leq \langle \nabla f(y), y - x \rangle - \frac{1}{2L} \|\nabla f(y) - \nabla f(x)\|_2^2$
2. $\langle \nabla f(y) - \nabla f(x), y - x \rangle \geq \frac{1}{L} \|\nabla f(y) - \nabla f(x)\|_2^2$.

Proof. 1. Let us consider a fixed $z \in \Omega$. From Lemma 5.1 and the first order condition for convexity, we have

$$f(z) - f(x) \leq \langle \nabla f(x), z - x \rangle + \frac{L}{2} \|z - x\|_2^2. \quad \text{(Lemma 5.1)}$$

$$f(z) - f(y) \geq \langle \nabla f(y), z - y \rangle \overset{\times(-1)}{\Rightarrow} f(y) - f(z) \leq \langle \nabla f(y), y - z \rangle \quad (5.2)$$

Then, it follows that

$$\begin{aligned} f(y) - f(x) &= f(y) - f(z) + f(z) - f(x) \\ &\leq \langle \nabla f(y), y - z \rangle + \langle \nabla f(x), z - x \rangle + \frac{L}{2} \|z - x\|_2^2. \end{aligned} \quad (5.5)$$

Now we will minimize the right hand side of the above inequality in z, in order to obtain a tightest upper bound. The derivative of the right hand side with respect to z is given by

$$\nabla f(x) - \nabla f(y) + L(z - x),$$

which is equal to zero when $z = x - \frac{1}{L}(\nabla f(x) - \nabla f(y))$. Substituting this value in (5.5) gives the desired result:

$$f(y) - f(x) \leq \langle \nabla f(y), y - x + \frac{1}{L}(\nabla f(x) - \nabla f(y)) \rangle - \frac{1}{L} \langle \nabla f(x), \nabla f(x) - \nabla f(y) \rangle + \frac{L}{2} \| \frac{1}{L}(\nabla f(x) - \nabla f(y)) \|_2^2$$

$$\begin{aligned} &= \langle \nabla f(y), y - x \rangle - \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|_2^2 + \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|_2^2 \\ &= \langle \nabla f(y), y - x \rangle - \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|_2^2 \end{aligned}$$

2. Applying the recently proved inequality twice as follows

$$\begin{aligned} f(y) - f(x) &\leq \langle \nabla f(y), y - x \rangle - \frac{1}{2L} \|\nabla f(y) - \nabla f(x)\|_2^2 \\ f(x) - f(y) &\leq \langle \nabla f(x), x - y \rangle - \frac{1}{2L} \|\nabla f(y) - \nabla f(x)\|_2^2 \end{aligned}$$

and adding together both of the above inequalities we obtain

$$0 \leq \langle \nabla f(y) - \nabla f(x), y - x \rangle - \frac{1}{L} \|\nabla f(y) - \nabla f(x)\|_2^2,$$

as we wanted.

□

5.2. Convex optimization problem

An *unconstrained convex optimization problem* is one of the form

$$\text{minimize } f(x),$$

where $f : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^N$, is a convex and twice continuously differentiable function (what implies that its domain is open), called the *objective function*, and $x \in \Omega$ is called the *optimization variable*. The domain of the objective function, which is a convex set, is also known as the *domain* of the optimization problem. Therefore, the optimization problem consists in finding an x in the domain that minimizes $f(x)$ among all $x \in \Omega$. The *optimal value* of the problem is denoted by $p^* = \inf_{x\in\Omega} f(x)$.

Definition 5.4. (Optimal and locally optimal points) We say that a point $x^* \in \Omega$ is *optimal* if $f(x^*) = p^*$. We say that the problem is *solvable* when there exists an optimal point for the problem. If a point $x \in \Omega$ fulfills that $f(x) \leq p* + \varepsilon$, with $\varepsilon > 0$, it is called an *ε-suboptimal* point.

We say that a point $x \in \Omega$ is *locally optimal* if there exists an $R > 0$ such that

$$f(x) = \inf\{f(z) : \|z - x\|_2 \leq R, z \in \Omega\}.$$

In words, x is a locally optimal point if it minimizes f over all close points in the domain.

We will assume in the following that the considered optimization problem is solvable.

Proposition 5.2. *Any locally optimal point of a convex function is also a (globally) optimal point.*

Proof. Let us assume that $x \in \Omega$ is a locally optimal point of a convex function $f : \Omega \to \mathbb{R}$. Then,

$$f(x) = \inf\{f(z) : \|z - x\|_2 \leq R, z \in \Omega\},$$

for certain $R > 0$. Now suppose that there exists $y \in \Omega$ such that $f(y) < f(x)$, i.e., x is not a globally optimal point. It must be $\|y - x\|_2 > R$, because otherwise it would be $f(y) > f(x)$ for being x locally optimal. Let us consider the point $z = (1 - \theta)x + \theta y$, with $\theta = \frac{R}{2\|y-x\|_2}$, so $\|z - x\|_2 = R/2 < R$. It is clear that $z \in \Omega$ because the domain is a convex set and

$$f(z) \leq (1 - \theta)f(x) + \theta f(y) < f(x),$$

which contradicts the fact that x is locally optimal. Therefore, there cannot exist a point $y \in \Omega$ with $f(y) < f(x)$ and x is a globally optimal point. □

Since the objective function is convex and differentiable, we have seen before (see Corollary 5.1) that solving the given unconstrained optimization problem is equivalent to finding a solution of

$$\nabla f(x^*) = 0,$$

which is a set of N equations in N variables. Sometimes it is possible to find a solution of the set of equations analytically, but it is not often the case. Hence we need to use and iterative algorithm, that is, an algorithm that calculates a sequence of points $x^{(0)}, x^{(1)}, \ldots$ in the domain such that $f(x^{(k)}) \overset{k\to\infty}{\to} p^*$. That sequence is known as a minimizing sequence for the optimization problem. The algorithm stops when $f(x^{(k)}) - p^* \leq \varepsilon$, for some specified tolerance $\varepsilon > 0$.

Definition 5.5. (Sublevel set) If $x^{(0)} \in \Omega$ is the starting point of an iterative algorithm, the *sublevel set* associated with $x^{(0)}$ is defined as

$$S = \{x \in \Omega : f(x) \leq f(x^{(0)})\}.$$

The iterative algorithms described above require a starting point $x^{(0)} \in \Omega$ such that the sublevel set associated with it is closed. This condition is always true for any starting point in the domain whenever the objective function f is continuous and its domain is $\mathbb{R}^N$.

5.3. Gradient Descent Algorithm

The iterative algorithms mentioned in the previous section produce a minimizing sequence $x^{(k)}, k = 1, 2, \ldots$, of the form

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)},$$

where $k = 1, 2, \ldots$ denotes the iteration number, $t^{(k)} > 0$ (except when $x^{(k)}$ is an optimal point, in which case $t^{(k)} = 0$) is a scalar called the *step size* at iteration k and $\Delta x^{(k)}$ is a vector in $\mathbb{R}^N$ called the *search direction*.

In the case of *descent algorithms* it holds that

$$f(x^{(k+1)}) < f(x^{(k)}),$$

except when $x^{(k)}$ is optimal. Therefore, for any iteration k, we have that $x^{(k)} \in S$, being S the initial sublevel set and, in particular, $x^{(k)}$ is in the domain of the objective function f.

Roughly speaking, the gradient indicates the direction of maximum slope of the graph of a function, that is, the direction of fastest increase of the function. Therefore, in order to obtain in each iteration smaller values of the function f to minimize, a natural choice of the search direction is the negative gradient: $\Delta x = -\nabla f(x)$. The iterative descent algorithm that uses this search direction is known as *gradient descent algorithm*. It is described in Algorithm 1.

Algorithm 1: Gradient Descent

Input: Objective function f, step size η, starting point $x^{(0)}$, tolerance ε
$k \leftarrow 0$
repeat
  $x^{(k+1)} \leftarrow x^{(k)} - \eta \nabla f(x^{(k)})$
  $k \leftarrow k + 1$
until *stopping criterion is met*
return $x^{(k)}$, $f(x^{(k)})$

The step size η, also known as the *learning rate*, could be the same for every iteration (i.e. it is fixed beforehand) or change with the value of the gradient and the iteration number, as we will see later. Its selection plays a crucial role in the convergence of the algorithm. The stopping criterion of the algorithm is usually, but not only, of the form $\|\nabla f(x^{(k)})\|_2 < \varepsilon$, where $\varepsilon > 0$ is a small value that determines the tolerance of the returned solution.

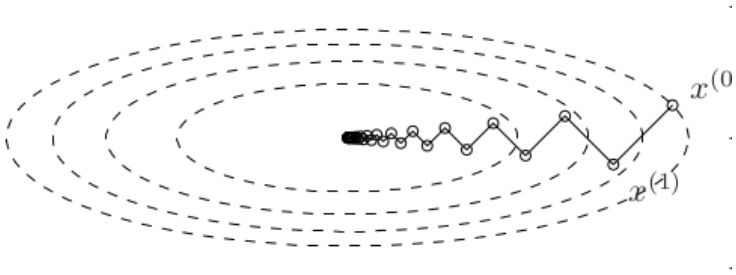


Figure 5.2.: Iterates of the gradient descent algorithm in $\mathbb{R}^2$. The dashed lines are level curves of the objective function.

5.3.1. Analysis of convergence

We will prove sublinear convergence of the gradient descent algorithm for convex and L-smooth functions. Linear convergence can be proven for *strong convex functions* (see [17]), a stronger notion of convexity, but we will not go into detail in that case.

Theorem 5.3. *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a convex and L-smooth function. Let $x^{(k)}, k = 0, 1, 2, \ldots$, be the minimizing sequence produced by the gradient descent algorithm with fixed step size $\frac{1}{L} \geq \eta > 0$, and $x^* \in \mathbb{R}^N$ be an optimal point. Then,*

$$f(x^{(k)}) - f(x^*) \leq \frac{2\|x^{(0)} - x^*\|_2^2}{\eta k}.$$

Proof. It is true that

$$\begin{aligned} \|x^{(k+1)} - x^*\|_2^2 &= \|x^{(k)} - x^* - \eta \nabla f(x^{(k)})\|_2^2 \\ &= \|x^{(k)} - x^*\|_2^2 - 2\eta \langle x^{(k)} - x^*, \nabla f(x^{(k)}) \rangle + \eta^2 \|\nabla f(x^{(k)})\|_2^2 \\ &\leq \|x^{(k)} - x^*\|_2^2 - 2\eta \frac{1}{L} \|\nabla f(x^{(k)})\|_2^2 + \eta^2 \|\nabla f(x^{(k)})\|_2^2 \\ &= \|x^{(k)} - x^*\|_2^2 - \eta \left(\frac{2}{L} - \eta \right) \|\nabla f(x^{(k)})\|_2^2, \end{aligned} \tag{5.6}$$

where we have used the second inequality of Lemma 5.2 and the fact that $\nabla f(x^*) = 0$. Note that $\eta \left(\frac{2}{L} - \eta \right) > 0$, because $\frac{2}{L} > \frac{1}{L} \geq \eta$. Therefore $\|x^{(k)} - x^*\|_2$ is a decreasing sequence in k, so $\|x^{(k)} - x^*\|_2 \leq \|x^{(0)} - x^*\|_2$. By applying Lemma 5.1, we have

$$\begin{aligned} f(x^{(k+1)}) &\leq f(x^{(k)}) + \langle \nabla f(x^{(k)}), x^{(k+1)} - x^{(k)} \rangle + \frac{L}{2} \|x^{(k+1)} - x^{(k)}\|_2^2 \\ &= f(x^{(k)}) + \langle \nabla f(x^{(k)}), x^{(k)} - \eta \nabla f(x^{(k)}) - x^{(k)} \rangle + \frac{L}{2} \|x^{(k)} - \eta \nabla f(x^{(k)}) - x^{(k)}\|_2^2 \\ &= f(x^{(k)}) - \langle \nabla f(x^{(k)}), \eta \nabla f(x^{(k)}) \rangle - \frac{L}{2} \|\eta \nabla f(x^{(k)})\|_2^2 \\ &= f(x^{(k)}) - \eta \|\nabla f(x^{(k)})\|_2^2 - \frac{L}{2} \eta \|\nabla f(x^{(k)})\|_2^2 \\ &= f(x^{(k)}) - \eta \left(1 - \frac{L\eta}{2} \right) \|\nabla f(x^{(k)})\|_2^2 \end{aligned} \tag{5.7}$$

Since $\eta \leq \frac{1}{L}$, it holds that

$$-\left(1-\frac{L\eta}{2}\right) = \frac{L\eta}{2} - 1 \leq \frac{L}{2}(1/L) - 1 = -\frac{1}{2}. \tag{5.8}$$

Subtracting $f(x^*)$ from both sides of (5.7) and using (5.8), we can conclude the following:

$$f(x^{(k+1)}) - f(x^*) \leq f(x^{(k)}) - f(x^*) - \frac{\eta}{2}\|\nabla f(x^{(k)})\|_2^2 \tag{5.9}$$

If we apply the first order condition for convexity, the Cauchy-Schwartz inequality and we take into account the fact that the sequence $\|x^{(k)} - x^*\|_2$ is decreasing, we get

$$\begin{aligned} f(x^{(k)}) - f(x^*) &\leq \langle \nabla f(x^{(k)}), x^{(k)} - x^* \rangle \\ &\leq \|\nabla f(x^{(k)})\|_2 \|x^{(k)} - x^*\|_2 \leq \|\nabla f(x^{(k)})\|_2 \|x^{(0)} - x^*\|_2. \end{aligned} \tag{5.10}$$

Solving (5.10) for $\|\nabla f(x^{(k)})\|_2$ and inserting the result in (5.9) yields

$$f(x^{(k+1)}) - f(x^*) \leq f(x^{(k)}) - f(x^*) - \frac{\eta}{2\|x^{(0)} - x^*\|_2^2}(f(x^{(k)}) - f(x^*))^2 \tag{5.11}$$

Let us denote $\beta = \frac{\eta}{2\|x^{(0)}-x^*\|_2^2}$ and $\delta_k = f(x^{(k)}) - f(x^*)$. It is clear that $\beta > 0$ and $\delta_k \geq 0$. Additionally, from (5.9) we can deduce that $\delta_{k+1} \leq \delta_k$. By manipulating (5.11), we obtain

$$\delta_{k+1} \leq \delta_k - \beta\delta_k^2 \Leftrightarrow \beta\frac{\delta_k}{\delta_{k+1}} \leq \frac{1}{\delta_{k+1}} - \frac{1}{\delta_k} \Leftrightarrow \beta \leq \frac{1}{\delta_{k+1}} - \frac{1}{\delta_k}, \tag{5.12}$$

where in the first equivalence we multiply both sides of the inequality by $\frac{1}{\delta_k\delta_{k+1}}$ and in the second one we use that $\frac{\delta_k}{\delta_{k+1}} \geq 1$. Since the inequality (5.12) is true for every iteration of gradient descent, summing up both sides over iterations and calculating the telescopic sum[3], gives

$$\sum_{i=0}^{k-1}\beta \leq \sum_{i=0}^{k-1}\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \Longrightarrow k\beta \leq \frac{1}{\delta_k} - \frac{1}{\delta_0} \leq \frac{1}{\delta_k}.$$

That is,

$$f(x^{(k)}) - f(x^*) \leq \frac{2\|x^{(0)} - x^*\|_2^2}{\eta k}.$$

□

In words, the theorem states that the gradient descent algorithm is guaranteed to converge for convex and L-smooth functions with rate of convergence $O(1/k)$, when a suitable learning rate is selected.

Observation 5.1. If we had simply assumed that $\frac{2}{L} > \eta > 0$, we could repeat the previous

[3]See Telescoping Series.

proof by considering

$$\beta = \frac{\eta\left(1 - \frac{L\eta}{2}\right)}{\|x^{(0)} - x^*\|_2^2}$$

to obtain

$$f(x^{(k)}) - f(x^*) \leq \frac{2\|x^{(0)} - x^*\|_2^2}{\eta(2 - L\eta)k}.$$

5.3.1.1. Convergence for smooth but non-convex functions

When f is not a convex function, but it is L-smooth, it can be proven that the minimizing sequence $x^{(k)}, k = 0, 1, 2, \ldots$, still converges to a stationary point by using the previous proof. First, note that, since $L \leq \frac{1}{\eta}$, the following holds:

$$-\eta\left(\frac{2}{L} - \eta\right) = \eta^2 - \frac{2\eta}{L} \leq \eta^2 - 2\eta^2 = -\eta^2.$$

Using the above inequality in (5.6) and re-arranging the terms, we have

$$\|\nabla f(x^{(k)})\|_2^2 \leq \frac{1}{\eta^2}\|x^{(k)} - x^*\| - \frac{1}{\eta^2}\|x^{(k+1)} - x^*\|.$$

If we now sum up both sides over $k = 0, 1, \ldots, K-1$, and divide by K, we get

$$\begin{aligned}\frac{1}{K}\sum_{k=0}^{K-1}\|\nabla f(x^{(k)})\|_2^2 &\leq \frac{1}{K\eta^2}\sum_{k=0}^{K-1}(\|x^{(k)} - x^*\| - \|x^{(k+1)} - x^*\|) \\ &= \frac{1}{K\eta^2}(\|x^{(0)} - x^*\| - \|x^{(K)} - x^*\|) \\ &\leq \frac{1}{K\eta^2}\|x^{(0)} - x^*\|\end{aligned}$$

Therefore,

$$\min_{k=0,1,\ldots,K-1}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{1}{K}\sum_{k=0}^{K-1}\|\nabla f(x^{(k)})\|_2^2 \leq \frac{1}{K\eta^2}\|x^{(0)} - x^*\|$$

In this way, if we record the point $x^{(k)}$ with the minimum value of the norm of the gradient, this norm is guaranteed to converge to zero with a sublinear rate, i.e., the sequence of iterates converge to a stationary point.

5.3.2. Application to learning

In the learning framework we seek to minimize the empirical risk $L_{\mathcal{D}}$ over the training data set $\mathcal{D}$. Sometimes, a regularization term is added to the objective function, so our problem consists in minimizing the following

$$J(\alpha) = L_{\mathcal{D}} + \lambda\Omega(\alpha) = \frac{1}{m}\sum_{i=1}^{m}\ell(\alpha, z_i) + \lambda\Omega(\alpha), \quad \alpha \in \Lambda. \tag{5.13}$$

It is common that $\Lambda \subset \mathbb{R}^N$ and that $J : \Lambda \to \mathbb{R}$ is differentiable. Therefore, we can apply the gradient descent method to solve the minimization problem. In order to do so, we must be able to compute the gradient of J with respect to α. Provided that $\Omega(\alpha)$ and the function $\alpha \mapsto \ell(\alpha, z)$, for each $z \in \mathcal{Z}$, are differentiable, it holds

$$\nabla J(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\alpha \ell(\alpha, z_i) + \lambda \nabla \Omega(\alpha), \quad \alpha \in \Lambda. \tag{5.14}$$

In most learning scenarios it is often the case that the objective functions are not convex or L-smooth. Consequently, the gradient descent algorithm is not guaranteed to converge. Nevertheless, it has been proven empirically that gradient descent (as well as its variants that will be described in the next section) returns really good local optima in a reasonable amount of time. In fact, the most effective modern optimization algorithms in machine learning are based on gradient descent.

Gradient descent algorithm is sensitive to different parameter values, which must be adjusted for each particular problem. On the one hand, the starting point can considerably affect the quality of the solution obtained, so it plays a crucial role. On the other hand, the leaning rate must be a small value, but not too small, so that the algorithm is able to converge to a good local optimum (see Figure 5.3 and Figure 5.4).

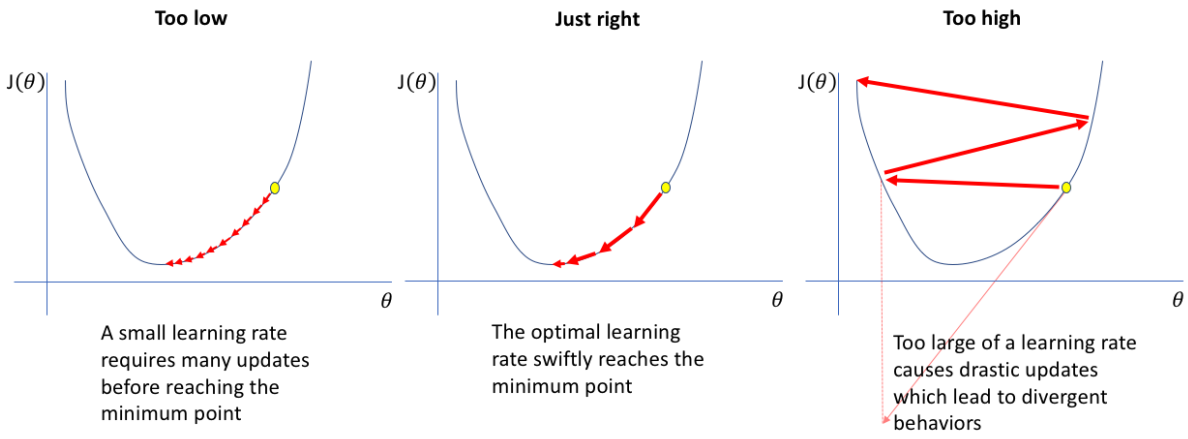


Figure 5.3.: Influence of the learning rate in the convergence of gradient descent. Image obtained from https://www.jeremyjordan.me/nn-learning-rate/.

We can see in Figure 5.4 that a very high learning rate may cause the algorithm to diverge, while a high learning rate arrives fast at a position close to a minimum but then it oscillates around it. In constrast, a low learning rate needs lot of iterations to converge, but the algorithm finds a better optimum. Ideally, we should have a high learning rate at the beginning, when far from the minimum to get quickly to a region near a good optimum, and then a low learning rate when close to the minimum to take more careful steps. The learning rate can be chosen by trial and error, but it is better to choose it by monitoring the learning curves that plot the evolution of the values of the objective function as the learning progress, like the ones depicted in Figure 5.4.

Additionally, the stopping criterion of the algorithm must be selected carefully, since an incorrect stopping rule may induce the algorithm to finish before finding a good enough op-

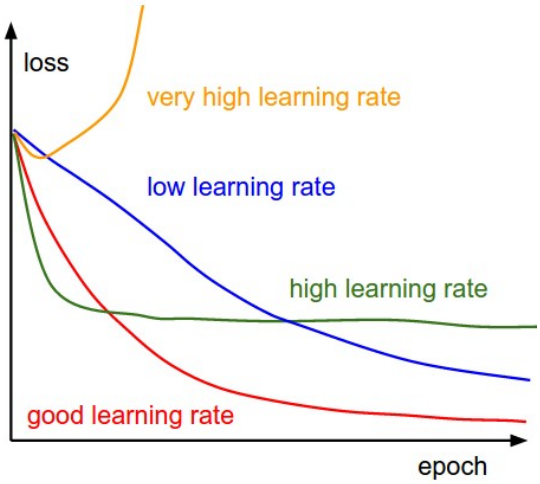


Figure 5.4.: Evolution of the value of the objective funcion (loss) with the number of iterations (epochs) for different values of the learning rate. Source https://cs231n.github.io/neural-networks-3/.

timum. Some frequently used criteria to decide when to stop the gradient descent algorithm are the following:

1. The value of the objective function drops below a certain threshold.
2. The norm of the gradient is close to zero.
3. A certain number of iterations (epochs) have been completed.
4. The value of the objective function does not change significantly after a certain number of iterations (*early stopping*).
5. A combination of the above.

5.3.3. Modifications of gradient descent

Here we will briefly explain several optimization algorithms in the learning framework that are modifications of the gradient descent algorithm. Some of them compute adaptive learning rates based on the value of the gradient in each iteration. It has been proven that all of the algorithms presented here converge for the case of convex functions.

5.3.3.1. Stochastic Gradient Descent (SGD)

The gradient descent method we have just described computes the gradient (5.14) using the entire training dataset at each iteration. This optimization algorithm, that processes all the training examples at the same time in a large batch, is known as *batch gradient descent*. It is computationally expensive, especially when the number of examples in the training dataset is large, since it requires evaluating the gradients on every example at each iteration, what normally is not a cheap calculation. Moreover, this algorithm is prone to getting stuck in close local optima.

As an alternative approach the *minibatch stochastic gradient descent*, or simply *stochastic gradient descent* method, arises. With this method, a small subset of the training data set $\{z1, \ldots, z_M\} \subset \mathcal{D}$ (with M significantly smaller than m) is selected at random at each iteration, which is then utilized to compute an estimate of $\nabla L_{\mathcal{D}}$ by taking the average gradient over the examples in the subset. The random subset is known as *minibatch*, and the number of examples M is a parameter known as *mininbatch size* (or *batch size*). This way, we can obtain an unbiased estimate of the gradient using only the examples in the minibatch. Originally, the name stochastic gradient descent was reserved for the case $M = 1$, that is, only one example come into play at each iteration. However, nowadays it is used in a more general way to refer to any batch size. The steps of the algorithm are shown in Algorithm 2.

Algorithm 2: Stochastic Gradient Descent (SGD)

Input: Loss function ℓ, training dataset $\mathcal{D}$, step size η, minibatch size M, regularization parameter λ
Initialize $\alpha^{(0)}$
$k \leftarrow 0$
repeat
  Sample a minibatch $\{z_1, \ldots, z_M\} \subset \mathcal{D}$ randomly
  Compute gradient estimate $g(k) = \frac{1}{M} \sum_{i=1}^{M} \nabla_{\alpha} \ell(\alpha^{(k)}, z_i) + \lambda \nabla \Omega(\alpha^{(k)})$
  $\alpha^{(k+1)} \leftarrow \alpha^{(k)} - \eta g(k)$
  $k \leftarrow k + 1$
until *stopping criterion is met*
return $\alpha^{(k)}$

The SGD may add some noise to the learning process due to the random sampling of the minibatch, what makes small minibatches provide a regularization effect. In fact, the best generalization error is obtained for a minibatch size of 1. However, training with small minibatches may require a low learning rate in order to maintain stability in the learning process, becuase in this case the gradient estimates could have high variante. As a consequence, the computation time needed can be very high, both because the number of necessary iterations to go thought the entire dataset is higher and because of the small learning rate. From its part, large batches achieve a more accurate estimate of the gradient. Nevertheless, if the examples are processed in parallel, the amount of memory required increases with the batch size.

The great advantage of SGD is that the computation time of an iteration does not grow with the number of training examples, what allows convergence even when the size of the training data set is large. Additionally, the randomness introduced by the algorithm can help avoiding flat regions and local minima.

In practice, SGD turns out to be very effective usually beating the performance of the original batch descent method.

5.3.3.2. Gradient descent with momentum

This technique is characterized by accumulating in each iteration information of the past gradients. In particular, at each iteration k, an exponentially decaying moving average of the negative past gradients is saved in a vector $v^{(k)}$, known as the *velocity*, which indicates the

direction and speed of the updates. In this way, the update rule is the following:

$$v^{(k+1)} \leftarrow \mu v^{(k)} - \eta \nabla J(\alpha^{(k)})$$
$$\alpha^{(k+1)} \leftarrow \alpha^{(k)} + v^{(k+1)},$$

where $\eta > 0$ is the learning rate and $\mu \in [0, 1)$ is a parameter called *momentum*, which determines how fast the contribution of the previous gradients decreases. The lower its value is relative to η, the faster the past gradients will be forgotten. When $\mu = 0$ we have the original gradient descent algorithm. Its value should ideally be adapted over time, so that at the beginning it is small and it increases later. The velocity is initialized to 0.

With this method, the step size depends not only on how large the gradients are (as it was the case with gradient descent) but also on how aligned the past gradients are. That is, the size of the step is biggest when many successive past gradients point in the same direction.

The main contribution of gradient descent with momentum is that it accelerates the convergence compared to the original algorithm, especially when the objective function has a high curvature, small but consistent gradients or there are only noisy estimates of the gradients available, as it is the case with stochastic gradient descent.

A further modification of the gradient descent method with momentum, known as *Nesterov momentum*, consists in evaluating the gradient after applying the current velocity to the parameters, as follows

$$v^{(k+1)} \leftarrow \mu v^{(k)} - \eta \nabla J(\alpha^{(k)} + \mu v^{(k)})$$
$$\alpha^{(k+1)} \leftarrow \alpha^{(k)} + v^{(k+1)}.$$

In the case of batch gradient descent, the rate of convergence with Nesterov momentum for convex functions increases from $O(1/k)$ to $O(1/k^2)$.

Until now we have considered a fixed learning rate. However, in practice it is necessary to gradually decrease the learning rate during the learning process, as we have previously mentioned, particularly when using SGD. That is exactly what the algorithms that we explain below try to accomplish.

5.3.3.3. RMSProp and AdaGrad

The *adaptive gradients (Adagrad)* algorithm adapts the learning rate individually for each parameter by making the adjustment inversely proportional to the square root of the sum of the squared values of the past gradients over the iterations, instead of using the same learning rate for all parameters. Then, the update rule is

$$r^{(k+1)} \leftarrow r^{(k)} + g^2(k) = r^{(k)} + g(k) \odot g(k),$$
$$\alpha^{(k+1)} \leftarrow \alpha^{(k)} - \frac{\eta}{\sqrt{r^{(k)}} + \epsilon} \odot g(k).$$

where $g(k)$ is the gradient vector estimate at iteration k calculated as in Algorithm 2, $\eta > 0$ is the learning rate and $\epsilon > 0$ is a small constant to avoid zero division. The square and square root operations are calculated element-wise and $\odot$ is used to refer to the element-wise vector product. We consider $r^{(0)} = 0$ (zero vector).

In this way, the parameters with the highest partial derivatives of the objective function present a fast decrease of their corresponding learning rates. In contrast, parameters with small partial derivatives maintain their learning rates almost intact. In other words, parameters whose learning rates are modified frequently will present low learning rates and, therefore, will update their values slowly, while parameters with rarely changing learning rates will update faster due to high values of the learning rates.

In the nonconvex setting, Adagrad may have the disadvantage of causing a premature excessive reduction of the learning rates, and, thus, reducing the effect of the updates. The *RMSprop (Root Mean Square Propagation)* algorithm solves this problem by using an exponential moving average of the squared gradients instead of just accumulating them. That is, in this case

$$r^{(k+1)} \leftarrow \gamma r^{(k)} + (1-\gamma) g^2(k)$$
$$\alpha^{(k+1)} \leftarrow \alpha^{(k)} - \frac{\eta}{\sqrt{r^{(k)}} + \epsilon} \odot g(k),$$

where the parameter $\gamma \in [0, 1)$ controls the influence of the past squared gradients.

RMSProp can be used in conjunction with momentum. In fact, the use of RMSProp combined with Nesterov momentum has been shown to be very effective in practice.

5.3.3.4. Adam

The *Adam (Adaptive Moment Estimation)* optimization algorithm combines the advantages of gradient descent with momentum and RMSProp. It keeps exponential moving averages of the past gradients ($m^{(k)}$), as in gradient descent with momentum, and the squared past gradients ($v^{(k)}$), as in RMSProp, that is,

$$m^{(k+1)} \leftarrow \beta_1 m^{(k)} + (1-\beta_1) g(k)$$
$$v^{(k+1)} \leftarrow \beta_2 v^{(k)} + (1-\beta_2) g(k) \odot g(k),$$

where the parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of the moving averages and present values close to 1, usually $\beta_1 = 0.9$ and $\beta_2 = 0.999$. $g(k)$ is the gradient vector estimate at iteration k and $\odot$ denotes the element-wise vector product. We can see these moving averages, $m^{(k)}$ and $v^{(k)}$, as estimates of the first moment (the expectation) and second moment (the uncentered variance) of the gradient, respectively.

The moving averages are initialized to zero vectors, what makes the moment estimates being biased toward zero, especially during the first iterations and when the parameters β_1, β_2 have large values (i.e. the rates of exponential decay are small). Indeed, it can be shown (see [19]) that

$$\mathbb{E}[m^{(k)}] \approx (1-\beta_1^k)\mathbb{E}[g(k)] \quad \text{and} \quad \mathbb{E}[v^{(k)}] \approx (1-\beta_2^k)\mathbb{E}[g^2(k)].$$

To solve this problem, Adam introduces a bias correction to the estimates, an aspect that the RMSProp algorithm lacks, as follows:

$$\hat{m}^{(k)} = \frac{m^{(k)}}{1-\beta_1^k}$$
$$\hat{v}^{(k)} = \frac{v^{(k)}}{1-\beta_2^k}$$

Therefore, the resulting update rule is the following:

$$\alpha^{(k+1)} \leftarrow \alpha^{(k)} - \frac{\eta}{\sqrt{\hat{v}^{(k)}}+\epsilon} \odot \hat{m}^{(k)},$$

being $\eta > 0$ the initial learning rate and $\epsilon > 0$ a small value that is used to avoid division by zero. The square root is calculated element-wise. In this way, $\hat{m}^{(k)}$ plays the role of the gradient, what makes it possible to incorporate momentum to gradient descent directly, and $\hat{v}^{(k)}$ adjust dynamically the learning rate of each parameter. All the steps of the algorithm are described in Algorithm 3.

Algorithm 3: ADAM

Input: Loss function ℓ, training dataset $\mathcal{D}$, initial learning rate η, minibatch size M, exponential decay rates β_1, β_2, small constant ϵ, regularization parameter λ

Initialize $\alpha^{(0)}$
$m^{(0)} \leftarrow 0$
$v^{(0)} \leftarrow 0$
$k \leftarrow 0$
repeat
  Sample a minibatch $\{z_1, \ldots, z_M\} \subset \mathcal{D}$ randomly
  $k \leftarrow k+1$
  Compute gradient estimate $g(k) = \frac{1}{M}\sum_{i=1}^{M} \nabla_\alpha \ell(\alpha^{(k-1)}, z_i) + \lambda \nabla \Omega(\alpha^{(k-1)})$
  $m^{(k)} \leftarrow \beta_1 m^{(k-1)} + (1-\beta_1) g(k)$
  $v^{(k)} \leftarrow \beta_2 v^{(k-1)} + (1-\beta_2) g(k)$
  $\hat{m}^{(k)} \leftarrow m^{(k)} / (1-\beta_1^k)$
  $\hat{v}^{(k)} \leftarrow v^{(k)} / (1-\beta_2^k)$
  $\alpha^{(k)} \leftarrow \alpha^{(k-1)} - \eta \hat{m}^{(k)} / (\sqrt{\hat{v}^{(k)}} + \epsilon)$
until *stopping criterion is met*
return $\alpha^{(k)}$

The Adam algorithm presents some important properties. Here we will explain two of them. The rest can be found in the original paper [19].

On the one hand, the step sizes are approximately bounded by the learning rate parameter. Let us assume that $\epsilon = 0$ and denote

$$\triangle_k = \frac{\eta}{\sqrt{\hat{v}^{(k)}}} \odot \hat{m}^{(k)},$$

the effective step taken at iteration k. Then, the effective step size has the following upper

bound for any iteration k:

$$\|\triangle_k\|_\infty \leq \max\left\{1, \frac{1-\beta_1}{\sqrt{1-\beta_2}}\right\}\eta.$$

On the other hand, the effective step sizes are invariant to rescaling of the gradient. In particular, if we rescale the gradients $g(k)$ by a factor $c > 0$, then $\hat{m}^{(k)}$ will become $c\hat{m}^{(k)}$ and $\hat{v}^{(k)}$ will be $c^2\hat{m}^{(k)}$. Therefore, the rescaling factors will cancel out:

$$\frac{1}{\sqrt{c^2\hat{v}^{(k)}}} \odot c\hat{m}^{(k)} = \frac{1}{\sqrt{\hat{v}^{(k)}}} \odot \hat{m}^{(k)}.$$

Additionally, for convex functions, it can be proven that the rate of convergence of Adam is $O(1/\sqrt{k})$.

There exist several variants of Adam. One of them, known as *NAdam*, consists in using Nesterov momentum instead of classical momentum.

As a conclusion, we can remark that optimization in learning involves deciding which optimization algorithm to use, adjusting the algorithm parameters, as well as selecting a good enough initial point or a method for generating a good enough initial point. Therefore, roughly speaking, we can say that it is more an art than a science.

Part III.

Deep Learning

6. Artificial Neural Networks

Artificial neural networks are a very powerful and flexible model, which can efficiently approximate complex target functions. Learning with artificial neural networks was proposed in the mid-20th century and, since then, it has become an effective learning paradigm which has shown outstanding performance on different learning tasks. The branch of machine learning devoted to the study of neural networks is known as **deep learning** and it is a very active area of research.

We begin the chapter by formally defining the model and explaining how to train artificial neural networks on data. We will show that they have a great approximation power, what makes them prone to overfitting and, consequently, regularization techniques are required. Finally, we will see that they have polynomial VC dimension.

The main references used for this part are [5], [6] and [8], although other sources have been also consulted as it will be indicated throughout the chapter.

6.1. Definition and notation

Artificial neural networks are often described as directed graphs whose nodes are called *neurons*, where each edge of the graph links the output of one neuron with the input of another node. We will focus on *feedforward neural networks*, whose corresponding graph does not contain cycles. Therefore, they are described by a directed acyclic graph which implements a composition of functions.

Definition 6.1. (Neuron) Let $w \in \mathbb{R}^d, d \in \mathbb{N}$, and $\theta : \mathbb{R} \to \mathbb{R}$ be a scalar function known as *activation function*. A neuron is then defined as a function $f : \mathbb{R}^{2d} \to \mathbb{R}$ as follows

$$f(x, w) = \theta\left(\sum_{i=1}^{d} w_i x_i\right), \quad \forall x, w \in \mathbb{R}^d.$$

In this way, a neuron receives as input the outputs of all the neurons connected to it in the graph, denoted by x, and a vector of parameters known as *weights*, denoted by w. Then, a weighted sum of the outputs is calculated, with weighting according to w, and an activation function θ is applied to the result, producing a real value that is the output of the neuron. By connecting neurons to each other we obtain a composition of the functions that they implement.

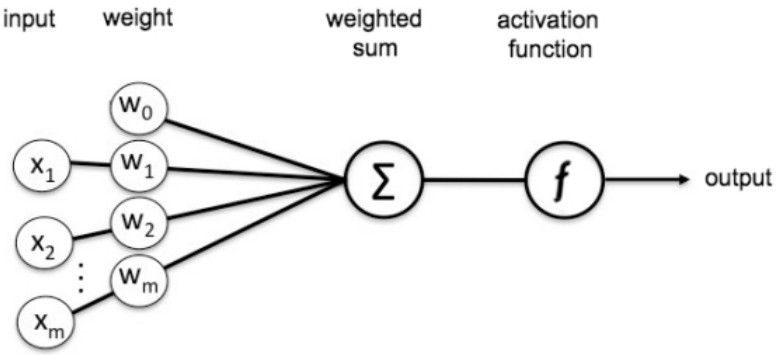


Figure 6.1.: Representation of a neuron. Image from [20].

A Neural Network is a collection of neurons connected to each other and organized into *layers*. The layers are labeled by $t = 0, 1, \ldots, T$, where T is the number of layers in the network or the *depth*. Every directed edge (arrow) of the graph has a weight associated to it and it connects a node in the layer t to a node in the next layer $t + 1$. A special node, called the *bias*, is added to each layer, which always outputs 1 and is not connected to any node in the previous layer.

The most common type of layer is called *fully-connected layer* and it is characterized because all neurons between two adjacent layers are pairwise connected, that is, each output of a layer affects all neurons in the next layer. This is the type of layer that we will consider throughout this chapter.

The layer $t = 0$ corresponds to the *input layer*, which has $m + 1$ nodes, being m the dimension of the input space, $\mathcal{X} \subset \mathbb{R}^m$. For $i = 1, \ldots, m$ the node i in the input layer outputs the feature x_i of the input vector, and the node 0 corresponds to the bias. It is not counted as a layer itself, because its nodes are not of the form considered in Def. 6.1. The last layer $t = T$ is the *output layer* and it determines the output of the function that the neural network implements. The bias node is not present in the output layer. For $0 < t < T$ we say that the layers are *hidden layers*.

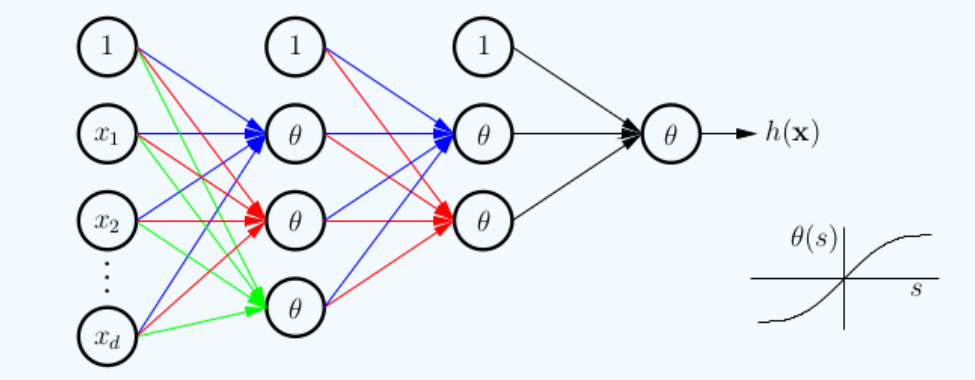


Figure 6.2.: Example of a 3-layer neural network with $m = d$ inputs, 2 hidden layers and 1 output. The activation function of every neuron is θ, whose graph is depicted in the bottom right corner. Image from [6] (Chapter 7).

The *dimension* of a layer t, that is, the number of neurons it has, is denoted by d_t, so that in total it contains $d_t + 1$ nodes (with the bias node). We call $\max_{t=1,\dots,T} d_t$ the *width* of the *network* and the *size* of the network is the total number of nodes it has.

For each layer $t = 0, 1, \dots, T$ we gather the outputs from its nodes $0, 1, \dots, d_t$ in a vector $x_t \in \{1\} \times \mathbb{R}^{d_t}$. Similarly, the weights associated with the incoming edges into layer $t = 1, \dots, T$ are collected in a matrix W_t, so that the weight w_{ij} is related to the edge going from the node i of layer $t-1$ into the neuron j in layer t. Then, the size of matrix W_t is $(d_{t-1} + 1) \times d_t$.

Since the activation function may differ among layers (and even among neurons in the same layer, although it is not common), we will denote by θ_t the activation function used by the neurons of layer $t = 1, \dots, T$.

Let us denote by $s_t = W_t^T x_{t-1}$, for $t = 1, \dots, T$ and $x_{t-1} \in \mathbb{R}^{d_{t-1}+1}$, where each component of the resulting vector is $s_{t,j} = \sum_{i=0}^{d_{t-1}} w_{ij} x_i$, $j = 1, \dots, d_t$. Therefore, it is clear that the output of each layer $t = 1, \dots, T$ is

$$x_t = \begin{bmatrix} 1 \\ \theta_t(s_t) \end{bmatrix} \quad \text{and} \quad x_T = \theta_T(s_T),$$

where $\theta_t(s_t)$ is a vector whose components are $\theta_t(s_{t,j})$.

We can sum up the above discussion in the following definition:

Definition 6.2. (Feedforward neural network) Let $T \in \mathbb{N}$ and $d = [d_0, d_1, \dots, d_T] \in \mathbb{N}^{T+1}$. Let $w = [W_1, \dots, W_T]$ with $W_t \in \mathbb{R}^{(d_{t-1}+1) \times d_t}$ a matrix of weights and $\theta = \{\theta_t : \mathbb{R} \to \mathbb{R} : t = 1, \dots, T\}$ a collection of scalar functions. Given an input features vector $x \in \mathbb{R}^{d_0}$, let us consider the following sequence:

$$x_0 = \begin{bmatrix} 1 \\ x \end{bmatrix}, \quad x_t = \begin{bmatrix} 1 \\ \theta_t(W_t^T x_{t-1}) \end{bmatrix}, \ t = 1, \dots, T-1, \quad x_T = \theta_T(W_T^T x_{T-1}).$$

Then, a *feedforward neural network* is determined by (T, d, θ, w) and it implements the function $h_{T,d,\theta}(\cdot, w) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_T}$, $h_{T,d,\theta}(x, w) = x_T$.

A hypothesis class of neural networks $\mathcal{H}_{T,d,\theta}$ is specified by fixing the number of layers T and the dimension of each layer, that is, the vector d, as well as the collection of activation functions θ. This triplet (T, d, θ) is known as the *architecture* of the neural network. Once the architecture is set, each hypothesis in $\mathcal{H}_{T,d,\theta}$ is determined by the weights over the edges of the network, w. In short,

$$\mathcal{H}_{T,d,\theta} = \{h_{T,d,\theta}(x, w) : w = [W_1, \dots, W_T], \text{ with } W_t \in \mathbb{R}^{(d_{t-1}+1) \times d_t}, t = 1, \dots, T\}$$

Observation 6.1. Note that, in Def. 6.2, the function that a neural network implements is, in fact, a composition of functions as the following chain shows:

$$x_0 \xrightarrow{W_1} s_1 \xrightarrow{\theta_1} x_1 \xrightarrow{W_2} s_2 \xrightarrow{\theta_2} x_2 \ \cdots \ x_{T-1} \xrightarrow{W_T} s_T \xrightarrow{\theta_T} x_T = h_{T,d,\theta}(x, w).$$

Example 6.1. (Multi-Layer Perceptron) A special kind of neuron, called *perceptron*, is obtained

when the activation function used is the sign function, which is given by:

$$sign(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}, \forall y \in \mathbb{R}.$$

A neural network formed by only one layer (which is the output layer) with a perceptron neuron can be used as the hypothesis class for binary classification problems, where each hypothesis in the class is specified by a vector of weights w. Let m be the dimension of the input vector, that is, $x \in \mathcal{X} \subset R^m$. Then, the perceptron represents a decision hyperplane $H = \{x \in \mathbb{R}^m : \sum_{i=0}^m w_i x_i = 0\}$ in the m-dimensional input space, so that it outputs 1 when the input instance lies on one side of the hyperplane and -1 when it belongs to the other side. In this way, the function implemented by the perceptron is the following:

$$h(x, w) = \begin{cases} 1 & \text{if } \sum_{i=0}^m w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}, \forall x \in \mathbb{R}^m, w \in \mathbb{R}^{m+1}.$$

For linearly separable training sets, it can be proven that, by using the hypothesis class represented by the perceptron, the hyperplane that separates the training data perfectly can always be learnt (see [6]).

By connecting several perceptron neurons in layers, we obtain the *Multi-Layer Perceptron*. This model is able to implement more complex functions than can classify correctly the data of a training set that is not linearly separable. In a later section we will show its expressive power.

6.2. Training a neural network

By *training* (or *learning*) a model we mean running an optimization algorithm that minimizes the objective function (5.13) over the parameters $\alpha \in \Lambda$. In the particular case of neural networks, once the architecture (T, d, θ) is selected, the gradient descent algorithm (or any of its variants) is executed to find the hypothesis in $\mathcal{H}_{T,d,\theta}$, that is, the set of weights w, for which the value of the objective function $J(w)$ is minimum.

In order to be able to apply the gradient descent algorithm, the gradient $\nabla J(w)$ must be calculated with respect to each weight matrix $W_t, t = 1, \ldots, T$, which is given by:

$$\nabla_{W_t} J(w) = \frac{1}{m} \sum_{i=1}^m \nabla_{W_t} \ell(h(w, x_i), y_i) + \lambda \nabla_{W_t} \Omega(w), \quad w = [W_1 \ldots, W_T],$$

where $w \mapsto \ell(h(w, x), y)$ determines the loss of predicting the value $h(w, x)$ when the true output is y, for each fixed $(x, y) \in \mathcal{D}$, and it is a differentiable function, as well as $\Omega(w)$ (at least almost everywhere). The gradient of the penalty term $\Omega(w)$ can be usually easily calculated analytically. However, there is not a simple closed expression for $\nabla_{W_t} \ell(h(w, x), y)$. We will explain the *backpropagation algorithm*, which shall allow us to calculate the partial derivatives of the loss $\ell(h(w, x), y)$ with respect to every weight efficiently.

Note that we can see $\nabla_{W_t} \ell(h(w, x), y)$ as a matrix of size $(d_{t-1} + 1) \times d_t$, where the element in the position (i, j) corresponds to the partial derivative of the loss with respect to the weight w_{ij} in the matrix W_t: $\frac{\partial \ell}{\partial w_{ij}}$.

For the application of the gradient descent algorithm to neural networks, it is a common practice to initialize the weights to small random values, in particular using a Normal distribution $w_{ij} \sim \mathcal{N}(0, \sigma^2)$, being σ^2 a small value normally chosen such that $\sigma^2 \cdot \max_{i=1,\dots,m} \|x_i\|_2^2 \ll 1$.

6.2.1. Forward Propagation

The *forward propagation algorithm* is responsible for computing the function $h_{T,d,\theta}(x, w)$ that the neural network implements. Its name comes from the fact that the information flows forward through the network in the calculation of the function, from the input layer, through the hidden layers and finally to the output.

The steps of the algorithm have already been explained in the previous section, when we defined the function implemented by a neural network. However, to make it more clear, we will show the steps here again:

Algorithm 4: FORWARD PROPAGATION

Input: Network architecture (T, d, θ), weigths w, input vector $x \in \mathbb{R}^{d_0}$

$x_0 \leftarrow \begin{bmatrix} 1 \\ x \end{bmatrix}$

for $t = 1, \dots, T-1$ **do**

  $s_t \leftarrow W_t^T x_{t-1}$

  $x_t \leftarrow \begin{bmatrix} 1 \\ \theta_t(s_t) \end{bmatrix}$

end

$s_T \leftarrow W_T^T x_{T-1}$

$x_T \leftarrow \theta_T(s_T)$

return x_T

Once the forward propagation algorithm has been executed, the outputs x_t of each layer in the network have been completely calculated, and, therefore, the value of the function $h_{T,d,\theta}(x, w)$ for the given input vector x is determined.

6.2.2. Backpropagation Algorithm

Backpropagation is an effective dynamic programming algorithm to compute the partial derivatives of any function with respect to each of its variables. In our context we will use it to obtain the partial derivatives of the loss function on any training example $(x, y) \in \mathcal{D}$ with respect to every weight of the network. The complexity of deriving all the partial derivatives is linear in the number of weights, i.e. $O(w)$, being w the total number of weights of the neural network.

This algorithm consists in applying several times the chain rule to obtain the partial derivatives in layer t by using partial derivatives in layer $t+1$. Therefore, it proceeds in a recursive manner, starting with the partial derivatives in the output layer and moving backwards throughout the hidden layers, hence the name of backpropagation.

6.2.2.1. The chain rule

In order to be able to understand the backpropagation algorithm, we must review some concepts about differentiability.

Definition 6.3. (Jacobian Matrix) Let $\Omega \subset \mathbb{R}^n$ be an open set and $F : \Omega \to \mathbb{R}^m$ a differentiable function at a point $a \in \Omega$. The differential of F at a is a linear transformation denoted by $D(F)_a$. The matrix associated to $D(F)_a$ with respect to the standard basis is known as the Jacobian matrix of F at point a, and it is denoted by $J(F)_a$. If $F = (F_1, \dots, F_m)$, then, the element (i, j) of the Jacobian matrix is given by:

$$\frac{\partial F_i}{\partial x_j}(a), \quad \forall i = 1, \dots, m, \forall j = 1, \dots, n.$$

Therefore, the Jacobian matrix is formed by the partial derivatives of the components of F with respect to each of its variables.

In the particular case of $m = 1$, we have that $J(F)_a = \nabla F(a)$ is the gradient as a row matrix.

Theorem 6.1. *(*Chain rule*) Let $\Omega \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$ be open sets and let $F : \Omega \to \mathbb{R}^m$ and $G : U \to \mathbb{R}^p$, $p \in \mathbb{N}$, be two differentiable functions at points $a \in \Omega$ and $b = F(a) \in U$, respectively, verifying that $f(\Omega) \subset U$. Then, we can define the composition $H = G \circ F : \Omega \to \mathbb{R}^p$, which is a differentiable function at point a with*

$$D(H)_a = D(G)_{F(a)} \circ D(F)_a,$$

or, in terms of the Jacobian matrices,

$$J(H)_a = J(G)_{F(a)} \cdot J(F)_a.$$

Example 6.2. Let us define $F : \mathbb{R}^n \to \mathbb{R}^m$ as $F(w) = Aw$, with $A \in \mathbb{R}^{m \times n}$ and $w \in \mathbb{R}^{n \times 1}$. Then F is differentiable at any point with $J(F)_w = A$.

Let us also consider $\theta_t : \mathbb{R}^m \to \mathbb{R}^m$, defined as in subsubsection 6.2.2.2, item 5. By the chain rule, it holds that

$$J(\theta_t \circ F)_w = \operatorname{diag}(\theta_t'(Aw))A.$$

6.2.2.2. Description of the algorithm

To describe the backpropagation algorithm, let us first provide some clarifications.

1. For the sake of simplicity, we will omit the bias node in the description of the algorithm. An extended description where the bias node is taking into account can be found in [6] (Chapter 7).

2. If the layers are not fully-connected, that is, there are some missing edges in the graph and, therefore, some missing weights, we will assume a 0 in the corresponding position for that weight in the matrix W_t where it would belong. We can make this assumption without loss of generality, since when calculating the partial derivative with respect to a particular weight we fix all other weights of the network.

3. When calculating the gradient of the loss with respect to a weight matrix W_t, the outputs of the layer $t - 1$ are fixed, they do not depend on the weights W_t, because all the other weights are fixed.

4. For each fixed output value $y \in \mathcal{Y}$, we shall see the loss function $\ell(\cdot, y) : \mathbb{R}^{d_T} \to \mathbb{R}$ as a function of the outputs of the network, which, in turn, depend on the weights of each layer. For each $t = 1, \dots, T$, let us consider the subnetwork formed by layers

$t, t+1, \dots, T$ and define the loss function $\ell_t : \mathbb{R}^{d_t} \to \mathbb{R}$ over the outputs of layer t, which we shall assume differentiable for each t. Those intermediate loss functions determine the loss ℓ produced by the considered subnetwork in the following way:

$$\ell_t(u) = \ell((f_T \circ \cdots \circ f_{t+1})(u), y), \quad \forall u \in \mathbb{R}^{d_t}, \tag{6.1}$$

being f_t the function implemented by layer t of the network. Note that for the last layer $\ell_T(u) = \ell(u, y), \quad \forall u \in \mathbb{R}^{d_T}$.

5. When applying the activation function of each layer θ_t to a vector of d_t size, we will see it as a function $\theta_t : \mathbb{R}^{d_t} \to \mathbb{R}^{d_t}$ which applies the scalar activation function element-wise. That is, if $s_t \in \mathbb{R}^{d_t}$, each component of $\theta_t(s_t)$ is $\theta_t(s_{t,j})$, being $s_{t,j}$ the component j of vector s_t. It is easy to see that $J(\theta_t)s_t$ is a diagonal matrix, whose $(j, j), j = 1, \dots, d_t$, diagonal element is $\theta_t'(s_{t,j})$, the derivative of the scalar activation function at the component j of vector s_t. We will denote this matrix by $\text{diag}(\theta_t'(s_t))$

6. For each $t = 1, \dots, T$ we will rewrite the matrix of weights W_t as a column vector $w_t \in \mathbb{R}^{d_{t-1}d_t}$ by concatenating the columns of the weight matrix. We will also define the matrix X_{t-1} of size $d_t \times d_{t-1}d_t$ as follows:

$$X_{t-1} = \begin{pmatrix} x_{t-1}^T & 0 & \cdots & 0 \\ 0 & x_{t-1}^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_{t-1}^T \end{pmatrix}.$$

   In this way, we can write $W_t^T x_{t-1} = X_{t-1} w_t$.

7. We will define the sensitivity vector of layer $t = 1, \dots, T$ as $\delta_t = J(\ell_t)_{x_t} = \nabla \ell_t(x_t)$ of size d_t, which quantifies how the loss changes with the outputs of that layer.

The following proposition shows the building block of the backpropagation algorithm:

Proposition 6.1. *Let (T, d, θ, w) be a feedforward neural network, where each activation function in θ is differentiable, and let $\ell(\cdot, y) : \mathbb{R}^{d_T} \to \mathbb{R}$ be a differentiable function, for each $y \in \mathcal{Y}$. Then, for each $x \in \mathcal{X} \subset \mathbb{R}^{d_0}$ it is true that:*

$$\nabla_{W_t} \ell(h(w, x), y) = x_{t-1}[\delta_t \odot \theta_t'(s_t)], \quad t = 1, \cdots, T, \tag{6.2}$$

where $\odot$ denotes element-wise multiplication and

$$\delta_t = [\delta_{t+1} \odot \theta_{t+1}'(s_{t+1})] W_{t+1}^T, \quad t = 1, \cdots, T. \tag{6.3}$$

Proof. ([5], Chapter 20) Let us define the auxiliary function $g_t : \mathbb{R}^{d_{t-1}d_t} \to \mathbb{R}$, for each $t = 1, \dots, T$, which expresses the loss as a function of the weights vector w_t:

$$g_t(w_t) = \ell_t(x_t) = \ell_t(\theta_t(s_t)) = \ell_t(\theta_t(X_{t-1} w_t)).$$

By applying the chain rule we obtain:

$$J(g_t)_{w_t} = \nabla_{w_t} g_t(w_t) = J(\ell_t)_{\theta_t(X_{t-1}w_t)} \text{diag}(\theta_t'(X_{t-1}w_t)) X_{t-1},$$

that is,

$$\nabla_{w_t} g_t(w_t) = J(\ell_t)_{x_t} \text{diag}(\theta_t'(s_t)) X_{t-1},$$

and using $\delta_t = J(\ell_t)_{x_t}$, we can write

$$\nabla_{w_t} g_t(w_t) = (\delta_{t,1}\theta_t'(s_{t,1})x_{t-1}^T, \ldots, \delta_{t,d_t}\theta_t'(s_{t,d_t})x_{t-1}^T) = [\delta_t \odot \theta_t'(s_t)]X_{t-1},$$

or in matrix form

$$\nabla_{W_t} g_t(W_t) = x_{t-1}[\delta_t \odot \theta_t'(s_t)]. \tag{6.4}$$

Taking the definition (6.1) of ℓ_t into account we have $\ell_t(\theta_t(W_t^T x_{t-1})) = \ell(h(w,x),y)$ and, since $\nabla_{W_t} g_t(W_t) = \nabla_{W_t} \ell_t(\theta_t(W_t^T x_{t-1}))$, (6.4) gives (6.2).

To proof (6.3), we observe, from the definition (6.1) itself, that the following equality holds:

$$\ell_t(u) = \ell_{t+1}(\theta_{t+1}(W_{t+1}^T u)), \quad \forall u \in \mathbb{R}^{d_t}.$$

By the chain rule:

$$J(\ell_t)_u = J(\ell_{t+1})_{\theta_{t+1}(W_{t+1}^T u)} \text{diag}(\theta_{t+1}'(W_{t+1}^T u)) W_{t+1}^T,$$

therefore,

$$\begin{aligned}
\delta_t = J(\ell_t)x_t &= J(\ell_{t+1})_{\theta_{t+1}(W_{t+1}^T x_t)} \text{diag}(\theta_{t+1}'(W_{t+1}^T x_t)) W_{t+1}^T \\
&= J(\ell_{t+1})_{x_{t+1}} \text{diag}(\theta_{t+1}'(s_{t+1})) W_{t+1}^T \\
&= \delta_{t+1} \text{diag}(\theta_{t+1}'(s_{t+1})) W_{t+1}^T \\
&= [\delta_{t+1} \odot \theta_{t+1}'(s_{t+1})] W_{t+1}^T.
\end{aligned}$$

□

In this way, during forward propagation the output of each layer x_t is calculated from the output x_{t-1} of the previous layer, while during backpropagation the sensitivity vectors δ_t are computed (backwards) for each layer t using the sensitivity vector of the next layers δ_{t+1}.

In order to start the backpropagation algorithm, we must be able to compute $\delta_T = \nabla_{x_T} \ell(x_T, y)$, which is normally an easy analytical calculation. For example, by choosing the squared loss $\ell(x_T, y) = \frac{1}{2}\|x_T - y\|_2^2$, we have $\delta_T = (x_T - y)$.

Note that the derivatives of the activation functions θ_t' must also be known. That is the reason why we impose that they are differentiable (or at least continuous and differentiable almost everywhere). The computation of these derivatives is normally straightforward, as we will see in the next section.

Once the output vectors x_t and the sensitivity vectors δ_t are computed for each layer $t = 1, \ldots, T$ by forward propagation and backpropagation, respectively, the gradients of the loss function with respect to the weight matrices can be easily obtained using (6.2).

We can now show the steps of the backpropagation algorithm:

Algorithm 5: BACKPROPAGATION

Input: Network architecture (T, d, θ) with each θ_t differentiable, weigths w, differentiable loss function ℓ, training example (x, y)

$(x_1, \dots, x_t), (s_1, \dots, s_T) \leftarrow$ FORWARD PROPAGATION$((T, d, \theta, w), x)$

$\delta_T \leftarrow \nabla_{x_T} \ell(x_T, y)$

$\nabla_{W_T} \ell(h_{T,d,\theta}(w, x), y) \leftarrow x_{T-1}[\delta_T \odot \theta'_T(s_T)]$

for $t = T-1, \dots, 1$ **do**

  $\delta_t \leftarrow [\delta_{t+1} \odot \theta'_{t+1}(s_{t+1})] W_{t+1}^T$

  $\nabla_{W_t} \ell(h_{T,d,\theta}(w, x), y) \leftarrow x_{t-1}[\delta_t \odot \theta'_t(s_t)]$

end

return $[\nabla_{W_1} \ell(h_{T,d,\theta}(w, x), y), \dots, \nabla_{W_T} \ell(h_{T,d,\theta}(w, x), y)]$

Therefore, by running backpropagation for each training example $(x_i, y_i) \in \mathcal{D}$, we obtain the partial derivatives of the loss function with respect to every weight of the network on each training data point, so that we can compute $\nabla_w J(w) = [\nabla_{W_1} J(w), \dots, \nabla_{W_T} J(w)]$, which is then used in the gradient descent algorithm to learn the best set of weights w.

6.3. Activation Functions

A key component of neural networks is the *activation function* that is used by the neurons of the layers. The design of activation functions is a highly active area of research and there is not yet a definite guide on when to use each of them. Here we will present some of the most commonly used activation functions together with their advantages and disadvantages (with the help of [21], [22] and [20]), so that we can have an idea of which type to try in different situations. However, in most cases the decision process consists in trial and error.

The activation functions used in neural networks are almost always of non linear type. Indeed, if we connected several layers of neurons with linear activation functions then the result function, which is a composition of the functions implemented by each layer, would be again a linear function, so the entire network would be equivalent to a single layer of linear neurons and it would not make sense to use a neural network with a lot of hidden layers.

6.3.1. Sigmoid functions

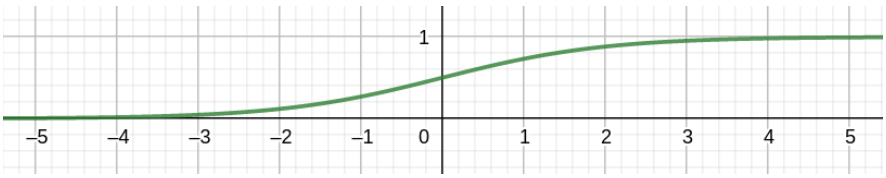
Sigmoid functions are characterized by its "S"-shape. They are bounded and continuously differentiable at every point, what makes them perfectly suitable for the backpropagation algorithm, with positive derivative (so they are monotonically increasing).

The **logistic sigmoid function** has the mathematical form:

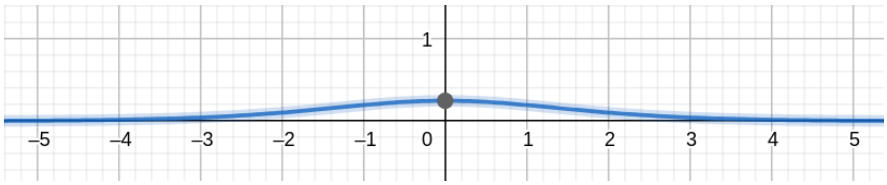
$$\sigma : \mathbb{R} \to [0, 1], \quad t \mapsto \frac{1}{1 + e^{-t}} = \frac{e^t}{1 + e^t}.$$

This function takes a real number and “squashes” it into the $[0, 1]$ range, so it can be used for binary classification problems in the output layer of a neural network to indicate the probability that the binary variable is 1. It is a symmetric function in the sense that $\sigma(-t) = 1 - \sigma(t)$, $\forall t \in \mathbb{R}$, and its derivative is given by

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t)), \quad t \in \mathbb{R}.$$



(a) Logistic sigmoid function $\sigma(t)$.



(b) Derivative of the logistic sigmoid function $\frac{d\sigma(t)}{dt}$.

Figure 6.3.: Graphs of the logistic sigmoid function and its derivative.

One of the problems of the logistic sigmoid function is that it is *not zero-centered*. This is an undesirable property, because in this case the weights need to be updated more times so that the network can be trained properly. Indeed, if the activation function of a neuron is the logistic sigmoid, then its output is always positive and, if the data coming into a neuron is always positive (i.e. $x > 0$ elementwise), then the gradient of the objective function with respect to the weights will become all positive or all negative during backpropagation, concretely, the same sign as δ_t (see Proposition 6.1). Therefore, only zig-zag updates are produced leading to a necessity of more epochs to reach the optimum (see Figure 6.4). The zero-centered property is desirable, but not necessary, since the problem can be mitigated when the gradients are added up across the data in a minibatch.

To solve this issue, the **hyperbolic tangent function** is introduced. It presents the same benefits of the logistic sigmoid function together with the zero-centered property. It is defined as follows:

$$\tanh : \mathbb{R} \to [-1, 1], \quad t \mapsto \frac{1 - e^{-t}}{1 + e^{-t}}.$$

It can be seen as a modification of the logistic sigmoid, since $\tanh(t) = 2\sigma(2t) - 1, \forall t \in \mathbb{R}$, hence it has the same properties as the logistic sigmoid function. It is an odd function, that is, $\tanh(-t) = -\tanh(t), \forall t \in \mathbb{R}$, whose derivative can be easly calculated as

$$\frac{d\tanh(t)}{dt} = (1 - \tanh^2(t)), \quad t \in \mathbb{R}.$$

The hyperbolic tangent shows in practice better training performance than the logistic sigmoid, so it has become more popular. Nevertheless, both sigmoid functions have high computational cost as a result of their exponential nature and they present a major drawback: the *vanishing gradient problem*. Let us explain this problem in more detail. When the value of the real variable t is very positive, the sigmoid neuron saturates to the value 1, and, when t is very negative, then it saturates to 0 (or -1 in the case of the hyperbolic tangent), leading

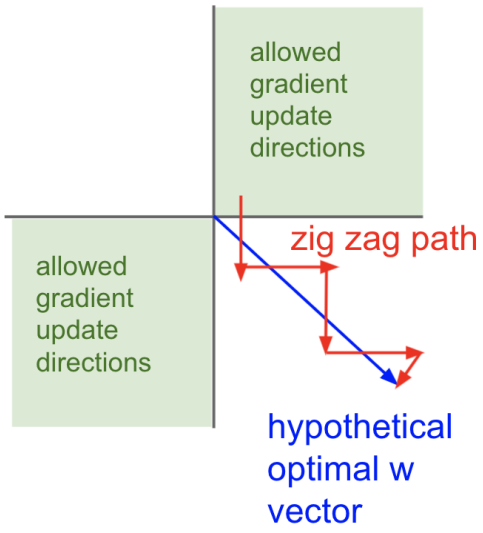


Figure 6.4.: Hypothetical representation of weight updates for not zero centered activation functions. Image from Standford University. Lecture 6 | Training Neural Networks I.

to an almost zero gradient of the activation function. During backpropagation, this local gradient is used to calculate the final gradient of the objective function. If a value close to zero is multiplied several times to other almost zero values, then the value of the gradient gets closer to zero as it propagates backwards thought the network. Therefore, the weights updates will be almost insignificant and the value of the objective function stops decreasing, causing the network to not being trained properly. As a consequence, we should be careful with the initialization of the weights, since, if they are too large, most neurons would get saturated. For this reason, nowadays their use in hidden layers of feedforward networks is discouraged.

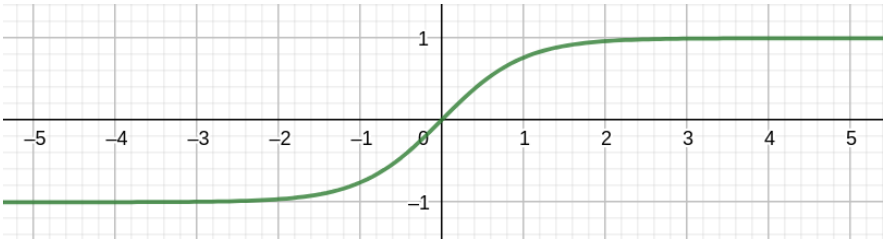
Sigmoid functions, however, can be used in the output layer when an appropriate loss function is chosen to mitigate the saturation caused by the sigmoid.

Finally, we will introduce the **softmax function**, which is typically used in the output layer for multi-class classification problems where the classes are disjoint, to represent the probability that the input vector belongs to each of the different classes of the problem. It is given by:

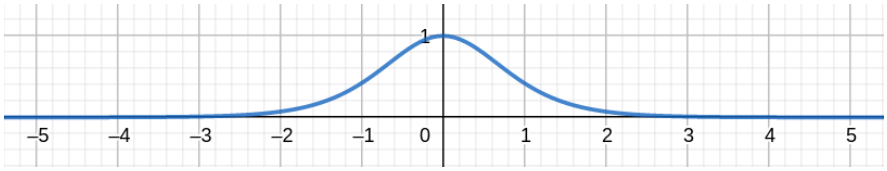
$$\text{softmax} : \mathbb{R}^N \to [0,1]^N, \quad \text{softmax}(s)_j = \frac{e^{s_j}}{\sum_{i=1}^{N} e^{s_i}},$$

where N is the number of disjoint classes, which must be equal to d_T, and $s_j, j = 1, \ldots, d_T$, are the components of the vector $s_T = W_T^T x_{T-1}$. The denominator in the softmax function guarantees that the outputs add up to 1, so the output of the network given by the softmax can be interpreted as a probability distribution defined over a discrete random variable with N possible values.

Unlike with local activation functions, the neurons of the output layer are not independent to each other when the softmax function is used, but a competitive factor is introduced among



(a) Hyperbolic tangent function $\tanh(t)$.



(b) Derivative of the hyperbolic tangent function $\frac{d\tanh(t)}{dt}$.

Figure 6.5.: Graphs of the hyperbolic tangent function and its derivative.

the output neurons.

Denoting by $y_j = \text{softmax}(s)_j$, the partial derivatives of the softmax function are the following

$$\frac{\partial y_j}{\partial s_i} = y_j(\delta_{ij} - y_i),$$

being

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

the Kronecker delta function.

The softmax function can be seen as a generalization of the logistic sigmoid function to multiple classes, and, as such, it presents the same problems as the sigmoid.

6.3.2. Rectified Linear Units (ReLU)

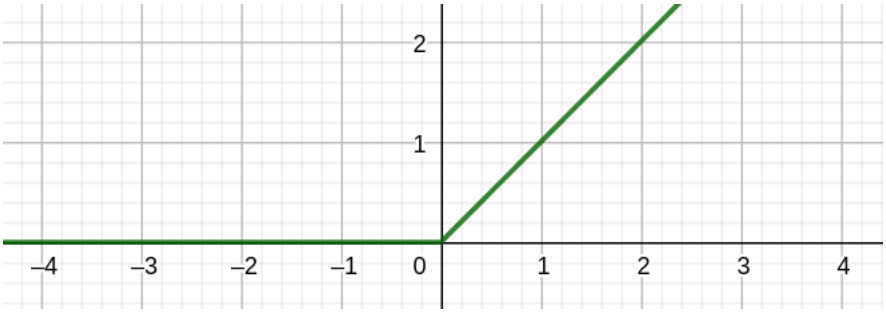
The **Rectified Linear Units (ReLU)** can be used to avoid some of the problems that sigmoid units suffer from, such as the vanishing gradient. They use the following activation function:

$$\text{ReLU} : \mathbb{R} \to \mathbb{R}_0^+, \quad t \mapsto \max\{0, t\} = \begin{cases} t & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases}.$$

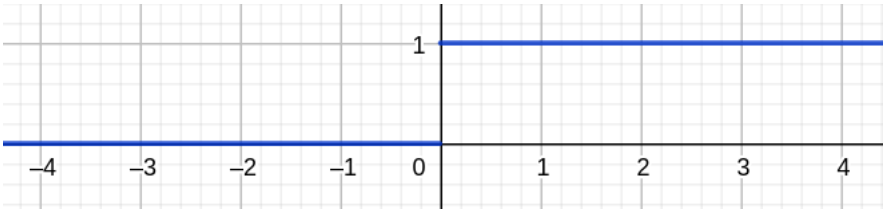
It is a continuous function, differentiable at all points but at $t = 0$. This may seem as an inconvenience for their use with the gradient descent algorithm. However, in practice it is not a concern because it is very unlikely that the input takes exactly the value 0, and usually one of the one-sided derivatives is returned for $t = 0$. Indeed, digital computers are subject to

errors and the absolute value 0 is never reached. Its derivative is the following step function:

$$\frac{d\text{ReLU}(t)}{dt} = \begin{cases} 1 & \text{if } t > 0 \\ \\ 0 & \text{if } t < 0 \end{cases}, \quad t \in \mathbb{R} \setminus \{0\}.$$



(a) ReLU activation function



(b) Derivative of the ReLU

Figure 6.6.: Graphs of the ReLu activation function and its derivative

ReLUs are similar to linear functions, being linear across half of their domain, what makes them easy to optimize. In addition, their derivative is large whenever the input is positive. As a result, these functions accelerate the convergence of gradient descent considerably. In addition, their computational cost is really cheap in comparison with sigmoid functions (which involve expensive operations), since they simply transform negative values into zero.

However, they are not zero-centered and not-bounded. The latter property may cause the gradients to explode if the weights have large values. This is known as the *exploding gradient problem*, and it can be seen as the inverse of the vanishing gradient problem. When large derivatives accumulate, the gradient will increase exponentially, resulting in large updates of the weights. This creates a very unstable model incapable of learning, maybe even ending up with overflow errors due to extremely big weights. As a result, the weights must be initialized to very small values.

Apart from the above drawbacks, ReLUs may experience the *dead neuron problem* (or *dying ReLU problem*). Since the activation function "deactivates" the neuron when the input is negative, there is a possibility that the weights will be updated in such a way that the neuron will output zero for any input (because of large negative bias term for example). The gradient in that case would be also zero, which means that the weights would not be updated. Therefore, the "dead" neurons would remain deactivated forever. A large part of the network could then become inactive during training, being unable to learn properly. This scenario is more likely when the learning rate is too high, so an appropriate selection of the learning

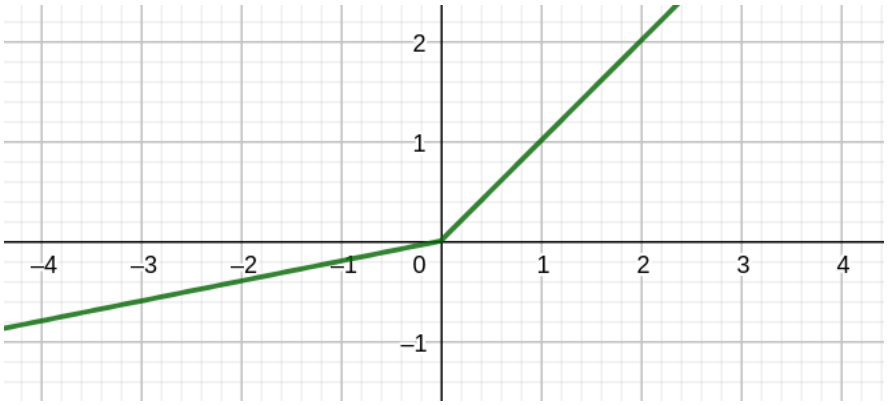
rate is required to alleviate the issue. Another usual solution is to initialize the bias to a small positive value to make sure that the ReLU neurons are initially active.

Some modifications of ReLU have been introduced as an attempt to avoid the dying ReLU problem. The **Leaky ReLU** uses a small slope $\alpha > 0$ for the negative input values (typically $\alpha < 1$), to avoid the 0 gradient when the input is negative:

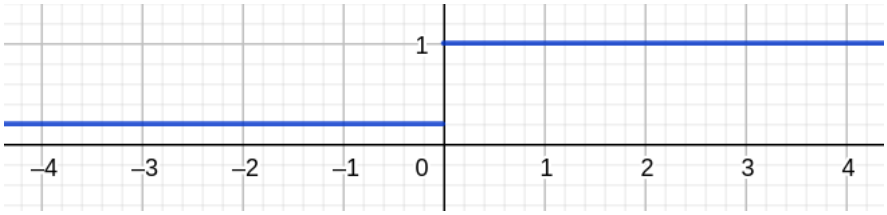
$$\text{LeakyReLU} : \mathbb{R} \to \mathbb{R}, \quad t \mapsto \begin{cases} t & \text{if } t \geq 0 \\ \alpha t & \text{if } t < 0 \end{cases}.$$

Its derivative can be calculated at every point but $t = 0$ as follows:

$$\frac{d\text{LeakyReLU}(t)}{dt} = \begin{cases} 1 & \text{if } t > 0 \\ \alpha & \text{if } t < 0 \end{cases}, \quad t \in \mathbb{R} \setminus \{0\}.$$



(a) LeakyReLu with $\alpha = 0.2$.



(b) Derivative of the LeakyReLu with $\alpha = 0.2$.

Figure 6.7.: Graphs of the LeakyReLu with $\alpha = 0.2$ and its derivative.

The LeakyReLU is zero-centered, but it presents a risk of vanishing gradient problem in the negative part, since the derivative is close to zero there.

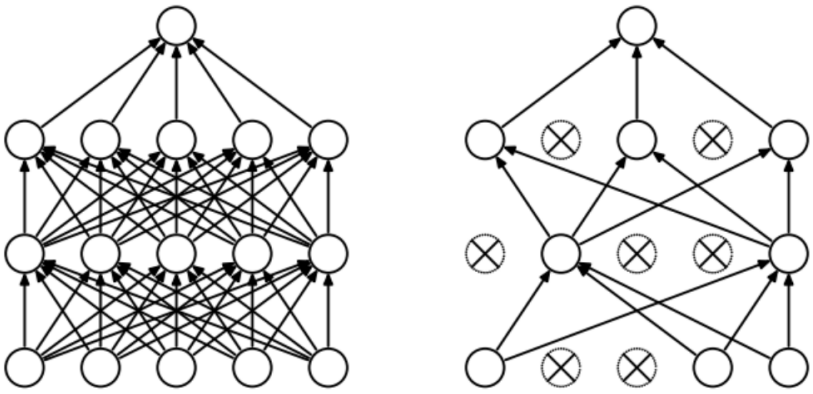
When α is made into a learnable parameter of each neuron, the activation function is called **Parametric ReLU, PReLU**.

In general, the ReLU function and its variants offer better performance and generalization in neural networks than the sigmoid functions. As a result, they are the most used nowadays in the hidden layers of neural networks, together with another activation function in the output layer.

6.4. Regularization techniques

6.4.1. Dropout

A simple but very effective regularization technique commonly used in deep learning is the method called *dropout* [23]. During each training iteration, individual hidden neurons of the network are deactivated (*dropped out*) with a probability $p \in [0, 1)$, so that the resulting network has a reduced size. In this way, a randomly chosen percentage p of hidden neurons are temporarily ignored, that is, not considered during the forward and back propagation. For example, for $p = 0.5$ half of the neurons are removed at random (see Figure 6.8). At the end of the training iteration deactivated neurons are restored and a new random set of hidden neurons is turned off during the next iteration.



(a) Standard Neural Net (b) After applying dropout.

Figure 6.8.: Example of dropout with $p = 0.5$. On the left we can see a neural network with two hidden layers and on the right the same network after applying dropout. Crossed units have been deactivated.

Since the entire network (without dropout) is used at test time, we have to compensate for the missing neurons during training by multiplying by p the weights associated with the layers where dropout was applied. That is, if a neuron is deactivated with probability p during training, then the outgoing weights of that neuron are multiplied by p at test time. By doing this, we make sure that the expected output of each neuron at training time (when dropout was used) is the same as the output at test time.

The main underlying idea of dropout is that, disconnecting certain neurons during training, may prevent them from co-adapting too much. In other words, without dropout, neurons are prone to develop co-dependency amongst each other, in such a way that each neuron specializes overly in a certain task limiting its power, what leads to overfitting.

Dropout increases the number of iterations needed for the optimization algorithm to converge, but the time required for each iteration decreases.

6.4.2. Batch normalization

We will explain now a technique that is used to ease the training of deep neural networks, known as *Batch Normalization* [24]. It is a method of *adaptive reparametrization* whose main

objective is to avoid the differences emerging among layers of a very deep neural network, what the authors of the method call *internal covariate shift*.

Without the use of techniques like this one it is unlikely for deep networks to be trained successfully with gradient descent and backpropagation. Since neural networks with many layers involve the composition of a lot of functions, unexpected results may arise when updating the weights of all layers using gradient descent. Many functions are changed simultaneously due to the updated weights that were calculated according to the gradient, which, in turn, determines the direction of update assuming that the functions in the other layers do not change (what is not the case).

Batch normalization consists in normalizing the inputs to each layer, what significantly alleviates the problem of coordinating the updates among layers. Let us consider a minibatch $\mathcal{B}$ of size M, $\mathcal{B} = \{x^1, \dots, x^M\}$, and a layer $t = 1, \dots, T$ with d_{t-1} inputs. Then, each input $x_j, j = 1, \dots, d_{t-1}$, to the layer is normalized by subtracting its mean and dividing by its standard deviation, both moments calculated over all the examples in the minibatch. That is, for each $j = 1, \dots, d_{t-1}$, we compute

$$\mu_j = \frac{1}{M}\sum_{i=1}^{M} x_j^i, \quad \sigma_j^2 = \frac{1}{M}\sum_{i=1}^{M}(x_j^i - \mu_j)^2,$$
$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}},$$

being ϵ a small positive value (e.g. 10^{-8}) to avoid division by zero.

Once the network has been trained, the values of μ_j and σ_j^2, $j = 1, \dots, d_{t-1}$, can be replaced by running averages computed during training (over the entire training dataset). In this way, at test time, the network can be evaluated on a single example without the need to compute the moments, which would require the use of more examples (but during inference we want the output to depend deterministically on the single input).

Normalizing the input of the layer in this way may affect the expressive power of the network. For example, in the case of sigmoid activation functions, the normalization may force the neuron to operate in their linear regime (near 0) what entails a loss of the non-linearity. To address this problem a couple of parameters are introduced for each input γ_j, β_j, which are learnable parameters used to scale and shift the normalized value as follows:

$$y_j = \gamma_j \hat{x}_j + \beta_j, \quad j = 1, \dots, d_{t-1} \tag{6.5}$$

This modification allows the normalized input to have any mean and variance. However, while in the original version the mean of the inputs was determined by a complex interaction between the parameters of the different layers, with this new reparametrization it only depends on the parameter β_j. In this way, the training process with gradient descent is much easier.

The authors recommend applying batch normalization to the weighted sum $s_{tj} = \sum_{i=0}^{d_{t-1}} w_{ij} x_i$, $j = 1, \dots, d_t$, instead of the original input x_t . More concretely, s_{tj} should be replaced by the normalization (6.5) of $\sum_{i=1}^{d_{t-1}} w_{ij} x_i$, where the bias term w_{0j} is omitted because it gets redundant with the use of β_j.

As we have explained in a previous section, a large value of the learning rate may result in exploding or vanishing gradients. The Batch normalization technique helps tackling these problems, enabling the use of higher learning rates and making the network more robust

to the parameters initialization. For instance, it prevents the neurons from getting saturated. Additionally, it has shown a regularization effect improving the generalization ability of the network (hence we have included this method in the section about regularization).

6.5. Expressive power of Neural Networks

In this section we shall discuss what type of functions can be implemented by neural networks depending on the activation functions used. In this way, at the same time we are determining the approximation error of the hypothesis classes of neural networks.

We start the analysis by focusing on the sign activation function, that is, on the multi-layer perceptron. The results presented for this type of functions and their respective proofs are obtained from [5] (Chapter 20).

Theorem 6.2. *For every $d_0 \in \mathbb{N}$ there exists a neural network architecture of depth $T = 2$ with the sign activation function such that $\mathcal{H}_{2,d,sign}$ contains all the functions from $\{\pm 1\}^{d_0}$ to $\{\pm 1\}$.*

Proof. Let $f : \{\pm 1\}^{d_0} \to \{\pm 1\}$ be an arbitrary Boolean function. For the proof we will construct a neural network with $d_1 = 2^{d_0}$ and $d_2 = 1$ and we have to show that we can adjust the weights of the network so that it implements f.

Let us consider all vectors in $\{\pm 1\}^{d_0}$ for which the Boolean function f outputs 1, that we will denote by $u_1, \dots, u_k$. It is clear that, for every $i = 1, \dots, k$ and every $x \in \{\pm 1\}^{d_0}$, $\langle x, u_i \rangle = n$ if $x = u_i$ and $\langle x, u_i \rangle \leq n - 2$ if $x \neq u_i$. We define the function $g_i(x) = sign(\langle x, u_i \rangle - n + 1)$ which is equal to 1 whenever $x = u_i$ and -1 otherwise.

By definition of the network, we can adjust the weights between the input layer and the hidden layer so that each neuron $i \in \{1, \dots, k\}$ in the hidden layer implements the function $g_i(x)$.

Finally, observe that $f(x)$ is the disjunction of the functions $g_i(x)$, that is

$$f(x) = sign\left(\sum_{i=1}^{k} g_i(x) + k - 1\right),$$

which can be implemented by the output neuron by setting the appropriate weights. □

In other words, the theorem states that every Boolean function can be represented by a neural network with only one hidden layer. However, since the size of the network is not limited, the number of required hidden neurons grows exponentially with the dimension of the input.

Theorem 6.3. *For every $n \in \mathbb{N}$, let $s(n)$ be the minimal integer such that there exists a neural network architecture $(T, d, sign)$ with $d_0 = n$ of size $s(n)$ whose associated hypothesis class $\mathcal{H}_{T,d,sign}$ contains all the functions from $\{\pm 1\}^n$ to $\{\pm 1\}$. Then, $s(n)$ is exponential in n.*

Proof. Let us assume that $\mathcal{H}_{T,d,sign}$ contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$, so that it can shatter the set of 2^n vectors in $\{0, 1\}^n$. Therefore, the VC dimension of this hypothesis class is 2^n. Additionally, Theorem 6.8 states that the VC dimension of $\mathcal{H}_{T,d,sign}$ is $O(w \log w) \leq O(h^3)$, being w the total number of weights of the network and h its size. As a result, $2^n \leq O(h^3)$, which yields $h \geq \Omega(2^{n/3})$, which concludes the proof. □

Moving to logistic sigmoid activation functions, we will present the **Universal Approximation Theorem** of Cybenko [2].

Let us consider the function

$$G(x) = \sum_{i=1}^{k} \alpha_i \sigma \left(\sum_{j=1}^{n} w_{ij} x_i + b_i \right) = \sum_{i=1}^{k} \alpha_i \sigma \left(w_i^T x + b_i \right), \quad \forall x \in \mathbb{R}^n, \tag{6.6}$$

where $k, n \in \mathbb{N}$, $\alpha_i, \beta_i \in \mathbb{R}, \forall i = 1, \ldots, k$, and $w = \left[w_{ij} \right] \in \mathbb{R}^{k \times n}$. The scalar function $\sigma : \mathbb{R} \to \mathbb{R}$ fulfills the following condition

$$\lim_{t \to -\infty} \sigma(t) = 0 \quad \text{and} \quad \lim_{t \to \infty} \sigma(t) = 1,$$

and is what Cybenko calls a *sigmoidal* function. Note that this term differs from the logistic sigmoid functions we defined in section 6.3, which are a particular case of sigmoidal functions as considered by Cybenko.

The function G is implemented by a neural network with one hidden layer with k neurons (without bias), n inputs (plus bias) and one output neuron. All the hidden neurons use a sigmoidal activation function, while the output neuron applies the identity function. The weight matrix of the hidden layer is w^T and the weights on the edges going out from the bias node of the input layer, which are not included in w, are given by $b_1, \ldots, b_k$, for each neuron $i = 1, \ldots, k$ in the hidden layer. The vector $\alpha = [\alpha_1, \ldots, \alpha_k]$ represents the weights associated with the output layer.

Let I_n denote the n-dimensional unit cube $[0, 1]^n$ and let $C(I_n)$ be the space of real-valued continuous functions defined on I_n, with the supremum norm $\| \cdot \|_\infty$[1]. Then, the Universal Approximation Theorem states the following:

Theorem 6.4. *(*Universal Appproximation Theorem; G.Cybenko, 1989*) Let σ be any continuous sigmoidal function. Then, the finite sums of the form (6.6) are dense in $C(I_n)$ with respect to the supremum norm. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$ there exist $k \in \mathbb{N}$, $\alpha_i, \beta_i \in \mathbb{R}, \forall i = 1, \ldots, k$, and $w = \left[w_{ij} \right] \in \mathbb{R}^{k \times n}$ such that the function G described in (6.6) satisfies*

$$|G(x) - f(x)| < \varepsilon \quad \text{for all } x \in I_n.$$

In addition, the theorem is still valid when I_n is replaced by any other compact subset $K \subset \mathbb{R}^n$.

Since the logistic sigmoid function defined in section 6.3 fulfills the conditions of the above theorem, we can conclude that neural networks with one hidden layer and logistic sigmoid activation functions can approximate any continuous function on a compact domain to any desired precision, provided that no constraints are set on the size of the network. However, as in the case of Boolean functions, it is not possible to implement all continuous functions with a network of size polynomial in n. Indeed, the size of the resulting network is exponential in n, which can be proven by means of Theorem 6.3.

Observation 6.2. It is sufficient to consider neural networks with only one output unit, because any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be approximated by computing each of its components $f_j : \mathbb{R}^n \to \mathbb{R}$, $j = 1, \ldots, m$.

In the case of classification functions, Cybenko also provides an approximation theorem as a consequence of Theorem 6.4. Let λ be the Lebesgue measure on I_n and let $P_1, \ldots, P_k$ be a

[1] $\|f\|_\infty = \max\{|f(x)| : x \in I_n\}, \forall f \in C(I_n)$.

partition of the unit cube into k disjoint and measurable subsets. We define the classification function $f : I_n \to \{1, \dots, k\}$ as follows:

$$f(x) = j \iff x \in P_j. \tag{6.7}$$

We have the following result:

Theorem 6.5. *(G.Cybenko, 1989) Let σ be a continuous sigmoidal function. Let f be the classification function for any finite measurable partition of I_n defined as in (6.7). For any $\varepsilon > 0$ there exist a function G of the form described in (6.6) and a set $D \subset I_n$ with $\lambda(D) \geq 1 - \varepsilon$ such that*

$$|G(x) - f(x)| < \varepsilon \quad \textit{for all} \quad x \in D.$$

As the previous Theorem 6.4, this result remains valid when replacing I_n by any other compact subset of $\mathbb{R}^n$.

Theorem 6.5 claims that, given any classification function on a compact domain, there exists a neural network with one hidden layer and logistic sigmoid activation functions that approximates the classification function arbitrarily well and makes the measure of the set of points incorrectly classified as small as desired.

Later, in 1993, M. Leshno et al. [25] proved a generalization of the Universal Approximation Theorem, where other activation functions, like the hyperbolic tangent, were included:

Theorem 6.6. *A standard multilayer feedforward network with one hidden layer and locally bounded piecewise continuous activation functions can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not polynomial (a.e.), provided that sufficiently many hidden neurons are available.*

These results show that limited depth feedforward neural networks provide a very expressive hypothesis space. But, what happens when other type of constraints are placed on the network? Lu et al. carried out a study in 2017 to determine how the width affects the expressive power of a network, which resulted in an **Universal Approximation Theorem for Width-Bounded ReLU Networks** [26]:

Theorem 6.7. (Universal Approximation Theorem for Width-Bounded ReLU Networks; Lu et al. 2017) *For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a fully-connected network with ReLU activation functions $(T, d, ReLU, w)$ with width $d_m \leq n + 4$ such that the function $h_{T,d,ReLU}(\cdot, w) : \mathbb{R}^n \to \mathbb{R}$ implemented by this network satisfies*

$$\int_{R^n} |f(x) - h_{T,d,ReLU}(x, w)| dx < \varepsilon.$$

Therefore, any continuous function on $\mathbb{R}^n$ can be approximated to arbitrary accuracy by a deep network with ReLU neurons and width at most $n + 4$.

Moreover, Lu et al. proved that it is not possible to approximate all functions by ReLU networks of n-width, except for a set of measure zero.

Although all the above results indicate that the approximation error of neural networks can be made as small as desired, we must be careful because if the number of hidden neurons become too large, we may lose generalization ability, as we will see in the next section.

6.6. VC dimension of Neural Networks

In order to determine the estimation (or generalization) error of neural networks we must calculate the VC dimension of each class $\mathcal{H}_{T,d,\theta}$. We will present different results (as in [27]), depending on the type of activation functions that the neurons constituting the network use, focusing on the case of the binary classification problem, that is, the network has only one output neuron with a simple threshold activation function: $\theta_T(u) = 1$ if $u \geq 0$ and 0 otherwise, $\forall u \in \mathbb{R}$.

Intuitively, we can think that the VC dimension of a neural network should depend on the number of weights, since the weights are what determine a hypothesis of the class and they are the parameters that must be learned. We will see that it is actually the case.

For networks with activation functions that take only two values depending on a certain threshold (linear threshold functions), we have the following result:

Theorem 6.8. (Cover, 1968; Baum and Haussler, 1989 [28]) *The class of functions computed by multilayer feedforward neural networks with linear threshold activation functions and a number w of weights has VC dimension $O(w \log w)$.*

The sign function is included in this class of activation functions. Therefore, we can state that the VC dimension of the Multi-layer Perceptron introduced in section 6.1 is $O(w \log w)$.

In 1995, Akito Sakurai proved an upper and lower bound for a large class of neural networks, where the sigmoid functions are included.

Theorem 6.9. (Sakurai, 1995 [29]) *Let (T, d, θ) be the architecture of a feedforward neural network with $d_T = 1$ and $\theta_T = \mathbb{1}_{[0,\infty)}$. Let us denote by $h = \sum_{t=1}^{d_{T-1}} d_t$ the total number of neurons in the hidden layers and let w be the number of weights. If the activation functions of the hidden neurons are logistic sigmoid functions, then*

$$O(w \log h) \leq d_{VC}(\mathcal{H}_{T,d,\theta}) \leq O(w^2 h^2).$$

If the activation functions are hyperbolic tangent functions, it can be proven that the VC dimension of the network is $O(hw)$ ([6], Chapter 7).

Observation 6.3. We can see that the VC dimension of a network with sigmoid activation functions is higher that the one of the MLP. This was expected because a neural network with sigmoid neurons can approximate any of the binary classification functions implemented my the MLP to arbitrary accuracy, as Theorem 6.5 states. Therefore, every subset that can be shattered by the MLP can also be shattered by a neural network with sigmoid neurons.

We now suppose that each hidden neuron has an activation function that is piecewise polynomial, that is, it is of the form

$$\theta(u) = \phi_i(u), \text{ for } u \in [a_{i-1}, a_i), \quad i = 1, \dots p+1,$$

where each ϕ_i is a polynomial of degree at most l, p is the number of break-points and $a_0 = -\infty$, $a_{p+1} = \infty$. When $l = 0$ we say that the function is piecewise constant and for $l = 1$ it is piecewise linear.

Theorem 6.10. (Bartlett, Maiorov, Meir, 1998 [30]) *For any positive integers w, $h \leq W$, $T \leq W$, l and p, consider a network with real inputs, up to w parameters, up to h hidden neurons arranged in T*

layers, a single output neuron with the activation function $\mathbb{1}_{[0,\infty)}$, and all other hidden neurons with piecewise polynomial activation functions of a fixed degree l and with a fixed number of break-points p. Let $\mathcal{H}_{T,d,\theta}$ be the class of real-valued functions computed by this network. Then, $d_{VC}(\mathcal{H}_{T,d,\theta}) = O(wT\log w + wT^2)$

In particular, if the number of layers T is fixed, then the VC dimension is $O(w\log w)$, the same as for linear threshold functions, which are piecewise constant functions.

Surprisingly, the change from piecewise constant to piecewise polynomial functions does not increase the rate of growth of the VC dimension in the case of networks with a fixed number of layers. However, when the number of layers is not bounded, the rate of growth is faster for piecewise polynomial functions.

Note that ReLU and its variants are piecewise polynomial, so they belong to this category of activation functions. However, recently, in 2017, Bartlett, Harvey, Liaw and Mehrabian, discovered tighter bounds for the particular case of piecewise linear functions, such as ReLU:

Theorem 6.11. (Bartlett, Harvey, Liaw and Mehrabian, 2017 [31]) *Let us consider a feedforward neural network with T layers, w weights and a single output neuron with the activation function $\mathbb{1}_{[0,\infty)}$. If all the hidden neurons use piecewise linear activation functions with a fixed number of break-points, denoting by $\mathcal{H}_{T,d,\theta}$ the class of real-valued functions computed by this network, then there exist constant c, C such that*

$$c \cdot wT\log(w/T) \leq d_{VC}(\mathcal{H}_{T,d,\theta}) \leq C \cdot wT\log(w).$$

7. Convolutional Neural Networks

Convolutional neural networks (LeCun , 1989), CNNs or ConvNets for short, are a specialized kind of neural networks for processing (discrete) signals, i.e. data that present a grid-like topology, such as sounds (1D signals), images (2D signals) or videos (3D signals). They have shown great success in different practical applications, specially in the *computer vision* field. ConvNets are based in the use of a mathematical operation called *convolution*.

In this chapter, we first introduce the convolution operation. Next, we describe the most important layers constituting a convolutional neural network, namely convolutinal layers. After that, other types of layers typically used in CNNs are introduced, along with their purpose and functioning.
We have mainly followed [21], [8] and [22] to write this chapter.

7.1. The convolution operation

Convolutional neural networks are simply feedforward neural networks where some layers employ a linear operation known as *convolution*, hence the name of these networks, instead of the general matrix multiplication of the weight matrix by the inputs to the layer. The operation used in convolutional neural networks does not correspond exactly to the mathematical definition of convolution. However, it is closely linked to it, as we shall explain in this section.

In the functional analysis field the convolution is a mathematical operation on two real-valued functions f, g that produces a third function, denoted by $f * g$, which is usually seen as a modification of one of the functions by the other. It is defined as the integral of the product of f and g after one of them is reversed and shifted, whenever the integral exists. Formally:

Definition 7.1. (Convolution operation) [32] Let $f, g : \mathbb{R} \to \mathbb{R}$ be two Lebesgue-measurable functions. For each $x \in \mathbb{R}$, the function $t \mapsto f(x-t)g(t)$ is Lebesgue-measurable and we say that the *convolution* of f and g, $f * g$, is defined at point x if the following condition holds

$$\int_{\mathbb{R}} |f(x-t)g(t)|dt < \infty,$$

in which case

$$(f * g)(x) := \int_{\mathbb{R}} f(x-t)g(t)dt.$$

The term convolution is used to refer both to the mathematical operation itself and the resulting function. Geometrically, the convolution can be interpreted as the area under the function $g(t)$ weighted by $f(-t)$ shifted by the amount x. In this way, as x changes, the different parts of $g(t)$ are emphasized by the weighting function $f(x-t)$.

The convolution is a linear operator, and, as such, it presents several important properties, such as the commutative, associative and distributive properties.

In digital signal processing, where we work with discrete signals, the *discrete convolution* is used instead, which is defined for sequences of the ℓ_1 Banach space. The following results for the discrete case have been obtained from [33], as well as [34].

Definition 7.2. (Discrete convolution) Let $x, y : \mathbb{Z} \to \mathbb{R}$ be two sequences in the $\ell_1(\mathbb{Z})$ Banach space, that is, such that

$$\sum_{k\in\mathbb{Z}} |x_k| < \infty, \quad \sum_{k\in\mathbb{Z}} |y_k| < \infty.$$

Then, for each $j \in \mathbb{Z}$, it holds

$$\sum_{k\in\mathbb{Z}} |x_{j-k} y_k| < \infty,$$

and we can define the sequence $x * y$, called the *convolution* of x and y, as follows

$$(x * y)_j := \sum_{k\in\mathbb{Z}} x_{j-k} y_k, \quad j \in \mathbb{Z},$$

which is in $\ell_1(\mathbb{Z})$.

The discrete convolution shows the same properties as the continuous convolution defined in Def. 7.1.

We can generalize the definition of convolution to multidimensional domains. First, let us introduce the following definition:

Definition 7.3. (Support) Let X be an arbitrary set and $f : X \to \mathbb{R}$ a real-valued function. Then, the support of f, denoted by $\text{supp}(f)$, is the following subset

$$\text{supp}(f) = \{x \in X : f(x) \neq 0\} \subset X.$$

Definition 7.4. (Multidimensional convolution) [35] Let $f, g : \mathbb{R}^n \to \mathbb{R}$ be two continuous functions with compact support[1]. Then, the convolution of f and g is defined at point $x \in \mathbb{R}^n$ as follows

$$(f * g)(x) := \int_{\mathbb{R}^n} f(x - t) g(t) dt,$$

and it is continuous with compact support.

And for the discrete case:

Definition 7.5. (Multidimensional discrete convolution) Let $x, y : \mathbb{Z}^n \to \mathbb{R}$ be two functions with bounded support. We define the n-dimensional discrete convolution of x and y at point $j = (j_1, \ldots, j_n) \in \mathbb{Z}^n$ as the function

$$(x * y)(j) = x(j_1, \ldots, j_n) *^{n} \cdots * y(j_1, \ldots, j_n) := \sum_{k_1\in\mathbb{Z}} \cdots \sum_{k_n\in\mathbb{Z}} x(j_1 - k_1, \ldots, j_n - k_n) y(k_1, \ldots, k_n).$$

Whenever the convolution is defined and either f or g is differentiable, the convolution is

[1] Remember that a subset of the Euclidean space $\mathbb{R}^n$ is compact if, and only if, it is closed and bounded.

differentiable with

$$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}.$$

In the case of multidimensional domains we have an analogous expression for the partial derivatives :

$$\frac{\partial}{\partial x_i}(f * g) = \frac{\partial f}{\partial x_i} * g = f * \frac{\partial g}{\partial x_i}, \ \ \forall i = 1, \ldots, n.$$

If one of the functions, f or g, involved in the convolution is reflected, we obtain the *cross-correlation* operation:

Definition 7.6. (Cross-correlation) Let $f, g : \mathbb{R} \to \mathbb{R}$ be two Lebesgue-measurable functions. For each $x \in \mathbb{R}$, the *cross-correlation* of f and g, $f \star g$, is defined at point x as

$$(f \star g)(x) := \int_{\mathbb{R}} f(t)g(t + x)dt,$$

or, equivalently,

$$(f \star g)(x) := \int_{\mathbb{R}} f(t - x)g(t)dt,$$

which is equal to the convolution of $f(-t)$ and $g(t)$.

This operation is not commutative, but it is easy to see that $(f \star g)(x) = (g \star f)(-x)$.

As for the convolution operation, a definition for discrete functions can be formulated:

Definition 7.7. (Discrete Cross-correlation) Let $x, y : \mathbb{Z} \to \mathbb{R}$ be two discrete functions. The *cross-correlation* of x and y, denoted by $x \star y$, is defined for each $j \in \mathbb{Z}$ as

$$(x \star y)(j) := \sum_{k \in \mathbb{Z}} x(k)y(k + j),$$

or, equivalently,

$$(x \star y)(j) := \sum_{k \in \mathbb{Z}} x(k - j)y(k),$$

which is equal to the discrete convolution of $x(-k)$ and $y(k)$.

Both definitions of cross-correlation can be extended to the multidimensional case in the same way as convolution.

Cross-correlation is commonly used as a measure of similarity between two functions and it is sometimes called a *sliding dot product*, since it is calculated by sliding one function across the other and computing the dot product at each position.

The operation used in convolutional neural networks is cross-correlation, but it is often referred to as convolution instead, since the difference between both operations is minimal.

Normally, in the digital signal processing field, one of the functions that appears in the cross-correlation (or convolution) is the input signal that we want to process, x, and the other is a *filter*, or *kernel*, w, used to process the signal. When the filter is finite, that is, the function w is defined over a finite set of integers $\{0, 1, \ldots, K - 1\}$, the cross-correlation operation consists

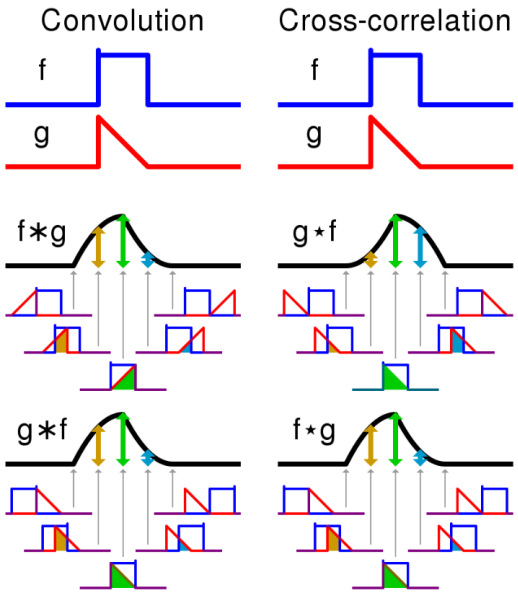


Figure 7.1.: Visual comparison of convolution and cross-correlation of two continuous functions f and g. Source Cross-correlation.

in K multiplications and $K-1$ additions for each value of the signal:

$$(x \star w)(j) = \sum_{k=0}^{K-1} w(k)x(k+j)$$

In deep learning, both the input signal and kernel are multidimensional arrays, known as *tensors*. Only their values for a finite set of points can be stored in memory, so the cross-correlation can be computed as a finite summation over the elements of the array.

Let us show an example of cross-correlation for discrete 2D-signals (images) in order to make the concept more clear.

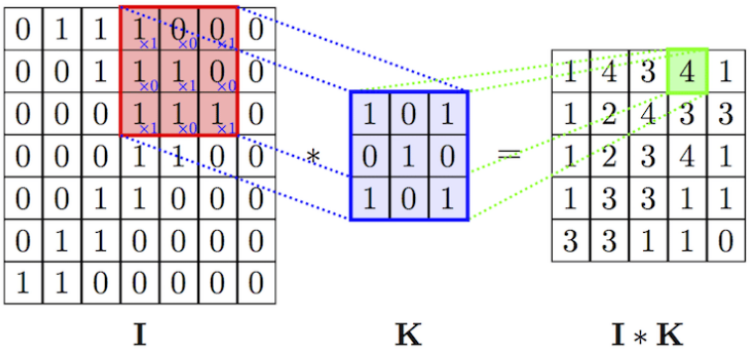


Figure 7.2.: Example of cross-correlation operation between two discrete 2D-signals. Image from https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-30_fig5_349391561.

In Figure 7.2 we have a black and white input image $\mathbf{I}$ of size 7×7 and we want to apply a 4×4 filter $\mathbf{K}$ to it via cross-correlation. To obtain the output filtered image, the filter must be shifted along the input image. For each shift, an element-wise multiplication is performed between the kernel and the image pixel values that match the size of the kernel and then the results are added up, providing a single value of the output. For example, for the position shown in the figure, the output is calculated as follows: $1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 = 4$. By repeating the process at all positions where the filter can be placed over the image, the result of the cross-correlation operation $I * K$ is obtained.

7.2. Convolutional layer

Convolutional layers are the main building block of convolutional neural networks. They receive a 1D signal (such as a sound), 2D signal (such as an image) or a 3D signal (such as a video or a 3D image) as input, and they process it by convolution (specifically, by cross-correlation) with a kernel. The values of the kernel, plus an optional bias term, are the parameters of the layer (i.e. the weights) that must be learned during training. After the convolution, an activation function is applied to the result, producing the output of the convolutional layer. We will focus in the case where the input to the layers is an image with a certain width and height.

Intuitively, a convolutional neural network will learn a filter for each layer that detects only a particular feature in the input, such as an edge with a certain orientation or a spot of some color. Therefore, it makes sense to include a set of filters in each convolutional layer, each of them corresponding to a set of parameters to be learned. The output produced by each filter is a signal of the same type of the input and we call it a *feature map*. These feature maps are stack along the depth dimension, generating an output volume. In this way, the output of a convolutional layer with K filters will consist of K different feature maps, and, therefore, it will have depth K. As a result, the layers of a convolutional neural network are arranged in volumes, where the depth corresponds to the number of filters of each layer.

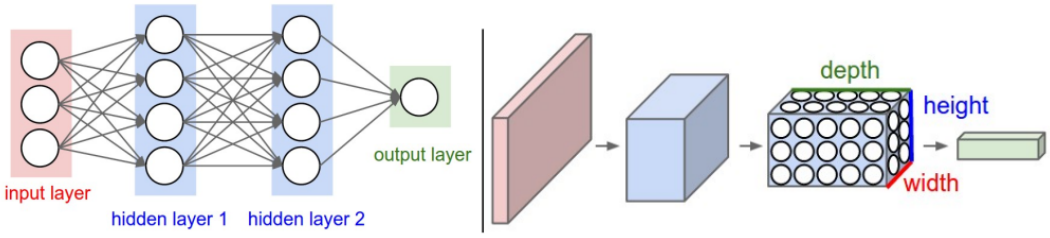


Figure 7.3.: Comparison between a regular 3-layer neural network and a convolutional 3-layer neural network. Each convolutional layer transforms the 3D input, to a 3D output volume.

A convolutional layer is characterized by the following two key feutures, which differentiates it from the fully-connected layers of a regular neural network.

- **Local connectivity**. Unlike fully-connected layers, each neuron of a convolutional layer is connected only to a local region of the input. The spatial extent of this connectivity

is known as the *receptive field* and it is a hyperparameter[2] which coincides with the size of the filters. The extent of the receptive field along the depth dimension must be equal to the depth of the input volume. That is, the connections are local along width and height, but full along the entire depth of the input volume. The spatial extent of the local connectivity of each neuron is usually set to be the same for the width and height dimensions.

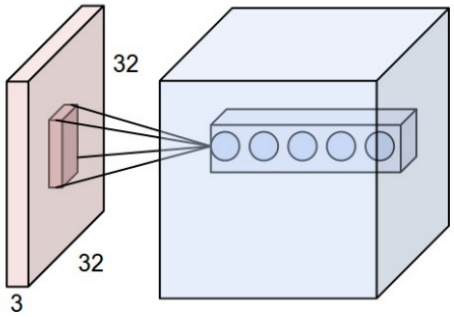


Figure 7.4.: Receptive field of a set of neurons. The red volume represents the input image and the blue volume is a convolutional layer with depth 5. Each neuron in the convolutional layer is connected only to a local region of the input along the width and height, but to the full depth. All the five neurons along the depth dimension share the same receptive field (but not the same weights, since they are associated to different filters). The set of neurons along the depth dimension that have the same receptive field is sometimes called a *depth column*, or a *fibre*.

The motivation for local connectivity is the local correlation property of images (or other input data types, such as temporal signals). That is, in an image, pixels that are close to each other are more likely to be correlated than pixels that are very distant. Therefore, it makes sense to suppress connections between far away neurons, because they are unlikely to be correlated, and to keep the connections to neurons that are likely to share useful information.

- **Parameter sharing**. This property is motivated by the following assumption: if a neuron detects an useful feature at some particular region of the input, then it is reasonable to believe that it would be useful to detect the same feature at a different region of the input. For this reason, all the neurons in each 2D slice which is arranged along the depth dimension are constrained to use the same weights and bias (the same kernel). In this way, different neurons whose entries are connected to different regions of the input will share the same parameters, and the network will be able to detect the same particular feature at any position of the input image. This constraint is what motivates the use of the convolution operation in the forward propagation.

The number of parameters that must be learnt for each convolutional layer is reduced compared to fully connected layers. Let us assume that the input to a layer has m components and that the layer has n neurons. A fully-connected layer would need mn parameters, while a convolutional layer using N filters of $k < m$ components, would only need kN parameters

[2]Hyperparameters are parameters of the network that are not learned, but fixed before the training process by the user.

(or $(k+1)N$ including the bias terms). This difference is of great importance in the particular case of multidimensional input signals, whose size is much larger compared to 1D inputs. The reduction in the number of parameters makes the training of the network easier and decreases the probability of overfitting. Additionally, it allows to include more filters in each layer and a higher number of layers, so that more features can be detected in the input image in an effective way (the expressive power of the network increases).

Convolutional layers present an important property, which is inherited from the convolution (or cross-correlation) operation: they are equivariant to translation. This means that if the input volume to the layer is shifted, then the output will be also shifted in the same way. Mathematically, a function f is equivariant to a transformation τ if, and only if, $f(\tau(x)) = \tau(f(x))$.

7.2.1. Hyperparameters of a convolutional layer

When designing a convolutional layer, the values of four main hyperparameters must be specified. These hyperparameters determine the size of the output volume. We have already introduced two of them: the **number of filters** K, each of them learning to detect a different feature in the input, which corresponds to the depth of the output, and the **size of the filters** F (i.e. the size of the receptive field). The two remaining are the **stride** and **zero padding**:

- The *stride*, S, determines the step size when sliding the kernel over the input to compute the convolution. When the stride is 1, we slide the filter one pixel at a time, that is, we compute a normal convolution. When the stride is 2, we move the filter 2 pixels at a time, skipping in this way one in two pixels, and so on. In the case of multidimensional input signals, we can establish a different value of the stride for each dimension.

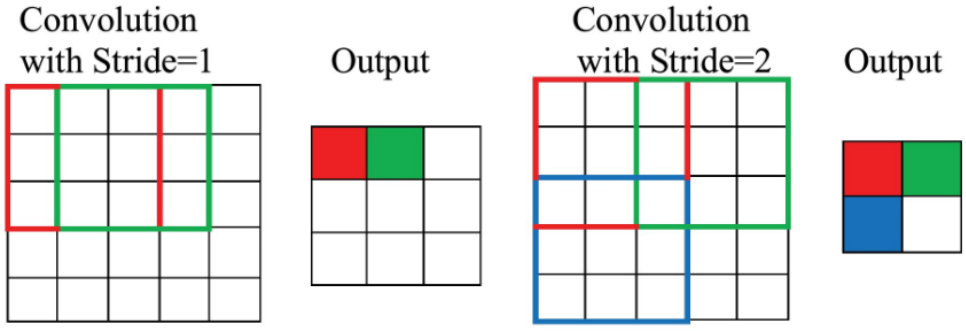


Figure 7.5.: Computation of a 2D convolution with a stride of 1 and a stride of 2 along both dimensions, with a kernel of size 3×3. Source https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/.

  When the stride is set to a value greater than one, the number of arithmetic operations needed, as well as the size of the output, is reduced. Thus, the computational cost of carrying out the convolution is also less, but at the expense of losing some level of detail when detecting features in the input. Such type of convolution is called a *downsampled convolution*.

- Sometimes it is useful to pad the borders of the input with zeros to obtain an output volume of the desired size. The amount of *zero padding*, Z, is an hyperparameter.

Without padding, the convolution can only be computed at positions where the entire kernel is completely contained within the input. In this case, every spatial dimension of the output shrink by one pixel less than the corresponding dimension of the kernel at each layer, so we must choose between a fast reduction of the spatial dimensions of the outputs and using small kernels. If the kernels used are large, the rate of shrinkage can become meaningful, limiting the number of convolutional layers that the network can contain. As convolutional layers are included, the spatial dimensions of the output shrink, eventually dropping to a size of 1×1. With padding, the size of the kernel and the size of the output become independent.

Zero padding is commonly set to preserve the spatial dimensions of the input, so that we can add as many convolutional layers as desired. The amount of padding needed to achieve that, when the stride is 1, is given by $Z = (F-1)/2$ for each border.

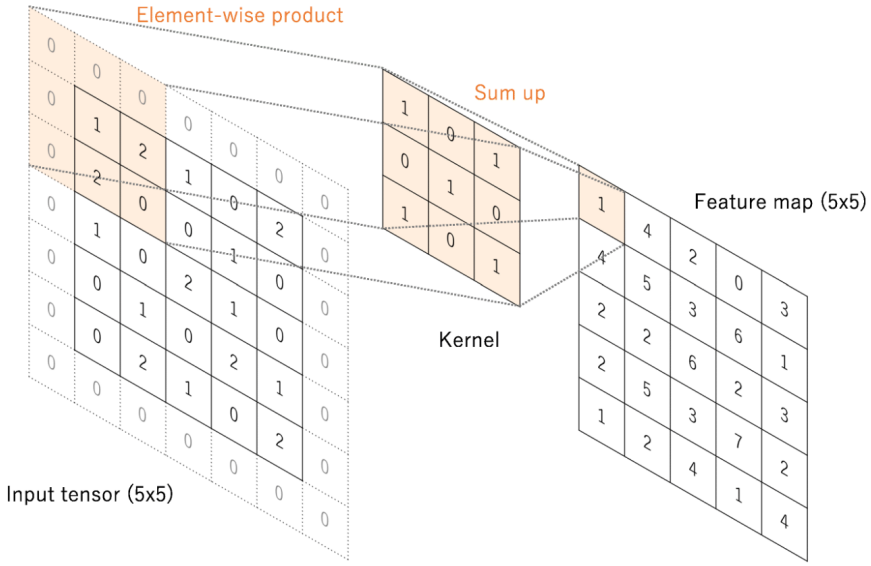


Figure 7.6.: Example of a 2D convolution with zero padding to preserve the input spatial dimensions. The size of the input and the size of the output are both 5×5. Figure obtained from [36].

Instead of padding with zeros, another option is to extent the borders by replicating the pixels of the closest border until the desired size is obtained.

The size of the output volume of a convolutional layer is then a function of the spatial dimensions of the input image, $W \times H$, the number of filters K, the size of the filters, $F \times F$ (with $F < H$ and $F < W$), the stride, (S_W, S_H), and the amount of zero padding at each border, Z. Specifically, the dimensions of the output are the following:

$$\text{width} = \frac{W - F + 2Z}{S_W} + 1, \quad \text{height} = \frac{H - F + 2Z}{S_H} + 1 \quad \text{and} \quad \text{depth} = K.$$

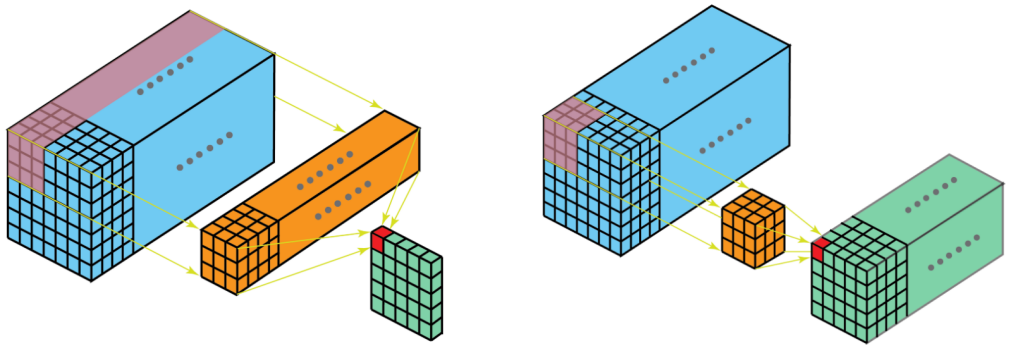
The hyperparameters must me adjusted in such a way that the dimensions of the output given by the above expressions are integer values.

7.2.2. 3D Convolutional layer

As we have already mentioned, the input to a convolutional layer can also be a 3D image, such as *Magnetic Resonance Images (MRI)*. In this case, 3D convolutional layers are used instead of the 2D convolutional layers described above.

In 3D convolutional layers, the filters used are 3 dimensional, and they slide in 3 dimensions, as opposed to 2D convolutions, where the kernel moves only in 2 dimensions. Multiple 3D filters can be used in each 3D convolutional layer, which are stack along a 4th dimension, resulting in a 4 dimensional output. The 3D input image may also consist of multiple channels, so the input and output of a 3D convolutional layer is 4 dimensional data. Therefore, the connections of the neurons of this type of layers to the input are local along all the 3 spatial dimensions and full along the 4th dimension.

Note that 3D convolution is different to multi-channel 2D convolution used in 2D convolutional layers. While in multi-channel 2D convolution the filters must have the same depth as the number of channels of the input (recall that we have full connection along the depth dimension) resulting in an output of depth 1 (2D output), in 3D convolutions the depth of the filters is usually smaller than the depth of the input, which produces an output of 3 dimensions.



(a) Multi-channel 2D convolution

(b) 3D convolution

Figure 7.7.: Comparison between multi-channel 2D convolution and 3D convolution. Source https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

7.2.3. 1×1 convolutions

Sometimes, filters of size 1×1 are used in convolutional layers. In the case of 2 dimensional data, a 1×1 convolution simply scales the input by the value of the filter. However, as convolutional layers work with multi-channel (3 dimensional) data, a 1×1 convolution can be employed to reduce the dimensionality of the input. Indeed, given an input volume of size $W \times H \times D$, after applying a 1×1 convolution with a filter of size $1 \times 1 \times D$, we obtain a 2 dimensional output of size $W \times H \times 1$. By applying K of such filters and concatenating the results, we can obtain an output volume with depth $K < D$. In this way, 1×1 convolutions combine different channels of the input by using the values of the kernel and allow us to learn from the interactions between channels.

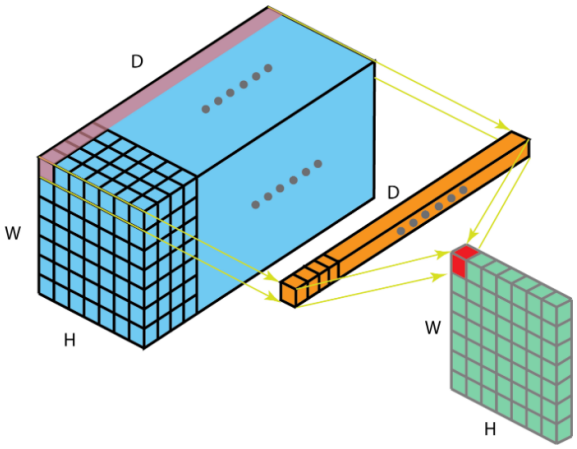


Figure 7.8.: Example of a 1×1 convolution applied to an input volume with multiple channels. Source https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

Besides their use for dimensionality reduction, convolutional layers using 1×1 convolutions also include a non-linear activation function, what allows the network to learn more complex features.

It is common to use 1×1 convolutions in the last layers of the network, since the first layers need to use larger filters in order to be able to capture the local spacial information.

7.2.4. Backpropagation

It can be proven that the backward pass of a convolutional layer is also a convolution. Additionally, during backpropagation, the gradient of the loss with respect to the weights is computed for each neuron of the layer. Since the same weights are used for all neurons in a 2D slice arranged along the depth dimension, those gradients must be added up across the neurons in the 2D slice. This addition of the local gradients is then used to update the weights, so that the set of weights shared by the slice of neurons are updated in the same way.

For a full explanation and proof of backpropagation in convolutional layers see [37].

7.3. Pooling layer

Another commonly used operation in convolutional neural networks is pooling. The purpose of a pooling layer is to progressively reduce the size of the feature maps, so that the number of parameters is also reduced, what helps preventing overfitting and decreasing the computation time needed. The pooling operation can be used to reduced the spatial size of the data, known as *spatial pooling*, which is normally the case, or to reduce the depth of the volumes (i.e. the number of channels), called *cross-channel pooling*, that is what 1×1 convolutions do. We will focus on spatial pooling. More concretely, what a pooling function does is to replace the output of a convolutional layer at a certain position with a summary statistic of the neighbor values of the output at that position. Different pooling functions have been used in the literature, but the most popular are the following:

- **Max Pooling**. The maximum value within a rectangle neighborhood of the output at each position is returned.
- **Average pooling**. At each position of the output, the average of the values within a rectangle neighborhood at that position is calculated.
- L_2 **norm pooling**[3]. The L_2 norm of the values within a rectangle neighborhood of the output at each position is computed.

In practice, it has been observed that max pooling performs better than the other types.

Just as convolutional layers, pooling layers depend on hyperparameters that must be adjusted:

- The size of the neighborhood considered to produce each value of the output of the pooling layer, that is, the *spatial extent* of the pooling operation, F. Normally, the spatial extent is the same vertically and horizontally.
- The *stride*, S, that is, the step size over the input of the pooling layer to obtain consecutive values of the output. Typically, the slide is chosen to be the same as the spatial extent, so that there is no overlapping.

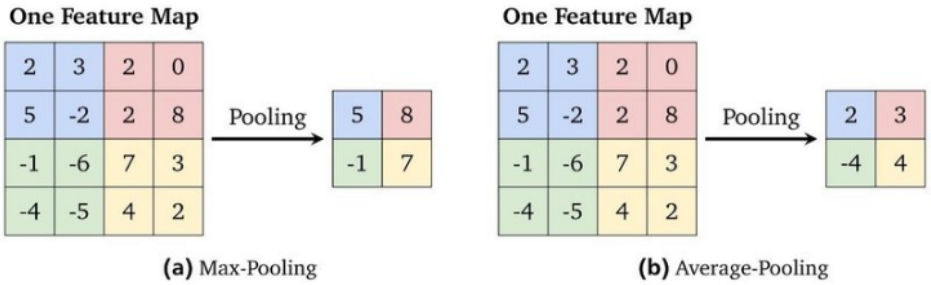


Figure 7.9.: Example of max pooling and average pooling operations with $S = 2$ and $F = 2$ in the two dimensions. Both spatial dimensions of the input have been halved, discarding the 75% of the input values. Image obtained from https://www.researchgate.net/figure/Example-for-the-max-pooling-and-the-average-pooling-with-a-filter-size-of-22-and-a_fig15_337336341.

Since the main objective of pooling layers is to reduce the dimensions of the input, it makes no sense to use zero padding in such layers.

In a convolutional neural network, pooling layers are normally inserted in-between consecutive convolutional layers. Therefore, a spatial pooling layer accepts an input volume of size $W \times H \times D$, which is the output volume of a previous convolutinal layer. If the spatial extent of the pooling operation is given by $F \times F$, and we use stride of S in both dimensions, the output volume of the pooling layer will consist of D channels, each of them with a width of $(W - F)/S + 1$ and a height of $(H - F)/S + 1$. Note that the depth of the volume remains unchanged because the spatial pooling operation is applied independently on every 2D feature map.

[3]Given a matrix $A = (a_{ij})$ in the space $\mathbb{R}^{n \times m}$, the L_2 norm is defined as $\|A\|_2 = \sum_{j=1}^{m} \sum_{i=1}^{n} |a_{ij}|^2$. It is also known as the *Frobenius norm*.

Generally, only two spatial pooling layers settings are used in practice: a pooling layer with $F = 3$ and $S = 2$, with overlapping (so it is sometimes called *overlapping pooling*), and, most commonly, a max pooling layer with $F = 2$ and $S = 2$. Larger spatial extents of the pooling operation are too destructive, so they are not frequent in practice.

In contrast to other types of layers, pooling layers do not introduce any learnable parameters, because they just compute a fixed function of the input chosen beforehand.

One disadvantage that pooling layers introduce is the loss of information about the exact position of the features in the input image. After some pooling layers, the network loses completely the ability to locate features in the input signal. Consequently, pooling layers are most frequently used in contexts where it is more important to determine whether a certain feature is present in the input than its exact location, such as classification tasks.

As we commented at the beginning, 1×1 convolutions can be seen as a special type of cross-channel pooling that does not used a function established beforehand, but the parameters of the function are learned during training.

During backpropagation, the only thing we need to do in the case of a max pooling layer is to route the gradient of the loss to the input that produced the maximum value during forward propagation. Thus, it is common to keep, during forward propagation, the index of the input generating the maximum value, so that the backward pass through a max pooling layer can be executed efficiently.

As convolutional layers, spatial pooling layers can be used with 3D data: instead of considering 2D rectangle neighborhoods of the input, we have 3 dimensional neighborhoods for which the summary statistics are reported. For example, if $F = 2$ and $S = 2$ along the three dimensions, a max pooling layer will split the input into not overlapping $2 \times 2 \times 2$ cubes and return the maximum value within each cube.

7.4. Transposed Convolutions

In many network architectures, it is necessary to transform a tensor with the shape of the output of some convolution operation back to the shape of its input. This transformation is known as *up-sampling* and here is where *transposed convolutions* come into play. A transposed convolution is no more than a normal convolution where zeros are added at the borders and between the values of the input as required.

The following result is given in [38], Relationship 13:

Proposition 7.1. *Let us consider the output of a 2D convolution over an input of size $i \times i$ with kernel of size $k \times k$, stride s in both dimensions and p padding at each border, such that $i + 2p - k$ is a multiple of s. Recall that the size of the output is given by*

$$o = \frac{i - k + 2p}{s} + 1 \tag{7.1}$$

for each dimension. Then, it has an associated transposed convolution applied over an stretched input of size $\tilde{i}$, obtained by adding $s - 1$ zeros between each value of the input, with a $k \times k$ kernel, stride of 1 and zero padding of $\tilde{p} = k - p - 1$, whose output has size

$$\tilde{o} = s(o - 1) + k - 2p \tag{7.2}$$

for both spatial dimensions.

Observation 7.1. Note that the expression (7.2) is obtained by solving (7.1) for i.

Let us now see a couple of examples from [38]. First, we consider a simple example of a convolution without padding and unitary stride. Assume we have the output of a convolution on a 4×4 input ($i = 4$) with a 3×3 kernel ($k = 3$) using unitary stride ($s = 1$) and no padding ($p = 0$), which is of size 2×2 ($o = 2$) as shown in Figure 7.10.

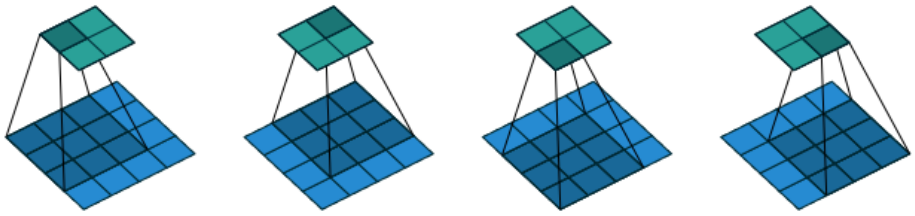


Figure 7.10.: Convolution of a 3×3 kernel (gray square) over a 4×4 input (blue square) with unit stride and no padding, which results in a 2×2 output (green square).

The transposed convolution associated to this convolution will allow us to recover the 4×4 shape of the input from the 2×2 output. It is equivalent to convolving a 3×3 kernel over the 2×2 tensor ($\tilde{i} = 2$) padded with $\tilde{p} = k - p - 1 = 3 - 1 = 2$ zeros at each border using stride 1:

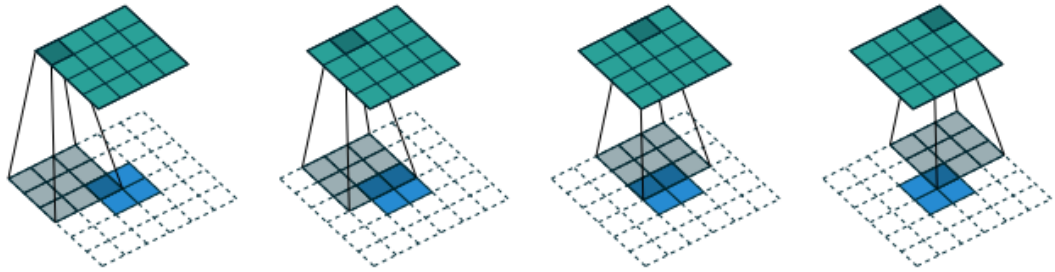


Figure 7.11.: The transposed of the convolution in Figure 7.10 is another convolution of a 3×3 kernel (gray square) over the 2×2 input (blue square) padded with 2×2 borders of zeros (dashed borders) with unit stride, resulting in a 4×4 output (green square).

A bit more complicated case is when the stride of the original convolution is not unitary. Let us consider the convolution depicted in Figure 7.12. Then, its transposed convolution, shown in Figure 7.13, is described by $\tilde{i} = 5$, $\tilde{p} = k - p - 1 = 3 - 1 - 1 = 1$, $k = 3$, $s = 1$ and its output has size $\tilde{o} = 5$.

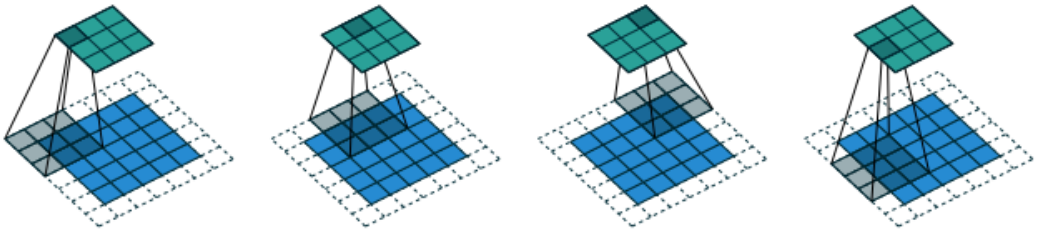


Figure 7.12.: Convolving a 3×3 ($k = 3$) kernel (gray square) over a 5×5 input tensor ($i = 5$) (blue square) padded with a 1×1 border ($p = 1$) (dashed border) using stride 2 ($s = 2$) results in a 3×3 output (green square).

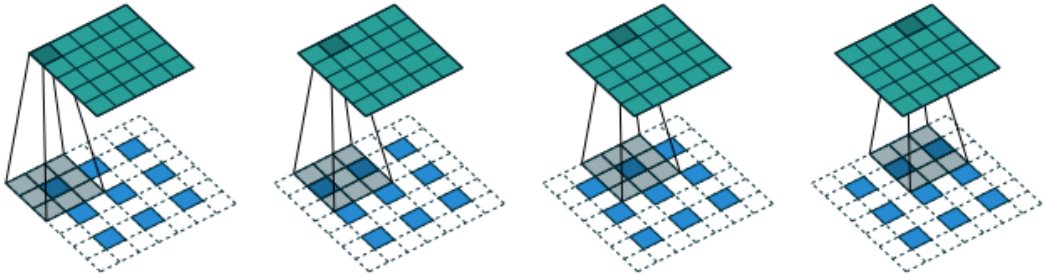


Figure 7.13.: The transposed of the convolution shown in Figure 7.12 is equivalent to convolving a 3×3 kernel (gray square) over a 5×5 input, obtained by inserting $s - 1 = 2 - 1 = 1$ zeros (dashed squares) between each two values of the 3×3 input (blue squares), using unit stride and padding of 1 at each border (dashed border), which outputs a tensor of size 5×5 (green square).

By using several transposed convolutional layers, we can map a low dimensional feature map to a high dimensional space.

Note that with a transpose convolution the original high dimensional input image cannot be obtained, but just an image with the same dimensions.

It goes without saying that this type of convolution and the result presented above can easily be generalized to N-dimensional data and non-square cases.

7.4.1. Checkerboard artifacts

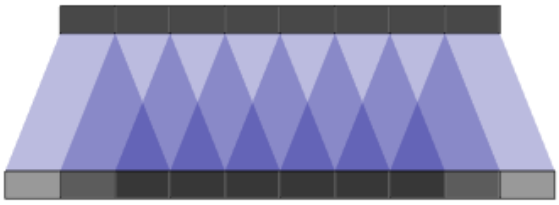
It has been observed that, when using transposed convolutions, *checkerboard artifacts* appear in the output image. An explanation of this problem together with some advice on how to avoid it can be found in [39]. We will not go into much detail on this issue, but let us just put into context.

An *artifact* is any anomaly that appears in an image and that is not present in the original image. With transposed convolutions, artifacts with a checkerboard like pattern have been observed in the resulting image. One of the main reasons for this pattern to appear is the use of transposed convolutions with *uneven overlap*, that is, with a kernel size that is not a

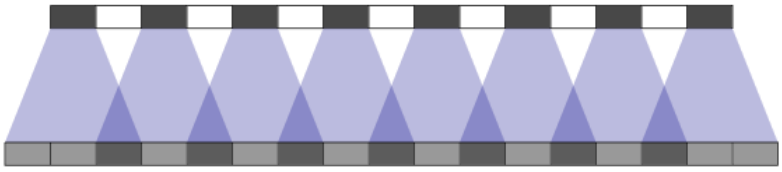
multiple of the stride used in the original convolution. In this case, some values of the output "receive" information from a higher number of pixels of the input than others, or, in other words, some values of the output are the result of convolving the kernel with a region of the input that has a higher number of zeros. For example, in Figure 7.13, we can see that the first value of the output at the corner, receives information from only one value of the input and 8 zeros, while the second one receives information from two values. Some values in the second row of the input receive information from 4 values of the input and only 5 zeros. This uneven overlapping is what causes the checkerboard artifacts.

To avoid the uneven overlap, the kernel size should be chosen to be divisible by the stride. However, artifacts may appear even if we do not use convolutions with uneven overlap, since the network may learn parameters for the kernel that produce checkerboard artifacts.

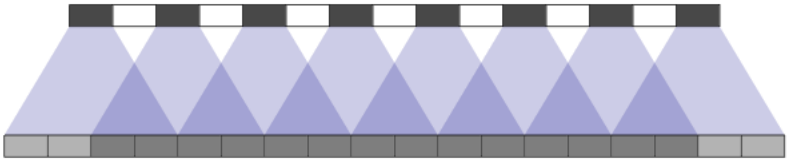
In [39] it is recommended to perform up-sampling separately, for example by resizing the image first using interpolation and then apply the convolution layer.



(a) 1D transposed convolution with $k = 3$ and $s = 1$.



(b) 1D transposed convolution with $k = 3$ and $s = 2$.



(c) 1D transposed convolution with $k = 4$ and $s = 2$.

Figure 7.14.: One dimensional transposed convolutions with different kernel and stride sizes. The tensor on the top is the input, and the tensor on the bottom is the output after applying the transposed convolution. Note that when there is no uneven overlap, only the values at the borders of the output receive information from a different number of input values than the others.

7.5. Normalization layer: Instance vs Batch Normalization

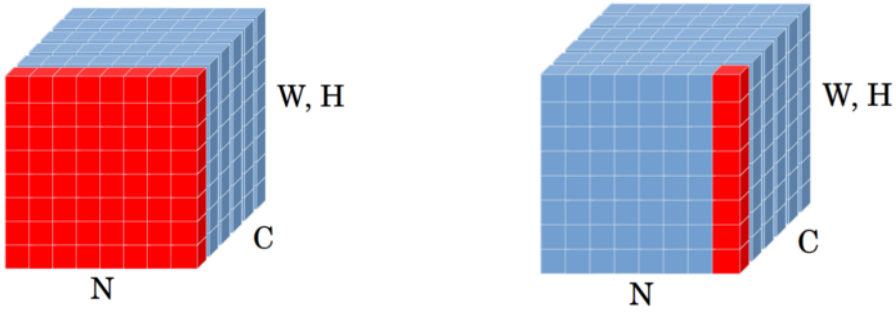
In the previous chapter we explained the benefits of using batch normalization in deep feed-forward neural networks. In particular, in convolutional neural networks the same benefits apply. Therefore, it is common to normalize the inputs to a convolutional layer by using different techniques. It is the objective of the so-called *normalization layers*. We will describe here two types of normalization layers with the help of [40].

Let $x \in \mathbb{R}^{N \times C \times W \times H}$ be a four dimensional tensor containing a mini-batch of N examples, each of them with C features maps (channels) and $W \times H$ spatial dimensions. Let x_{nijk} be the $nijk$-th element of this tensor. What a **batch normalization layer** does is to normalize each individual feature map by using the mean and variance calculated across all the examples of the mini-batch and both spatial dimensions:

$$\mu_i = \frac{1}{HWN} \sum_{n=1}^{N} \sum_{j=1}^{W} \sum_{k=1}^{H} x_{nijk}, \quad \sigma_i^2 = \frac{1}{HWN} \sum_{n=1}^{N} \sum_{j=1}^{W} \sum_{k=1}^{H} (x_{nijk} - \mu_i)^2,$$
$$\hat{x}_{nijk} = \frac{x_{nijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}},$$

where $0 < \epsilon \ll 1$ is used to avoid zero division. For its part, an **instance normalization layer** [41] computes the mean an variance over the spatial dimensions of each feature map and each single training example and uses them to normalize that example:

$$\mu_{ni} = \frac{1}{HW} \sum_{j=1}^{W} \sum_{k=1}^{H} x_{nijk}, \quad \sigma_{ni}^2 = \frac{1}{HW} \sum_{j=1}^{W} \sum_{k=1}^{H} (x_{nijk} - \mu_{ni})^2,$$
$$\hat{x}_{nijk} = \frac{x_{nijk} - \mu_{ni}}{\sqrt{\sigma_{ni}^2 + \epsilon}}.$$



(a) Batch normalization.

(b) Instance normalizacion.

Figure 7.15.: Comparison of the dimensions affected by batch and instance normalization.

In the case of 3D data of size $W \times H \times D$, the mean and variance are calculated over the three spatial dimensions of the data, instead of just the two dimensions W and H.

Both types of normalization layers include learnable parameters γ and β, used to scale and shift, respectively, the normalized values, as explain in subsection 6.4.2.

Since instance normalization does not depend on the examples in the mini-batch, as opposed to batch normalization, it can be applied at test time the same way as it is used during training.

When the mini-batch size is small, the mean and variance over the examples in the mini-batch may become too noisy and introduce error to the training process. Therefore, in this case instance normalization normally performs better than batch normalization.

Instance normalization is also known as *contrast normalization*, because it normalizes the contrast across the spatial elements of a feature map.

7.6. Residual Blocks

With the use of very deep convolutional neural networks a problem was observed: the *degradation problem*. As more layers are added to a network, its performance accuracy gets saturated (as it is expected since a sufficiently deep neural network should be able to learn every detail of the training data) but then it degrades quickly. This degradation is not due to overfitting, because not only the error at test time decreases, but also the error over the training data.

Intuitively, if a shallow network is able to achieve a certain performance over the training set, the same network with more layers should present at least the same training accuracy. In other words, if a network with a certain number of layers can implement a set of functions $\mathcal{H}$, when adding more layers (more parameters) it should be able to implement another set of functions $\mathcal{H}'$ such that $\mathcal{H} \subseteq \mathcal{H}'$, because the original network could be recovered by replacing the newly added layers with an identity mapping. However, in practice we observe that this does not happen, probably because it is very difficult for the neurons in the layers to learn the identity function, specially when they use non-linearities and other complex transformations.

In order to alleviate the degradation problem, *residual learning* was proposed in 2016 [42]. Let $h(x)$ be an underlying function to be learned by a few stacked layers, where x is the input to the first of these layers. In the residual learning framework, this set of layers is forced to learn a *residual function* $f(x) := h(x) - x$ (hence the name), instead of $h(x)$. Therefore, the original function becomes $h(x) = f(x) + x$. When the identity mapping is the desired underlying function, with this residual learning reformulation it is easier to learn: the optimization algorithm will simply push all the weights and biases associated with the stacked layers towards zero, so that $f(x) = 0$ and $h(x) = x$.

Residual learning is implemented by feedforward neural networks with *shortcut connections* (also known as *residual connections* or *skip connections*), that is, connections that skip one or more layers, which perform the identity function. Then, their outputs are added to the outputs of the stacked layers. In this way, every few stacked layers, a *residual block* as shown in Figure 7.16 is considered. The output of this block is given by

$$y = \theta(f(x, \{W_t\}) + x),$$

where x is the input, θ is the ReLU activation function (other activation functions are possible), $f(x, \{W_t\})$ is the residual mapping to be learned and W_t is the weight matrix associated to every layer t in the residual block. For example, in Figure 7.16 there are two layers, so $f(x, \{W_1, W_2\}) = W_2\theta(W_1 x)$. The operation $f + x$ is computed with a shortcut connection

and an element-wise addition. Note that the activation function of the second layer is applied after the addition, because otherwise only positive (or zero) residual values could be added to the identity mapping, which would reduce the expressiveness of the residual block.

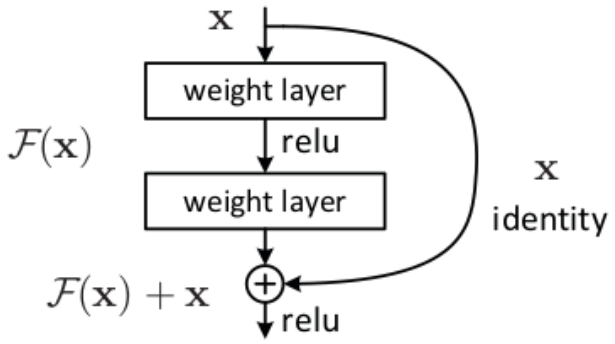


Figure 7.16.: Example of a residual block with two layers. Here $\mathcal{F}$ is the residual function.

The shortcut connections do not introduce any extra parameters to be learned and do not increase the computational complexity, except for the insignificant element-wise addition. Therefore, a network with residual blocks can be trained end-to-end using gradient descent and backpropagation.

In order to be able to compute the addition of f and x they must have the same dimension. However, it is not always the case. We can apply a linear projection W_s to the identity mapping to match the dimension of the residual value, so that

$$y = \theta(f(x, \{W_t\}) + W_s x).$$

Although we have considered a residual block with two layers, the number of stacked layers can vary. Nevertheless, with only one layer, the output of the residual block before applying the activation function is linear $y = W_1 x + x$, which is equivalent to a single linear layer without the shortcut connection, so it does not make sense to use a residual block with only one layer.

By stacking residual blocks together, very deep residual networks can be built, which showed a substantial increase in the performance accuracy with respect to deep neural networks without residual connections. Indeed, if the additional layers can easily learn the identity mapping, they will not hurt the network performance, because they would be skipped (that is, the identity function would be applied) if they were not useful, and, otherwise, the weights associated to them would be adjusted to be non-zero.

Residual learning can also be applied to convolutional neural networks. In this case, the residual function f is implemented by multiple convolutional layers and the element-wise addition is computed channel by channel. A convolution may be added to the shortcut connection to make the shape of the output of the stacked convolutional layers match the shape of the input. For example, if the number of channels do not coincide, a 1×1 convolution could be introduced to transform the depth of the input into the desired value for the addition.

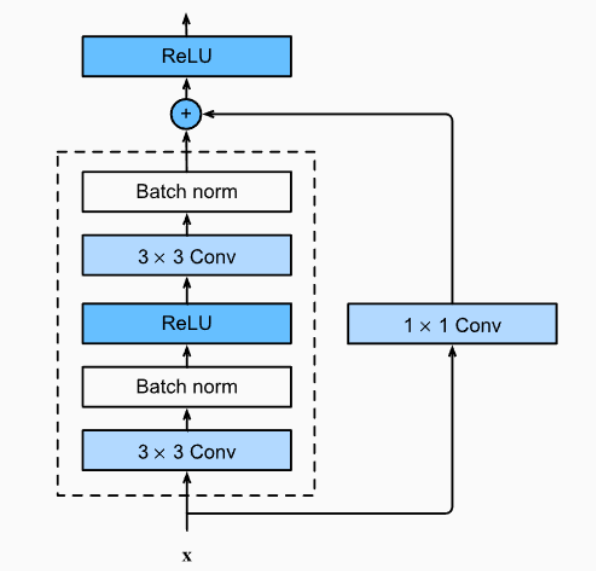
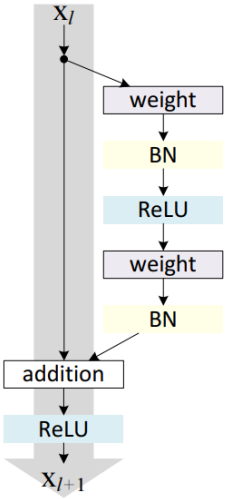
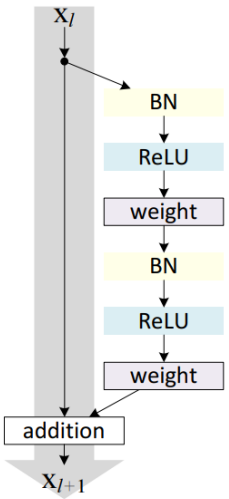


Figure 7.17.: Residual block formed by two convolutional layers, with filters of size 3×3, with a 1×1 convolution in the shortcut connection. Image from [43].

There exist different possible combinations of batch normalization, ReLU activations and convolutional layers in a residual block. He et al. [44] carried out a study on how different orders affect the network performance and came up to the conclusion that a setting that they call *full pre-activation*, shown in Figure 7.18b, offers the best results.



(a) Original setting of a Residual Block



(b) Full pre-activation setting

Figure 7.18.: Comparison between the original order of batch normalization (BN), ReLU activations and weight layers in a residual block and the new order proposed by He et al. x_l refers to the input to the l-th residual block and x_{l+1} is the output.

Part IV.

Segmentation of prostate cancer lesions in Magnetic Resonance Images

8. Semantic Image Segmentation

In this chapter we will focus on semantic image segmentation. *"What is in this image, and where in the image is it located?"* - This is the question that the semantic segmentation task tries to solve. We begin the chapter by describing the task at hand following the references [45], [46] and [47]. Once we know the problem we face, the most common deep learning architecture used for solving this task, namely *Unet*, is explained, along with some of its variants that will be used in our work. Finally, the two most popular loss functions used for training the segmentation models are introduced: the *pixel-wise cross-entropy loss* and the *soft Dice loss*.

8.1. Description of the task

Image segmentation is a computer vision task that aims at dividing an image into multiple regions or segments, hence its name. In other words, every pixel in the image is labeled with a certain class according to what it shows.

We can distinguish between two main types of image segmentation: *semantic segmentation* and *instance segmentation*. In semantic image segmentation, only the class of each pixel is relevant, i.e., all pixels belonging to the same category are labeled using only one class, while in instance segmentation separate objects of the same category are marked with different class labels.

(a) Original image (b) Semantic Segmentation (c) Instance Segmentation

Figure 8.1.: Comparison between semantic and instance segmentation. Source [48].

In this way, the goal of the semantic segmentation task is to take an image of size $W \times H \times C$ (where C is the number of channels) and generate a segmentation map of size $W \times H \times 1$, where each pixel contains a class label $0, 1, \ldots, n-1$, being n the total number of different classes. The class labels may be one-hot encoded, producing a binary output channel for each possible class. This n-channel segmentation map can then be converted back to a single segmentation map by taking the *argmax* of each depth-wise pixel vector.

Like every supervised machine learning task, the semantic segmentation problem requires a ground truth segmentation map associated with each input image used for training. Every single channel of this ground truth map (in case it is one-hot encoded) is often referred to as a *mask*, since by overlapping it onto the input image we can see all the regions belonging to a specific class.

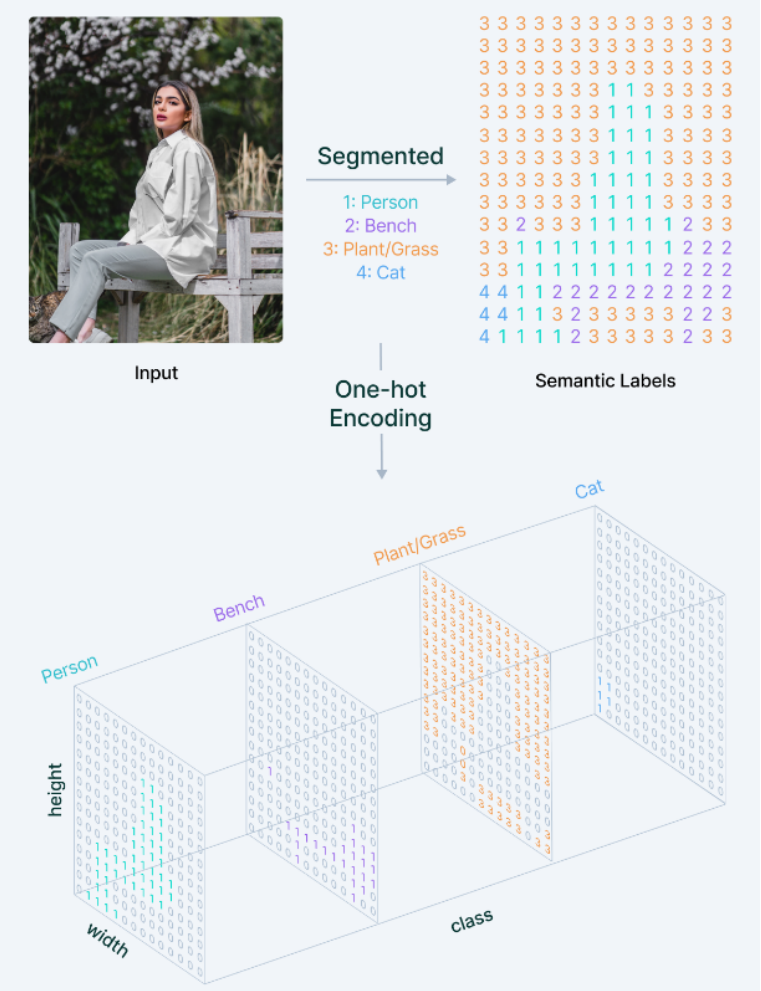


Figure 8.2.: Example of semantic segmentation of an input image. The resolution of the segmentation map has been lowered compared to the original image's resolution for visual clarity, but in reality both resolutions must match. Source [47].

8.2. The Unet Architecture

The basic deep learning architecture used for image segmentation consists of an *encoder* and a *decoder*:

- The **encoder** downsamples the spatial resolution of the input while extracting different features from the image used to efficiently distinguish between classes. It presents the typical architecture of a convolutional neural network, containing several convolutional layers followed by non-linear activation functions and pooling layers. The initial layers tend to learn low-level features (such as colors or edges) and the later layers learn high-level features (such as objects or complex curves and patterns). As more layers are added, the spatial size of the resulting feature maps decreases, thus, to maintain the expressiveness, the number of channels needs to be increased. As a result, the output of the encoder is a deep low-resolution tensor containing high-level information.
- For its part, the **decoder** produces the full-resolution segmentation map by successively upsampling the low-resolution tensor output by the encoder. To achieve that, several convolutional layers are added coupled with upsampling layers, which are responsible for increasing the spatial size of the input (see Figure 8.3). The number of channels is decreased as the spatial resolution increases, since we are getting back to the low-level information.

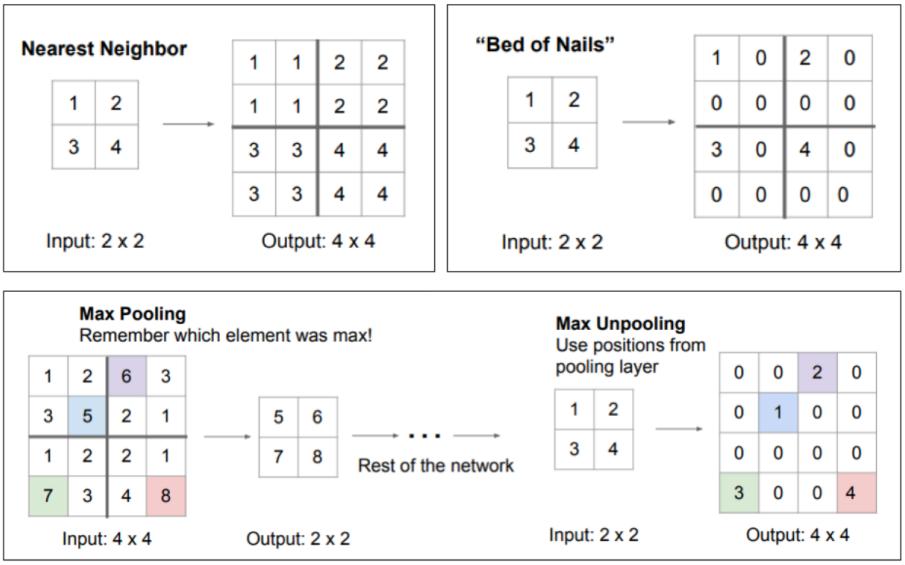


Figure 8.3.: Different *unpooling* techniques for upsampling the resolution of a feature map. Unpooling can be seen as the opposite of pooling: while a pooling operation summarizes a local region of the input into a single value, an unpooling operating distributes a single value of the input into a higher resolution region of the output. Nevertheless, transposed convolutions are the most used upsampling technique, as they allow to learn the parameters of the filters. Source [22], Lecture 9.

The most popular semantic segmentation network used nowadays was proposed by Ronnenberger et al. [3] in 2015, and it was called *Unet*, due to its "*U*" shape. As stated in the

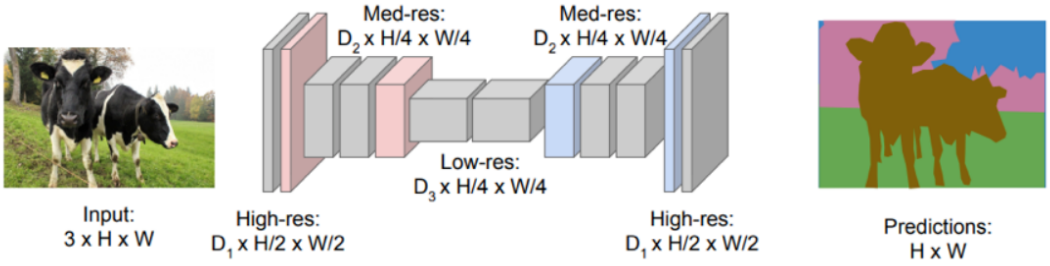


Figure 8.4.: Convolutional encoder-decoder architecture. The red volumes represent downsampling layers while the blue ones are upsampling layers. The gray volumes are simply convolutional layers. Image from [22], Lecture 9.

original paper, it *consists of a contracting path to capture context and a symmetric expanding path that enables precise localization*. The symmetry is achieved by using the same number of filters in the expanding path (i.e. the decoder) than in the contracting path (i.e. the encoder), and it is what provides the network with its U-shape (together with the skip connections that we will explain soon). No fully-connected layers are included in the network and no zero padding is added to the feature maps before applying the convolutions, so the size of the output of a convolutional layer in the network is reduced compared to the size of its input.

The Unet architecture uses *skip connections*, which allow the decoder to access the feature maps generated by the encoder layers. By simply stacking the encoder and the decoder layers, a loss of low-level information has been observed due to the downsampling operations and the resulting network struggles to produce fine-grained segmentation maps. Thanks to these skip connections, each layer in the contracting path sends its resulting feature maps to the corresponding layer of the expanding path, what provides the necessary details to generate finer and more accurate segmentations. In other words, high resolution feature maps from the contracting path are concatenated to the corresponding upsampled feature maps of the expanding path and a successive convolutional layer then learns a more accurate upsampled output based on the information of this concatenation.

The Unet architecture is shown in Figure 8.5. As we can see, the encoder is formed by a set of two 3×3 unpadded convolutions, each of them followed by a ReLU activation function and a 2×2 max pooling layer with stride of 2. The number of filters is doubled after each pooling layer. In the decoder an upsampling operation is applied to each feature map followed by a 2×2 convolution (what the authors called *up-convolution*) that halves the number of channels. After the up-convolution, the resulting feature map is concatenated with the correspondingly cropped feature map from the encoder, and two 3×3 convolutions are applied to the result, each followed by a ReLU function. It is necessary to crop the feature maps from the contracting path because the pixels at the borders are lost after every convolution. The final layer of the network uses a 1×1 convolution to map the 64 feature channels to the desired number of classification classes.

Observation 8.1.

- The input spatial size must be selected in such a way that the max pooling operations in the contracting path are applied to an input of even spatial dimensions.
- The output segmentation map is smaller than the input original image by a constant border width because of the unpadded convolutions.

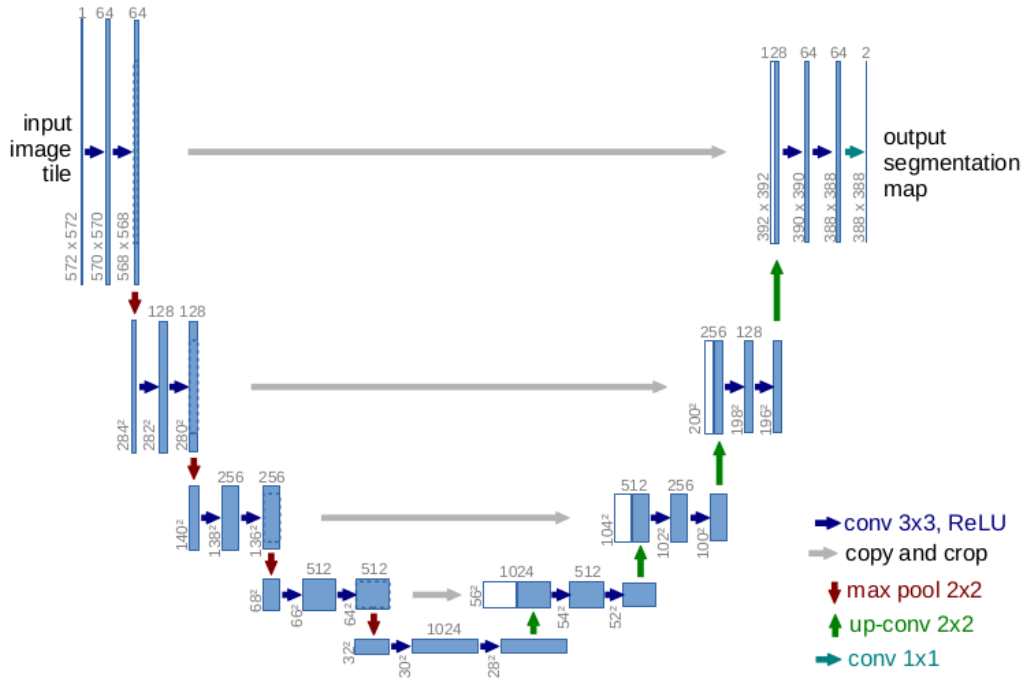


Figure 8.5.: Unet architecture as proposed in the original paper [3]. The arrows represent the different operations, while the blue boxes denote multi-channel feature maps. The number of channels is indicated on the top of the box and the spatial size is shown at the lower left edge of the box. The white boxes refer to copied feature maps.

8.2.1. Modifications of Unet

8.2.1.1. 3D Unet

Since volumetric data is abundant in the biomedical field, a 3D extension of the original 2D Unet was proposed later on, concretely one year later [49]. As opposed to the original Unet, which is entirely a 2D architecture, the 3D Unet receives 3D inputs and processes them by using 3D operations, that is, with 3D convolutions, 3D pooling and 3D up-convolutions. Additionally, it avoids bottlenecks in the network architecture by doubling the number of feature channels before each pooling layer in the contracting path and by halving them after each upconvolution in the expansive path. It also includes a batch normalization layer before each non-linearity.

The network architecture is illustrated in Figure 8.6. Each step in the encoder consists of two $3 \times 3 \times 3$ convolutions, each followed by a batch normalization and a ReLU function, and then a $2 \times 2 \times 2$ max pooling layer with a stride of 2 in each dimension. Each step in the decoder consists of a $2 \times 2 \times 2$ upconvolution with a stride of 2 in each dimension and two $3 \times 3 \times 3$ convolutions, each of them followed by a batch normalization and a ReLU. The last layer consists of a $1 \times 1 \times 1$ convolution which reduces the number of feature channels to the appropriate number of classes.

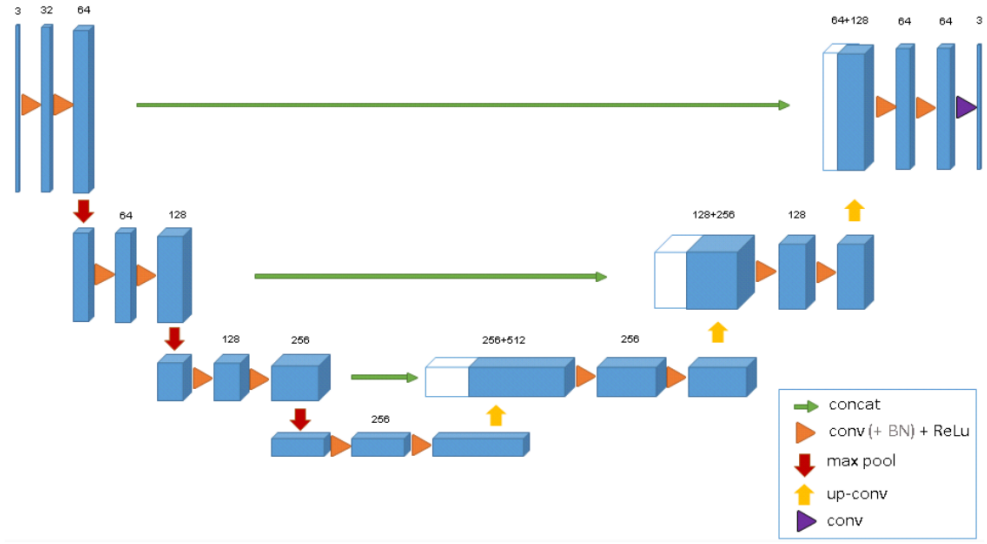


Figure 8.6.: 3D Unet architecture. The arrows denote the different operations and the blue boxes are feature maps with the number of channels denoted on top of each box. The white boxes represent the corresponding cropped feature maps from the contracting path.

8.2.1.2. Residual Unet

Zhang et al. [50] showed in 2018 that the performance of Unet improves when replacing the plain convolutional blocks with residual blocks. The use of residual blocks not only makes the training of the network easier, but also avoids, together with the skip connections between the encoder and decoder, the degradation problem and the loss of information.

Here we will present the U-ResNet architecture used in [12], which is adapted to volumetric data, and it is, in turn, based on the 2D architecture employed in [51]. This U-ResNet is the network that we will use later in our work.

The architecture of the U-ResNet is depicted in Figure 8.7. As the original Unet, it is formed by an encoder, a decoder and a transitional block connecting both parts.

The encoder is built with residual units consisting of four convolutional layers with a kernel size of 3 (in each dimension), each followed by a batch normalization layer, a dropout layer and a Parametric Rectified Linear Unit (PReLU). The number of convolutional layers in a residual block may vary in other implementations. A shortcut connection joins the input and output of a residual unit, and it includes a convolution to change the input dimensions in case they do not match the output dimensions. Instead of adding pooling layers for downsampling, a stride of 2 in each dimension is applied to the first convolutional layer to half the spatial dimensions of the input feature maps, while the subsequent convolutional layers in a residual block use a stride of 1. Every convolution in the transitional block is applied with a stride of 1, so that no downsampling occurs here.

The upsampling blocks in the decoder are composed of a transposed convolution with a kernel size of 3 (in each dimension) and a stride of 2 in each dimension, that doubles the spatial size of the input, and a convolutional layer with a kernel size of 3 (in each dimension)

and stride of 1. Each layer in an upsampling block is followed by batch normalization, dropout and a PReLU activation function, except for the last layer in the final upsampling block, which simply applies a convolution.

Padding values for the convolutions operations are set to ensure the output size is the same as the input size in case the stride is 1, and the output size is half (respectively double) of the input size in case the stride is 2 in a downsampling block (respectively in an upsampling block). Therefore, the necessary cropping of feature maps in the original Unet for the concatenation is not needed in this network.

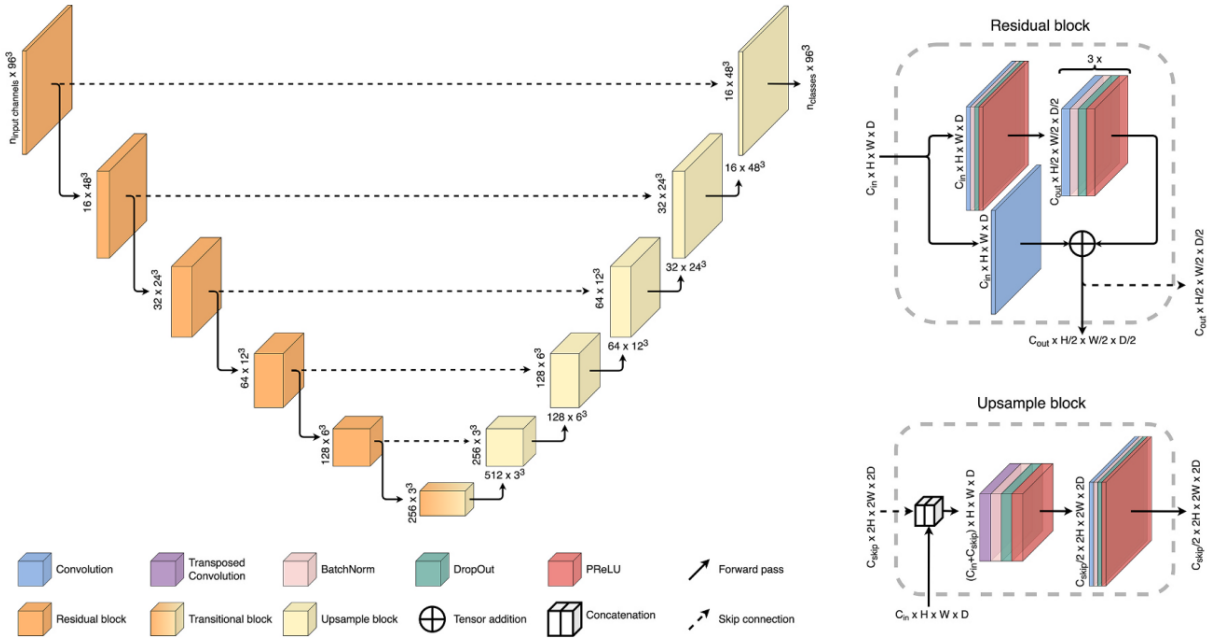


Figure 8.7.: U-Resnet architecture as described in [12]. In this case, the input volume has a shape of $96 \times 96 \times 96$. The dimensions of the output feature maps are indicated after the corresponding arrow. All convolutions have a kernel of size $3 \times 3 \times 3$. The first convolution in each residual block and upsampling block uses a stride of 2, while all the other convolutions are applied with a stride of 1. The exception is the transitional block, where all convolutions use a stride of 1. The final upsampling block is different to the others in that the last convolution is not followed by any other operation and the transposed convolution does not half the number of channels, but it maps them to the desired number of classes.

Observation 8.2.

1. The input spatial dimensions must be divisible by 2 as many times as the downsampling is applied in the network. That is, they must be multiples of powers of 2. For instance, in the architecture shown in Figure 8.7, the input dimensions must be a multiple of $2^5 = 32$.

2. Parametric Rectified Linear Units are used instead of Rectified Linear Units to allow the network to learn the optimal activation, which improves the segmentation quality.

3. The use of convolutions with a stride of 2 for downsampling and transposed convolutions with a stride of 2 for upsampling, instead of pooling and unpooling layers, allows the network to learn the most appropriate downsample and upsample operations and, at the same time, reduces the number of layers in the network.

4. After the transposed convolution in an upsampling block, only one convolutional layer is added. This fact dismisses the use of residual connections in the upsampling blocks, since, as we already explained before, a residual block with only one convolutional layer is equivalent to a plain convolutional layer.

5. The concatenation of feature maps from the decoding path and the corresponding encoding layer takes place before the upsampling operation, since they both have the same spatial dimensions and an upsampling is not needed before the concatenation, as it is in the original Unet. This is due to the absence of downsampling in the transitional block.

8.3. Loss functions

Here we will explain the two most common loss functions used for the task of semantic image segmentation, although there exist many others, such as focal loss, Tversky loss, contrastive loss, Jaccard loss, etc.

After the final layer of the Unet architecture, a *pixel*[1]*-wise softmax activation* is applied to the n resulting feature maps, being n the number of available classes, so that n new channels of the same size are obtained where each depth-wise pixel vector contains the probability that it belongs to each class $c \in \{0, 1, \cdots, n-1\}$. We will write $y_{i,c}^{pred}$ to refer to the predicted probability of pixel i of belonging to class c, and $y_{i,c}^{true}$ will be 1 in case the ith pixel belongs to class c and 0 otherwise. With this notation, we can now define the most typical loss functions.

8.3.1. Cross-Entropy Loss

Cross-entropy is a measure of the difference (or similarity) between two probability distributions and it can be used to define a loss function for the segmentation task. With this loss, the class predictions are evaluated individually for each depth-wise pixel vector and then an average is computed over all pixels. The pixel-wise loss is calculated as the *log* loss, summed over all the available classification classes. That is, the total loss is calculated as follows:

$$CE(y^{pred}, y^{true}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=0}^{n-1} y_{i,c}^{true} \log(y_{i,c}^{pred}),$$

where we have assumed that there are N pixels in each feature map.

The pixel-wise cross-entropy loss is differentiable, what makes it feasible for the optimization. However, it is not suitable in case we are dealing with a class imbalance, that is, the classes are not equally represented in the image, what is a common problem in medical images where the background class is dominant. This is because pixel-wise cross-entropy treats all pixels in the image equally, so the training could be dominated by the most frequent class.

[1] We talk about pixels here, but in the case of 3D images they are voxels.

A possible solution is to weight each loss term by a class-specific weight w_c, so that underrepresented classes are assigned a weight bigger than 1. In this way, the loss function is biased towards the underrepresented classes to compensate for the class imbalance. The cross-entropy loss is then of the form:

$$CE(w, y^{pred}, y^{true}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=0}^{n-1} w_c y_{i,c}^{true} \log(y_{i,c}^{pred}).$$

8.3.2. Dice Loss

Another important loss function is the Dice loss, which is based on the *Sørensen–Dice coefficient*. This coefficient is a measure of overlap between two samples and ranges from 0 to 1, taking the value of 1 when the overlapping is complete. Given to sets A and B, the dice coefficient is defined as

$$DSC = \frac{2|A \cap B|}{|A| + |B|},$$

where $|A|$ and $|B|$ represent the number of elements in the set A and B respectively, and $|A \cap B|$ is the number of elements that both sets have in common. The number 2 in the numerator is used to compensate for the fact that the denominator counts the common elements between the two sets twice.

In order to calculate the Dice coefficient for two segmentation maps A and B (the predicted one and the ground truth), an element-wise multiplication between both maps is carried out followed by an addition of the elements of the resulting matrix. Similarly, the cardinalities $|A|$ and $|B|$ are computed as the sum of the elements of the respective segmentation map, or as the square of this sum.

In this way, we would like to maximize the Dice coefficient, or, equivalenty, to minimize $1 - DSC$. This is the loss function known as the *soft Dice loss*, because no thresholding is applied to convert the probabilities into a binary mask, but these probabilities are used directly to compute the loss. Therefore, the soft dice loss is calculated for each class $c \in \{0, 1, \ldots, n-1\}$ separately as follows:

$$D_c(y^{pred}, y^{true}) = 1 - \frac{2\sum_{i=1}^{N} y_{i,c}^{true} y_{i,c}^{pred}}{\sum_{i=1}^{N} y_{i,c}^{true} + \sum_{i=1}^{N} y_{i,c}^{pred}}$$

Then, it is averaged over all classes to obtain the final loss:

$$\frac{1}{n}\sum_{c=0}^{n-1} D_c(y^{pred}, y^{true}) = \frac{1}{n}\sum_{c=0}^{n-1}\left(1 - \frac{2\sum_{i=1}^{N} y_{i,c}^{true} y_{i,c}^{pred}}{\sum_{i=1}^{N} y_{i,c}^{true} + \sum_{i=1}^{N} y_{i,c}^{pred}}\right).$$

Just as the cross-entropy loss, the Dice loss is differentiable. Additionally, it is not affected by the class imbalance problem, but it shows excellent results when dealing with unbalanced classes.

9. Description of the problem

Prostate Cancer (PCa) is the most frequently diagnosed type of cancer in males in most countries around the world, including Spain, and the fifth in the number of deaths (2020). Magnetic Resonance Imaging (MRI) has shown excellent results in early diagnosis, monitoring and treatment planning for prostate cancer. In fact, international guidelines recommend performing a mpMRI (multiparametric magnetic resonance imaging) of the prostate to identify clinically significant PCa lesions, prior to any biopsy. However, interpreting these images is a complex and time-consuming task even for experienced radiologists. Additionally, it is subject to the observer's experience level, as well as human mistakes, and, as a result, it suffers from inter-observer and intra-observer variability. Computer-aided diagnosis (CAD) systems that use machine learning and deep learning approaches have been shown to improve diagnostic accuracy and reduce inter-observer variability. In particular, the automatic segmentation of suspicious PCa lesions can provide a significant reference to the radiologists, reducing the time needed for the interpretation, the diagnostic errors and the dependence on expert knowledge.

To improve the generalization ability of our models to images obtained with different MRI scanners and annotated by different radiologists, several public databases have been used: ProstateX, I2CVB, Prostate158, PROMISE12, NCI-ISBI 2013 and Medical Segmentation Decathlon.

In this chapter we briefly analyze the prostate anatomy and we describe some characteristics of the prostate cancer along with a system to measure its aggressiveness (the *Gleason grading system*) and the most frequent PCa screening techniques. We will also explain the different MRI modalities commonly used for the diagnosis of PCa. To this end, we make use of [52] and [53]. Additionally, we offer a motivation for our work. Details of different public databases are described in this chapter as well. Finally, we present some previous projects that used deep learning models to segment prostate lesions.

9.1. Prostate anatomy

The prostate is an exocrine gland of the male reproductive system, which is about the size and shape of a walnut. It is located in the pelvis, just in front of the rectum and below the bladder, with the urethra passing through it. Its primary role is to generate a fluid that enriches and protects sperm. In adults, its weight ranges from 7 to 16 grams, with an average weight of 11 grams. The prostate increases at puberty, to reach its normal size, and later after the age of 60, generating benign prostatic hyperplasia (BPH).

A 3D zonal classification of the prostate gland was proposed by McNeal [54], and it is the most widely used anatomical description of the prostate since then. According to this classification, the prostate is divided into four heterogeneous regions, which are depicted in Figure 9.2:

1. The **anterior fibromuscular stroma (AFS)**. It has no glandular tissue and it is composed only of muscle and fibrous tissue.
2. The **transitional zone (TZ)**, consisting of 5% of glandular tissue.
3. The **central zone (CZ)** constitutes the 20% of the prostate gland.

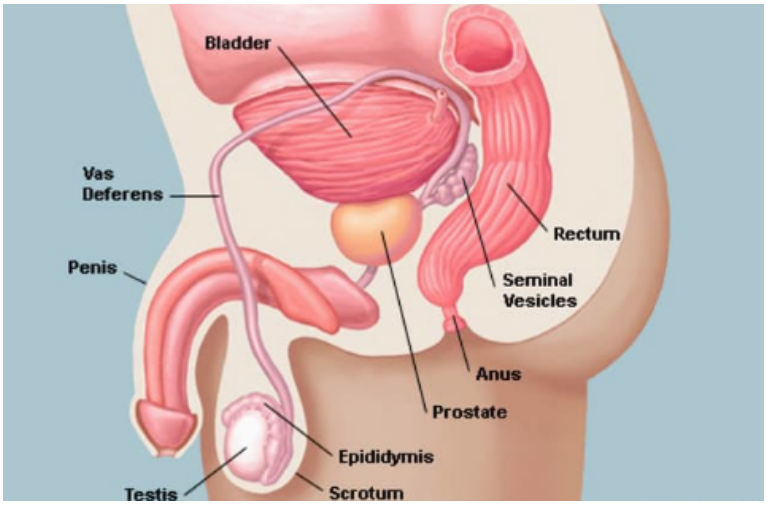
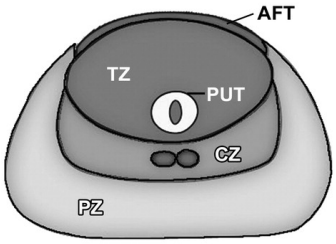


Figure 9.1.: Anatomy scheme of the male reproductive system. Source https://www.webmd.com/men/picture-of-the-prostate.

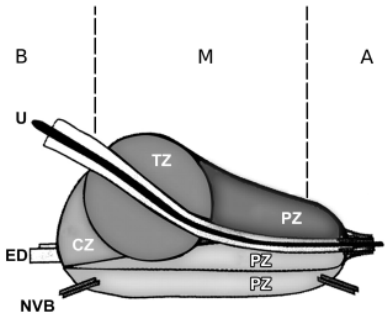
4. The **peripheral zone (PZ)** represents 70% to 80% of the glandular tissue.

In medical imaging, the CZ and TZ tissues are difficult to differentiate, so they are usually combined into a single region and named **central gland (CG)**.

Additionally, the prostate is divided into three longitudinal regions shown in Figure 9.2b: the base, median gland and apex.



(a) Anatomy of the prostate in transverse plane.



(b) Anatomy of the prostate in sagittal plane.

Figure 9.2.: Prostate anatomy in different planes. *AFT*: anterior fibromuscular tissue, *CZ*: central zone, *ED*: ejaculatory duct, *NVB*: neurovascular bundle, *PUT*: periurethral tissue, *PZ*: peripheral zone, *U*: urethra, *TZ*: transitional zone, *B*: base, *M*: median gland, *A*: apex. Source [53].

9.2. Prostatic Carcinoma

Prostatic Carcinoma or Prostatic Cancer (PCa) is a glandular cancer that appears in different zones of the prostate with different probabilities. In particular, between 70% and 80% of prostate cancers occur in PZ, around 10% to 20% develop in TZ, while only 5% of PCas appear in CZ, being the latter the most aggressive cancers and the most likely to affect other organs. As a result, segmentation of the different prostatic zones, especially PZ, is critical for the diagnosis of PCa.

To measure PCa aggressiveness the *Gleason Grade System* is used. Under this grading system, prostate cancer is given a grade called *Gleason score (GS)*, based on the target tissue's appearance under a microscope. In this regard, tissue samples from a prostate biopsy are examined under a microscope and a score between 1 and 5 is assigned to the main pattern of cell growth (that is, the region where the cancer is most significant) and to the second most predominant cell pattern, in such a way that cells that look more like healthy cells receive a lower grade and the other way around. These two scores are added to obtain a value between 2 and 10, although in practice the final score only ranges from 6 to 10. Therefore, lesions with a grade of 6 are similar to healthy tissue and they grow more slowly, while cancers with $GS \geq 8$ are different from healthy cells and they are more likely to grow and spread. According to the the International Society for Urological Pathology guidelines [55], cancers with a score $GS > 6$ are considered clinically significant (CS).

Prostate Cancer is the most frequently diagnosed cancer among men in most countries (including Europe and the USA), with more than 1.4 million new prostate cancer cases diagnosed in 2020 and $375,000$ associated deaths worldwide. It is the second most frequent cancer among men, after lung cancer, and the fifth most common cause of cancer death (2020) [56]. In 2020, incidence rates varied from 6.3 to 83.4 per $100,000$ men across regions, with the highest rates found in Northern and Western Europe, the Caribbean, Australia/New Zealand, Northern America, and Southern Africa. The survival rate of men with prostate cancer is relatively high, although human aging factors could aggravate the disease.

9.3. PCa Screening and Magnetic Resonance Imaging (MRI)

In order to examine and detect PCa abnormalities, *Prostate Specific Antigen (PSA)* measurement and *Digital Rectal Examination (DRE)* are performed. A higher than normal level of PSA may indicate an abnormality in the prostate gland. However, and elevated PSA does not necessarily means a cancer is present. In other circumstances, such as a benign prostatic hyperplasia, prostate infections, inflammation, a recent ejaculation or rectal examination, PSA levels may increase. Therefore, PSA measurement alone does not provide conclusive evidence for the existence of PCa. Both PSA and DCE have relatively low specificity and sensitivity.

When PSA or DCE indicates a possible cancer, a *transrectal ultrasound (TRUS)*-guided biopsy is carried out. It is an invasive procedure, which can result in serious risks for the patients. Furthermore, it has relatively low sensitivity and may underestimate lesions' aggressiveness.

With *Magnetic Resonance Imaging (MRI)*, prostate cancer can be reliably visualized, improving the localization of cancerous tissues and the subsequent targeted biopsy. MRI is an advanced imaging technique which produces multiplanar 3D data of high spatial resolution. Biopsies executed with the aid of MRI are more accurate and less harmful than the ones realized with the use of TRUS. MRI provides better soft-tissue contrast, better resolution and it is radiation free. All in all, besides its non-invasive nature, MRI provides accurate information

about the lesion's location, volume and level of aggressiveness, thus with great potential in early diagnosis, monitoring and treatment planning for prostate cancer. It reduces the number of unnecessary biopsies and over-diagnosis compared to TRUS. As a result, international guidelines recommend the use of MRI by radiologists for the clinical diagnosis of prostate cancer prior to any biopsy [55].

To standardize prostate MRI examinations and reports, the *Prostate Imaging Reporting and Data System (PI-RADS)*, which is now in version 2.1 [57], was developed by the International Prostate MRI Working Group. It provides a guideline for the interpretation of MRI images, according to which intraprostatic lesions are detected, delineated, their risk degree is graded and a corresponding treatment is suggested.

Four main MRI modalities are usually used for the detection of PCa, which in conjunction constitute the *multiparametric Magnetic Resonance Imaging (mpMRI)*:

1. **T2-weighted imaging (T2W).** It is the basic MR imaging protocol used for the diagnosis of PCa. It uses the differences in the T2 relaxation time[1] of tissues to generate a grayscale image of the scanned object. Due to the good resolution and contrast provided, it is employed for noninvasive PCa diagnosis, localization and staging.

   T2W imaging is usually carried out in axial, coronal and sagittal planes. It allows to visually differentiate between healthy and cancerous tissues thanks to differences in intensity and homogeneity. Concretely, in the peripheral zone, cancerous tissues present a low signal intensity, while in normal tissue PZ is hyperintense, and both PZ and CG are more homogeneous in malignant tissue compared to the surrounding normal tissues. This is because the presence of foci of congested glands (cancer) surrounded by less dense benign cells results in a region of the image of low intensity signal. It has been observed that PCa aggressiveness is inversely correlated with the signal intensity, so that more aggressive lesions imply a lower signal intensity. In addition, in T2W images, the two prostate zones PZ and CG are well-distinguished from each other and from the surrounding non-prostatic tissues. The two zones of a healthy prostate are depicted in Figure 9.3. Since both CG healthy tissue and cancerous tissue have a low signal intensity, detecting PCa in CG is a more difficult task. Furthermore, several factors may reproduce PCa patterns in T2W images, such as BPH or post-biopsy hemorrhage. Consequently, the T2W modality alone is not completely reliable.

2. **Diffusion-weighted imaging (DW).** It is the most recent MRI technique used for the detection and diagnosis of PCa. This modality exploits the variations in the movement of water molecules in different soft tissues. PZ is primarily a glandular structure that allows water molecules to move freely, while CG is composed of muscular or fibrous tissue that constrains the movement of water molecules.

   The growth of PCa destroys the normal glandular structure and increases cellular density. Higher cellular density implies a restricted water diffusion, that is, these factors are inversely correlated. Therefore, water movement is more restricted in prostate cancer regions than in both healthy CG and PZ. The amount of water motion is inversely correlated to the singal intensity. As a result, PCa is characterized by high signal intensity compared to healthy prostatic tissues. However, CG tissues may also present high signal intensity and look similar to PCa.

   As a disadvantage, this technique has low spatial resolution and low specificity due to the detection of false positives.

[1] See NMR Relaxation.

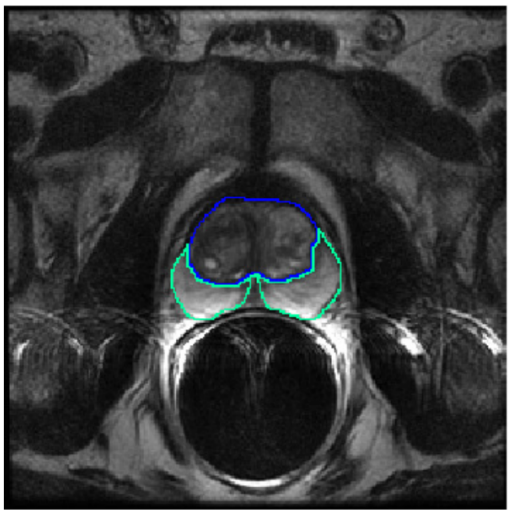


Figure 9.3.: Slice of a T2W image from the I2CVB dataset where a healthy prostate is depicted. The blue contour contains the CG, while the green contour surrounds the PZ. Source [52].

The combination of T2W and DW MRI for the diagnosis of PCa has shown to be more reliable than T2W MRI alone.

3. **Aparent diffusion coefficient map (ADC).** With the objective to eliminate the drawbacks of DW MRI, quantitative maps are extracted from DW MRI, which are known as ADC maps. The tissue appearance in an ADC map is the inverse of DW images, that is, PCa is characterized by low signal intensity and healthy tissue looks brighter. Similar to DW images, some tissues in CG may resemble PCa, and, in addition, hemorrhage may distort the images.

   Diagnosis of PCa using a combination of T2W and ADC map improves in comparison with the use of T2W alone.

4. **Dynamic Contrast-Enhanced imaging (DCE).** This technique exploits the vascularity characteristic of tissues. A contrast agent is injected into the patient which extravasates from vessels to extravascular-extracellular space (EES) and is released back into the vasculature before being eliminated by the kidneys. The vessel networks in tumors are different from the ones of healthy tissues. They are more porous because their capillarity walls present lot of orifices. This high vascular permeability implies more contrast agent exchanges between vessels and EES.

   In healthy PZ, the mainly glandular tissue restricts exchanges between vessels and EES, while the fibrous tissue of the normal CG facilitates the arrival of the contrast agent in EES.

   There exist different ways to analyze the results of DCE qualitatively, semi-quantitatively, and quantitatively, although in practice there is not a known standardization.

   DCE MRI combined with T2W improves sensitivity compared to T2W alone. However, as the above techniques, it has some disadvantages, such as sensitivity to patient

motion, similarity of PCa with prostatitis located in PZ and with BPH located in CG.

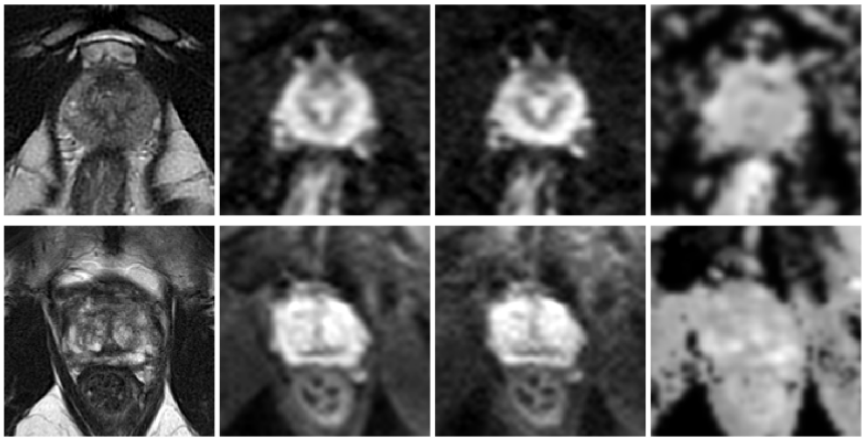


Figure 9.4.: Multiparametric MRI of the prostate of two patients (top and bottom rows correspond to different patients). From left to right: Axial T2W MRI, two DW images (with different values of the attenuation coefficient b[2]) and ADC map of DW MRI. Image obtained from [10].

9.4. Motivation

The interpretation of mpMRI requires an experienced radiologist and it is a labor-intensive, complex, and time-consuming process. Even with the help of PI-RADS, the accuracy of the human mpMRI based diagnosis of PCa depends on the experience of the radiologist, and it suffers from low repeatability as well as mistakes caused by human limitations (such as mental fatigue, distraction, and limited human visual perception) and the complexity of the task itself. As a consequence, there exists non-negligible inter-observer and intra-observer variability.

With the help of computer vision algorithms, the efforts required by radiologists and the time needed for image interpretation can be reduced. Furthermore, the quality of the interpretation and the repeatability can be improved, reducing diagnostic errors at the same time. *Computer-aided diagnosis (CAD)* systems are defined as *the use of computer algorithms to aid the image interpretation process* [59]. Evidence has shown that they improve the diagnostic performance of radiologists. In particular, the automatic segmentation of PCa by means of machine learning algorithms can provide a significant reference for the clinical diagnosis of PCa to the radiologist, allowing for a more focused interpretation, thus improving efficiency and diagnostic accuracy while reducing inter-observer variability.

Machine Learning approaches using multiparametric MRI require registration of the different MRI modalities, since they are normally misaligned due to patient movement between the different modalities acquisitions and differences in resolution. For the registration process, tedious manual annotations of a large number of images in all modalities are needed. Additionally, the acquisition time of multiparametric MRI is notably longer than that of a

[2]See [58] for more information about DW imaging principles.

single modality. Although the combination of different modalities could improve the PCa detection accuracy of CAD systems, this improvement in the performance of the system is minimized by the high labor costs and time resources that the use of mpMRI entails. Therefore, we will opt for the use of a single modality, concretely T2W images, as they present the highest spatial resolution compared to other modalities.

9.5. Databases

In MRI images, the voxel intensities, spatial resolution and other appearance characteristics can greatly differ between acquisition protocols and scanners. Thus, a segmentation model designed for use in clinical practice needs to deal with these issues. If the deep learning segmentation model was trained using data from only one MRI scanner, its generalization ability to data from different scanner manufacturers could be limited. Additionally, training the model with segmentation masks provided by a single radiologist could introduce personal biases, because it would learn that radiologist's limitations and abilities. As a result, the performance of the model when compared with other radiologists' segmentations, especially from other institutions, and, thus, with different formations, could be affected. Consequently, we decided to use different datasets publicly available for the training and validation of our models.

There are not many public databases that include the segmentation masks of the PCa lesions. For this reason, we used the only three databases publicly available that provide segmentation masks of the lesions as well as the prostate zones. Details of each ot them are provided in the first subsection. The second subsection introduces public databases that provide segmentation masks for only the whole prostate gland or/and the prostate zones, which will be used to train and validate the models for the segmentation of the whole prostate and prostate zones.

9.5.1. Segmentation of PCa lesions

The following public databases provide several prostate mpMRI images together with the ground truth segmentation maps of the PCa lesions and the prostate zones. They have been used to train and validate the models that learn to segment lesions present in the prostate, as well as the models in charge of segmenting the whole prostate and its zones.

Considering the images of all these databases, there are 375 patients in total, of whom 185 have at least one malignant lesion and 190 are considered to be healthy. Therefore, the dataset that we will use for the segmentation of PCa lesions is balanced.

9.5.1.1. ProstateX

The ProstateX Challenge database ([60], [61]) consists of 204 patients examined at Radboud University Medical Center between 2011 and 2012. Prostate mpMRIs were acquired using two different types of Siemens 3T MR scanners, Magnetom Trio and Skyra. Sequences collected include T2-weighted (T2W), proton density-weighted (PD-W), dynamic contrast-enhanced (DCE), and diffusion-weighted (DW) imaging. T2W images have a resolution of around 0.5 mm in-plane and a slice thickness of 3.6 mm.

An experienced radiologist, with over 20 years of experience in prostate mpMRI reading, indicated areas of suspicion per modality with a point marker and scored them using PI-

RADS. MR-guided targeted biopsies were performed for areas with associated PI-RADS$\geq$ 3. Subsequently, biopsy specimens were graded by a pathologist.

Lesion coordinates and their associated Gleason Grade (GG) were provided as ground truth (GG was provided later, as part of the ProstateX-2 Challenge). For each patient, one to four lesions were indicated, with an average of 1.62 lesions per patient. A total of 330 findings were provided, of which 76 were defined as clinically significant prostate cancer (CSPCa), that is, with a GS$>$ 6.

In 2021, an external group from the Department of Advanced Biomedical Sciences of the University of Naples "Federico II" [62] performed a quality assessment of the ProstateX training dataset, and provided whole-prostate, zonal anatomy and lesions segmentations for all patients. Two radiology residents (with over 2 years of experience in reading prostate mpMRI) and two board-certified radiologists (with over 5 years' experience) worked in pairs to review all the 204 prostate MRI scans. Of the 330 available lesions, only 299 were verified and segmented individually slice-by-slice on the T2W and ADC images. A total of 76 CSPCa lesions and 223 not clinically significant lesions (NCSPCa) were annotated. Furthermore, the whole prostate gland and its zones were segmented on T2W images.

Since sometimes multiple axial T2W image sequences were available for a patient in the ProstateX dataset, the segmentation of some patients' lesions and the corresponding prostate zones was performed in different sequences. As a result, the same sequence cannot be used to train the deep learning models for the segmentation of the zones and the lesions at the same time. For this reason, 8 patients have been removed from our dataset. We keep another 8 T2W sequences that do not coincide with the corresponding sequences used for the segmentation of the prostate zones, but are the ones employed for the segmentation of the lesions, because in those cases the displacement of the zones segmentation masks with respect to the T2W image kept is not very significant.

Only 196 patients have been used from the ProstateX dataset, of whom 68 present at least one clinically significant lesion and the 128 remaining have low-grade or benign lesions, for a total of 77 CSPCa and 211 NCSPCa lesions. In this way, the number of patients with CSPCa accounts for 34.7% of the dataset and the number of CSPCa lesions represents 26.7% of the total lesions.

39 patients were randomly selected for the test set ($\sim$ 20% of the dataset), but in such a way that 13 have at least one CSPCa ($\sim$ 34% of the patients in the test set). The resulting test set has 16 CSPCa lesions and 54 NCSPCa lesions (the CSPCa lesions account for 23% approx. of the total number of lesions in the test set), and the training set contains 61 CSPCa lesions and 157 NCSPCa lesions (the CSPCa lesions account for 28% approx. of the total number of lesions in the training set). Thus, the proportions of CSPCa lesions and patients with CSPCa are balanced in both training and test datasets.

9.5.1.2. Initiative for Collaborative Computer Vision Benchmarking (I2CVB)

The Initiative for Collaborative Computer Vision Benchmarking (I2CVB) ([63], [52]) provided mpMRI images of the prostate acquired with scanners from two different vendors: a 1.5 Tesla General Electric (GE) scanner and a 3.0 Tesla Siemens scanner. The MRI images were obtained from patients with higher-than-normal levels of PSA, who underwent a subsequent TRUS-guided biopsy that proved the diagnosis. The dataset acquired with the 1.5T GE scanner consists of 21 patients, 17 of which have one CSPCa lesion. Prostate mpMRI images of 19 patients were obtained with the 3T Siemens scanner, including 17 individuals with malignant PCa and 2 healthy patients that had negative biopsies. For each of these patients, the ground

truth segmentations of the cancerous regions, as well as the prostate zones were performed by an experienced radiologist.

Different MRI modalities are available for each patient in the dataset, but only the T2W sequences have been used in our work. The images of these datasets differ in size as well as in pixel and inter-slice spacing. For example, the slice thickness of T2W images acquired with the 1.5T GE scanner is 3 or 3.5 mm, while the slice thickness of T2W images from the 3T Siemens scanner is 1.25 mm.

For the test set, 4 MRI images from each of the two scanners (20% of the complete dataset) were considered. Since 4/21 images from the GE scanner do not have a PCa lesion ($\sim$ 19% of the images), 1 of the images from the GE scanner of the test set is selected to be empty (without lesion), while all the images from the Siemens scanner in the test set contain a PCa lesion because only 2/19 patients are healthy ($\sim$ 11%) in the dataset of the 3T Siemens scanner. In this way, 1/8 patients of the test set are healthy ($\sim$ 13%), which is in correspondence with the 6/40 healthy patients (15%) in the entire dataset.

9.5.1.3. Prostate158

This database [12] contains prostate MRI images from 158 patients between 35 and 84 years old. Of these 158 patients, 139 were taken for the training set and 19 for the test set. 83 patients out of the 139 in the training set have confirmed PCa ($\sim$ 60% of the training set), with a PI-RADS score of 4 or 5, and the rest do not have suspicious PCa, presenting PI-RADS 1 or 2. All of these patients were examined at the Charité University Hospital of Berlin between February 2016 and January 2020.

MRI images were acquired on Siemens VIDA and Skyra clinical 3T scanners. T2W sequences have a slice thickness of 3mm, no interslice gap, and an in-plane resolution of 0.47×0.47 mm.

Images were segmented by two board-certified radiologists with 6 and 8 years of experience in two independent segmentation sessions. Pixel-wise segmentations of the central gland, peripheral zone, and PCa lesions were carried out by the two radiologists, using the axial T2W sequences for the zones segmentation and the ADC maps for the lesions.

Although T2W, DWI, and ADC sequences were provided in this dataset, we only make use of the T2W images. Additionally, only the segmentations performed by one of the radiologists have been used for the training and validation of our models.

Since the original test set is not publicly available, we divided the training set into two subsets, one for training and another one for testing. More precisely, 28 patients from the training set were taken apart ($\sim$ 20% of the training set) as a test set, of whom 17 ($\sim$ 60%) have clinically significant PCa and 11 do not present cancer.

9.5.2. Segmentation of the prostate anatomy

For the training and validation of the models responsible for the segmentation of the whole prostate and prostate zones, different public databases have been used, apart from the ones detailed above. Concretely, data from the Medical Segmentation Decathlon [64], the PROMISE12 [65] challenge and the NCI-ISBI 2013 challenge [66], has been considered. Data from these two last datasets has been obtained from the multi-site dataset [67].

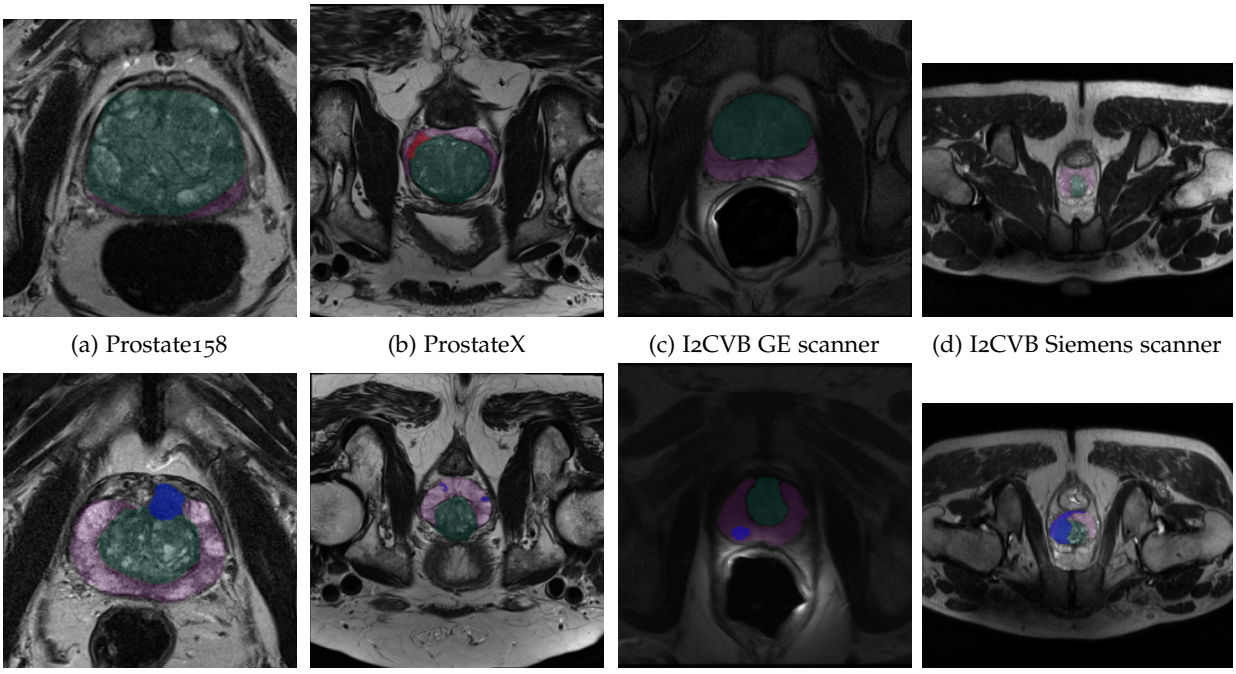
Task 5 of the **Medical Segmentation Decathlon** dataset [68] provides 32 prostate multiparametric MRIs with their corresponding segmentations of the central gland and the peripheral

zone. Data was provided by the Radboud University Nijmegen Medical Centre (The Netherlands). T2W images, with a resolution of $0.6 \times 0.6 \times 4$ mm, as well as ADC maps are available, although we only use the T2W MRIs.

From the **PROMISE12** challenge dataset 37 patients were considered. For each patient, a transversal T2-weighted MR image of the prostate with its corresponding whole prostate segmentation mask is provided. Patients with both benign disease and prostate cancer are included. The data is obtained from multiple centers and multiple MRI device vendors with different acquisition protocols (e.g. different slice thickness). We consider data from the Haukeland University Hospital in Norway (acquired with a 1.5T Siemens scanner and a spatial resolution of $0.625 \times 0.625 \times 3.6$ mm), from the Beth Israel Deaconess Medical Center in the US, (acquired with a 3T GE scanner, an in-plane spatial resolution of 0.25×25mm and a slice thickness of 2.2 or 3 mm), and the University College London in the United Kingdom (from 1.5T and 3T Siemens scanners, with an in-plane resolution of 0.325×0.325mm or 0.625×0.625mm, and a slice thickness of 3 or 3.6 mm).

Similarly, the National Cancer Institute's (NCI's) Cancer Imaging Program in collaboration with the International Society for Biomedical Imaging (ISBI) published a dataset with 60 prostate gland T2W magnetic resonance images, as part of the **NCI-ISBI** 2013 challenge. Data was provided by two different centers, the Boston University and the Radboud University Nijmegen Medical Centre, and obtained with two different scanners, a 1.5T Philips Achieva scanner (with a spatial resolution of $0.4 \times 0.4 \times 3$ mm) and a 3T Siemens TIM scanner (with an in-plane resolution of 0.6×0.6mm or 0.625×0.625mm, and a slice thickness of 3.6 or 4 mm), respectively. Each case consists of a T2W sequence and the CG and PZ segmentation masks marked by two experienced doctors.

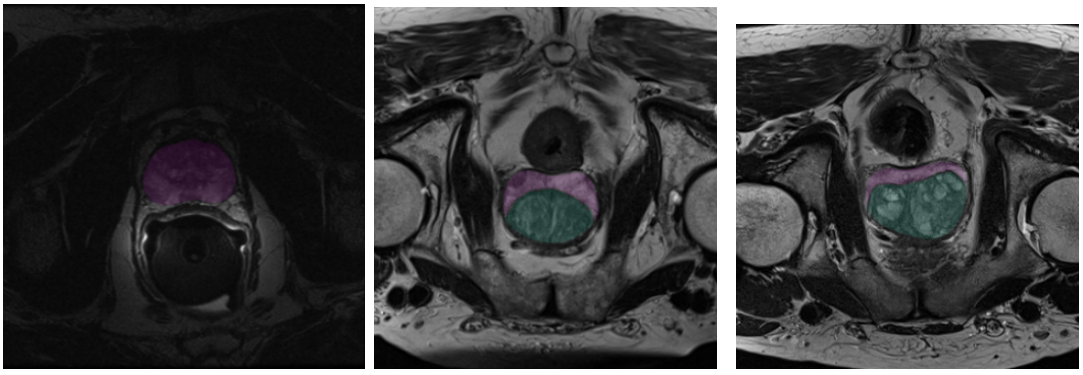
Therefore, in total there are 37 images with only the whole prostate gland segmentation mask available, and 92 patients with their associated prostate zones segmentations. We consider approximately the 20% of this data for testing. That is, 7 of the 37 patients with the whole prostate segmentation and 18 of the 92 patients with the zonal segmentations are taken as a test set.



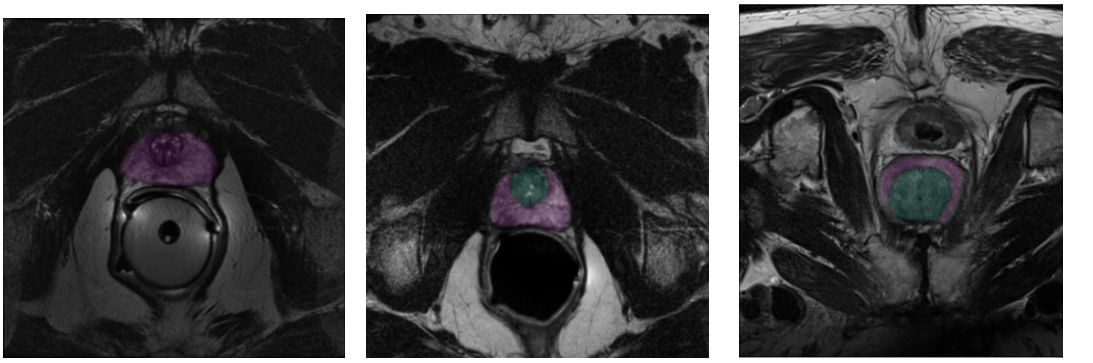
(a) Prostate158 (b) ProstateX (c) I2CVB GE scanner (d) I2CVB Siemens scanner

(e) Prostate158 (f) ProstateX (g) I2CVB GE scanner (h) I2CVB Siemens scanner

Figure 9.5.: T2W MRI slices of two patients from different databases with the ground truth segmentation masks of the prostate zones and PCa lesions. Each row corresponds to a different patient of the same dataset, without malignant lesions at the top and with PCa at the bottom. The green masks correspond to the central gland, the pink masks represent the peripheral zone, the blue regions indicate PCa lesions and the red region indicates a benign or low-grade lesion. Slices (a) and (e) are from the Prostate158 database, (b) and (f) from the ProstateX Challenge, (c) and (g) are from the I2CVB database and obtained with the GE scanner, and (d) and (h) are also in the I2CVB database but acquired with the Siemens scanner. Note the differences in appearance between the images of different scanners and in the segmentation masks. For example, segmentation masks of the prostate zones in the I2CVB dataset present a small gap in the transition between the two zones, while this gap does not appear in the segmentation masks of other datasets.



(a) PPROMISE12 (b) NCI-ISBI (c) Medical Segmentation Decathlon



(d) PPROMISE12 (e) NCI-ISBI (f) Medical Segmentation Decathlon

Figure 9.6.: T2W MRI slices of two patients from different databases with the ground truth segmentation masks of the prostate or prostate zones, as the case may be. Each row corresponds to a different T2W sequence of the same dataset, each provided by a different center in the case that the corresponding dataset contains images from different providers. Slices (a) and (d) are from the PROMISE12 challenge. More precisely, slice (a) corresponds to a T2W sequence provided by the Haukeland University Hospital in Norway and acquired with a 1.5T Siemens scanner, while image (d) was provided by the Beth Israel Deaconess Medical Center and obtained with a 3T GE scanner. In each case, the pink mask corresponds to the entire prostate gland. Slices (b) and (e) are from the NCI-ISBI Challenge, provided by the Radboud University Nijmegen Medical Centre (obtained with a 3T Siemens scanner) (b) and the Boston University (acquired with a 1.5T Philips scanner) (e), respectively. Images (c) and (f) are from the Medical Segmentation Decathlon dataset. In the segmentation masks of these two last datasets, the green region corresponds to the central gland and the pink region represents the peripheral zone. Note the differences in appearance between the images obtained with different scanners.

9.6. Literature Review

CAD systems for PCa lesion detection, segmentation and classification are actively being investigated, as it is demonstrated by a vast and growing literature. Different methods have been proposed since the early 2000s. These methods pioneered the field and mostly relied on hand-crafted features combined with traditional classification methods. More concretely, most of these systems followed a quite similar methodology consisting of 3 steps: feature extraction, feature reduction, and classification using any classical machine learning technique such as logistic regression, random forests, or support vector machines. In **2014, Litjens et al**. [69] presented the first CAD system able to provide PCa lesions candidate regions and their probability of being aggressive using pharmacokinetic, symmetry, and appearance-derived features from different MRI sequences and classical voxel-based classification algorithms.

Since the breakthrough of deep learning in the area of computer vision in 2012, CNN architectures have dominated all types of image analysis applications, including radiological image analysis, and phasing out classical ML classification algorithms. In contrast to the aforementioned machine learning methods, CNNs can be trained *end-to-end*. That is, there is no need for an initial step that requires deep domain specific knowledge and extraction of discriminant features by human researchers. Instead, the algorithm learns the optimal set of features directly from the training dataset.

Several studies have used CNNs to detect PCa lesions, to determine whether a specific prostate lesion represents clinically significant or insignificant cancer, as well as for the segmentation of prostate anatomy and other types of lesion segmentation. Nevertheless, fewer works have investigated CNNs for semantic segmentation of PCa lesions, which is arguably a more challenging task, and the few existing publications are quite recent. Moreover, the dice similarity coefficient results of these studies are all in a low range.

In this section, we briefly survey the literature on MRI-based computer-aided systems that use deep convolutional neural networks for the detection and localization of prostate cancer lesions, that is, we focus our survey on deep learning systems developed for semantic segmentation of PCa lesions.

One of the first publications for the task of CSPCa lesions segmentation using the U-Net architecture was made by **Kohl et al. in 2017** [70]. They proposed a generative adversarial network to segment the prostate into the peripheral zone (PZ), central gland (CG) and clinically significant PCa lesions (with a Gleason Score$\geq$ 7). Their architecture consists of a U-Net-like network that is trained to segment the three classes in a slice-wise manner, and a discriminator which is responsible for distinguishing between predicted (generated) and expert segmentations. They used bi-parametric MRI data (T2-weighted image, Apparent Diffusion Coefficient map and diffusion weighted image) of 152 patients from an in-house dataset and reported a dice similarity coefficient (DSC) of 0.41. This publication was the first to introduce adversarial loss for semantic segmentation of medical images. This idea of employing a generative adversarial network for PCa lesion segmentation was later applied by **Zhang et al. (2019)** [71], who additionally introduced channel and spatial location attention mechanisms into the generative U-Net network to learn more global and local features. Concretely, they added a channel attention layer after each convolutional layer in both the encoder and decoder of the U-Net, and a position attention layer was combined with each skip connection. Only T2W MRI images were used for training their network, but they achieved a DSC of 0.859.

A simpler model to segment the prostate zones and CSPCa lesions that was able to use 3D spatial information was proposed by **Alkadi et al. (2018)** [9]. They employed a deep

encoder-decoder convolutional neural network to segment the anatomical structure of the prostate and CSPCa lesions in both PZ and CG. A 3D sliding window approach was used to incorporate the 3D spatial information included in T2W MRI sequences while preserving the 2D information. Experiments were carried out on the T2W images provided by I2CVB that were acquired with the 3.0T Siemens scanner. With a leave-one-patient out cross-validation scheme, they obtained an average AUC of 0.995, an accuracy of 0.894, a recall of 0.928 and an IoU of 0.679 for the task of segmenting aggressive PCa lesions.

The study by **Hosseinzadeh et al. (2019)** [72] used a preparatory network (an anisotropic 3D U-Net) to produce deterministic or probabilistic zonal segmentation. They experimented with the early and late integration of this information into a 2D U-Net trained to produce a CSPCa probability map from prostate bi-parametric magnetic resonance imaging (bpMRI) and discovered that the early fusion of zonal probabilistic segmentation maps improves the detection results.

Schelb et al.[73] carried out a study published also in 2019 where they directly compared CAD systems for CSPCa lesion detection against radiologist mpMRI assessment and showed that they produced similar detection performance. Indeed, on a test set of 62 patients (sampled from the same study cohort as the training data) their system achieved a patient-level sensitivity of 92% and specificity of 47%, while radiologist assessment resulted in a sensitivity of 88% and a specificity of 50%. Additionally, their method reported a DSC of 0.34 for CSPCa segmentation.

To fully extract the high-level features provided by different MRI modalities, **Chen et al. (2020)**[74] designed three separate branches in the encoder for the three MRI modalities (T2-weighted, apparent diffusion coefficient and high b-value diffusion-weighted imaging images) and proposed a multiple branch U-Net (MB-Unet). In addition, three sub-branches for three consecutive image slices were added to the U-Net encoder to consider 3D spatial information. They worked with 136 prostate mpMRI images from their institution, all containing a PCa lesion with a PI-RADS score$\geq$ 4. For the segmentation of PCa lesions on the test dataset, MB-UNet achieved a per case DSC of 0.6333, specificity of 0.9993 and sensitivity of 0.7056.

The Stanford Prostate Cancer Network (SPCNet) introduced in 2021 by **Seetharaman et al.** [75], also relies on a multi-branch architecture to learn features specific to each MRI modality used (T2W MRI and ADC map) and capture volumetric information (using three adjacent slices). SPCNet was trained to distinguish between normal tissue, aggressive cancer and indolent cancer on bpMRI using pixel-level labels derived from histopathology images automatically mapped onto MRI. It was the first model to be trained and evaluated for the task of segmenting all PCa lesions regardless of whether they are visible on MRI or not. The authors tested their model on two different patient cohorts, obtaining an AUC of 0.80/0.81 to detect normal tissue, 0.64/0.75 to detect indolent cancer, and 0.86/0.89 to detect aggressive cancers at pixel-level, and they found that it showed performance similar to radiologists.

Saha et al. (2021)[76] used a multi-stage 3D computer-aided detection and diagnosis model for automatic segmentation of CSPCa in prostate bpMRI. They added attention mechanisms in a 3D U-Net++ backbone architecture and obtained an increase in sensitivity. An important performance gain was obtained with the addition of probabilistic anatomical prior to the input of the feature encoding branch of their model. They achieved a DSC of 0.49 for the radiologically-estimated training annotations (PI-RADS score$\geq$ 4) and of 0.58 for the histologically-confirmed testing annotations (Gleason score$>$ 6).

De Vente et al. (2021) [77] went a step further and tried to classify, apart from segmenting, the PCa lesions by their level of aggressiveness. They used a 2D U-Net to simultaneously detect and grade PCa lesions on bpMRI images (T2W MRI and ADC map) in an end-to-

end fashion. The output of this network is a lesion segmentation map that encodes the Gleason Grade Group (GGG). The most important contribution of their work is a method for ordinally encoding the model target, which the authors called soft-label ordinal regression, and consists in scaling the GGG scale system to a 0 to 1 scale and performing regression. Additionally, they studied the incorporation of zonal information as an attention mechanism with different strategies, but the results were not statistically better than when omitting zonal information. The dataset from the ProstateX-2 challenge was used but adapted for the segmentation task by delineating ground truth lesions from the lesion centroid coordinates provided in the challenge using in-house software. With 5-fold cross-validation, their model reached a lesion-wise weighted kappa[3] (κ) of 0.172 ± 0.169 and a Dice similarity coefficient for segmenting clinically significant cancer lesions of 0.37 ± 0.046 on the ProstateX-2 train set, and $\kappa = 0.13 \pm 0.27$ on the test set.

Later in 2021, another study was published that also went beyond the binary classification of CS PCa and tried to predict cancer aggressiveness. **Duran et al.** [78] presented the ProstAttention-Net to segment the prostate gland and cancer lesions with their associated GGG on bi-parametric MRI (T2W MRI and ADC map). The network consists of an encoder that extracts high-level features from the input images and a decoder that is divided into two branches: one of them segments the entire prostate gland and the other one uses this zonal information as an attention gate for the detection and grading of cancer lesions. A private dataset formed by 219 MRI images acquired with three different scanners was used for training and validation. The model obtained a Cohen's quadratic weighted kappa coefficient (κ) of 0.418 ± 0.138 during 5-fold cross-validation on the internal dataset, and $\kappa = 0.120 \pm 0.092$ on the ProstateX-2 dataset. For the segmentation of the whole prostate gland, it achieved a DSC of 0.875 ± 0.013.

Recently, in February **2022, Pellicer-Valero et al.** [10] used a Retina U-Net to detect PCa lesions on prostate mpMRI, segment them and estimate their most likely Gleason grade group (GGG). For training, validation and testing a combination of two datasets was employed: the ProstateX-2 dataset (with lesions automatically segmented by growing them from the image coordinates provided in the challenge) and an in-house dataset that they called IVO. In the test set, for the GGG$\geq$ 2 significance criterion, their model achieved a patient level AUC/sensitivity/specificity of 0.87/1.00/0.375 for the ProstateX-2 dataset, and 0.91/1.00/0.762 for the IVO. As for the segmentation of lesions, the average DSC over all patients for segmenting any type of PCa lesions (including benign lesions) was 0.276/0.255 for the IVO/ProstateX-2 dataset when using a 0.25 probability threshold and 0.245/0.244 with a 0.5 threshold.

By the end of **2021, Mehta et al.** [11] introduced AutoProstate, *a deep learning-powered framework for automatic MRI-based prostate cancer assessment*. In particular, AutoProstate consists of three modules, to segment the prostatic zones on T2-weighted imaging (T2WI), detect and segment CSPCa lesions using bpMRI, and generate an automatic web-based report containing several derived characteristics with clinical value. Ensembles of 2D nn-Unets [79] are used for the segmentation of prostatic zones and PCa lesions. AutoProstate was trained using the ProstateX dataset with the ground truth labels provided by Cuocolo et al. [62], and externally validated using a private dataset. During ten-fold cross-validation, a mean DSC of 0.78, 0.86, and 0.91 was achieved for the segmentation of the PZ, CG, and the whole prostate, respectively, and for the CSPCa lesion segmentation their model obtained a mean Dice coefficient of 0.39 and a lesion-level mean AUC of 0.85. In the external validation, it achieved a mean DSC of 0.75, 0.80, 0.89, and 0.46 for the PZ, CG, whole prostate and CSPCa

[3] See Understanding the Quadratic Weighted Kappa.

lesions, respectively, and a lesion-level mean AUC of 0.70 in the segmentation of CSPCa lesions (calculated using segmentation probability maps before thresholding).

The most recent work on this task was published in **July 2022. Liu et al.** [80] presented a more complex model for the accurate segmentation of CSPCa lesions using mpMRI. They proposed a multi-scale segmentation network with four types of cascading pyramid convolution modules in the skip connections between the encoder and the decoder, to improve the feature extraction capability of the network for small targets, and a double-input channel attention module in the decoder, to guide the shallow layer to output the features with higher discriminant ability by using the semantic information of the deep layer features. Furthermore, the feature maps of the skip connections and the decoder go through a residual refinement module to strengthen the recognition ability of each stage. During 5-fold cross-validation the model achieved a DSC of 0.7931 ± 0.0093 for the I2CVB dataset (MRI images obtained with the Siemens scanner), and a DSC of 0.8211 ± 0.0095 for their private dataset.

Also in **July 2022** (but at an earlier date), **Adams et al.** [12] trained two 3D U-Resnets for segmentation of the prostate anatomy and PCa lesions in biparametric MRI, and provided a public database, that they called Prostate158, for segmentation of the central gland, peripheral zone and PCa lesions with a PI-RADS score$\geq$ 4. Their models were trained using the Prostate158 dataset and their generalizability was evaluated on the Medical Segmentation Decathlon dataset and ProstateX dataset (with the labels provided by Cuocolo et al.). Images in the Prostate158 dataset were annotated by two different radiologists and the ground truth labels provided by both of them were used to test the models. In this way, compared to the first radiologist, their models reached a DSC of 0.88, 0.75 and 0.45 for the central gland, peripheral zone and PCa lesions, respectively, and compared with the second radiologist the DSC were 0.88 for CG, 0.73 for PZ and 0.4 for PCa. On the Medical Segmentation Decathlon, their model scored a DSC of 0.82 and 0.64 for the central gland and the peripheral zone, respectively, and on the ProstateX dataset, it reached a mean DSC of 0.86 for the CG and 0.71 for the PZ. Regarding the segmentation of cancer tumors, their model obtained a mean DSC of 0.11 for all aggressive lesions on the entire ProstateX dataset.

10. Methodology

In this work, we develop a deep learning based software for fully automatic segmentation of prostate cancer lesions, both clinically significant and not clinically significant, in T2W MRI images, that we call **AutoPCaSeg**. This software can be found in the Github repository https://github.com/pilarnavarro/AutoPCaSeg under the GNU General Public License, together with a description of the different files it contains and a list of dependencies.

Several deep learning models based on the U-Resnet architecture have been trained for different purposes using prostate T2W MRI images from various databases, previously preprocessed, to homogenize the differences between images in the same dataset and from different datasets, and augmented, to reduce overfitting. All these models have the overall objective of segmenting PCa lesions in the T2W MRI images given as input. To this end, two strategies have been followed. The first strategy divides the task of segmenting PCa lesions into three parts of increasing complexity: first, a model is trained to segment the entire prostate; the information learned is transferred to another model, which is further trained to segment the prostate zones; then, the learned weights are given to a third model, which learns to segment PCa lesions together with prostatic zones. In the second strategy, a new model is trained from scratch to segment only PCa lesions, using as input both preprocessed T2W images and prostate zonal probabilistic segmentations predicted by the model trained in the first strategy. The segmentation masks predicted by the models are further processed in a last step to improve the segmentation results, using prior information about the task at hand.

In this chapter, we describe in detail the methodology employed. We start with the preprocessing pipeline followed by the deep learning models and their architecture. Then, the training process followed is explained and we end the chapter with the postprocessing operations applied.

10.1. Sotfware Tools

Our software has been implemented in Python 3.8.10. This interpreted programming language stands out for its flexibility, simplicity and readability. Additionally, the existence of a great selection of libraries for data analysis, that allow to work with deep learning and machine learning models, is another factor that motivates the use of this programming language.

For the implementation of our models, PyTorch 1.10.2 has been used. It is defined as *an optimized tensor library for deep learning using GPUs and CPUs* [81]. It offers a great flexibility in the implementation of deep learning models, as well as GPU support, and it is fast and easy to code. For these reasons, we decided to use this library.

Additionally, our software is based on the MONAI framework [82]. It is a PyTorch-based framework for deep learning applied to medical images, which provides optimized tools for the development of *healthcare imaging training workflows* and, hence, it is perfectly suitable for our work.

Other Python libraries used for the development of our software are the following:

- Numpy 1.21.2
- Matplotlib 3.5.1
- Scikit-learn 1.0.2
- Segmentation-models-pytorch 0.3.0 [83]
- Scikit-image 0.19.2

10.2. Preprocessing

Due to the heterogeneous nature of the databases used to train our models, the preprocessing of the data is an indispensable step to homogenize differences within and between datasets. Several operations have been implemented to preprocess the data, and a preprocessing pipeline has been designed to require as little human intervention as possible.

Firstly, all images are spatially **resampled to a common resolution** of $0.5 \times 0.5 \times 1.25\text{mm}^3$ voxel size. The 0.5mm for the width and height of the voxels corresponds to the median in-plane resolution of all databases, and 1.25mm is the minimum slice thickness of all T2W sequences. The reason to choose the minimum thickness is to obtain a higher resolution, so that as few fine details as possible are lost when the resampling operation is applied and resampling artifacts are reduced. Trilinear interpolation is used to resample the T2W images, while nearest neighbor interpolation is used for the segmentation masks. Note that the segmentation mask associated with each class (PZ, CG, CSPCa lesions and NCSPCa lesions) is a binary map that is resampled independently of the other masks.

Some T2W sequences contain empty slices at both ends of the volume, that is, there are slices where the prostate is not visible. Therefore, it would seem reasonable to delete some of these empty slices. By **deleting empty slices**, we do not only get rid of a lot of useless information, but also we can obtain a similar number of slices for all T2W sequences. We proceed as follows:

1. First, a desired final number of slices is set.
2. We alternatively delete an empty slice from each end of the volume, until the T2W sequence contains the specified number of slices or until only one empty slice is left at each end, whatever happens first. We keep one empty slice at each end to preserve some possible important information that the empty slices could provide.
3. If the original number of slices is lower than the target value, no empty slices are deleted.

We select the number of slices to be a multiple of $2^4 = 16$, so that a downsampling operation that halves the number of slices can be applied 4 times. Concretely, we choose a target value of 32 slices, so the volume obtained after the last resampling does not contain only one slice.

After deleting empty slices, approximately 28% of the images contain 32 slices and most other T2W volumes present a number of slices close to 32. Linear interpolation is then applied to **resize the depth** of the T2W volumes to 32. For the corresponding segmentation masks, nearest neighbor interpolation is used instead. Thanks to the elimination of empty slices, this resize operation does not disrupt much the original images.

(a) Beginning of a T2W sequence

(b) End of a T2W sequence

Figure 10.1.: Example of empty T2W slices from a patient in the ProstateX dataset. Slice (a) is at the beginning of the sequence, while (b) is one of the last slices.

For **intensity normalization**, simple Z-score normalization is applied to each patient individually: the mean and standard deviation of the voxel intensity values of each patient are computed; then, the corresponding calculated mean is subtracted to the voxel values of each patient and the result is divided by the corresponding standard deviation. In this way, every T2W image in the dataset has a mean intensity value of 0 and a standard deviation of 1.

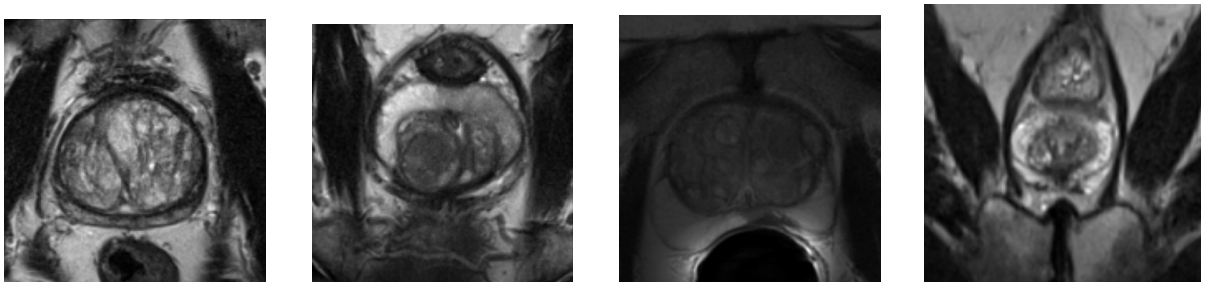
In order to reduce the interference of irrelevant background noise, all slices of every T2W sequence and the corresponding segmentation masks are **cropped around the center** of the image to a common size of 192×192. We visually check that the entire prostate gland is fully contained in the resulting images in every case and larger cropping is not possible without affecting the prostate in some cases. For memory and time efficiency, the spatial size of each slice is further reduced to 160×160 using bilinear interpolation for the images and nearest neighbor interpolation for the segmentation masks. This size has been chosen to be able to perform downsampling 4 times, and, at the same time, to not lose much information with a very small size.

Some segmentation masks corresponding to different classes are overlapped in the original datasets. Since PCa lesions are most frequently contained in the prostate, their **segmentation masks intersect** with the masks of the prostate zones. Therefore, to train a model that learns to segment PCa lesions and the prostate zones simultaneously, we must get rid of these intersections to obtain disjoint classes. For this purpose, overlapping regions are set to 0 in the corresponding prostate zones' segmentation masks. Similarly, the segmentation masks corresponding to not clinically significant PCa lesions and clinically significant PCa lesions may overlap. In this case, we prioritize CSPCa lesions over NCSPCa lesions. That is, the overlapping area is set to 0 in the segmentation mask associated with NCSPCa lesions. Note that this procedure is only applied when necessary, not in every case.

The **background segmentation mask** is computed for all patients as the complement of the union of all the masks needed in each case. Additionally, in the cases when the **segmentation mask of the entire prostate** is necessary but not included in the dataset, it is calculated as the union of the segmentation masks associated to each of the prostate zones, which are always available.

Finally, the binary segmentation masks of all classes, including the background, are stacked together in a single **one-hot encoded multi-channel segmentation mask**. The number of

channels of this combined ground truth segmentation mask will depend on the segmentation task that we are trying to solve. In this way, for example, for the segmentation of only the prostate zones, it will have 3 channels (corresponding to the background, CG and PZ), while for the segmentation of CSPCa lesions, NCSPCa lesions and the prostate zones it will contain 5 channels (corresponding to the background, CSPCa lesions, NCSPCa lesions, CG and PZ).



(a) Prostate158 (b) ProstateX (c) I2CVB GE scanner (d) I2CVB Siemens scanner

Figure 10.2.: T2W MRI slices of patients from different databases after preprocessing.

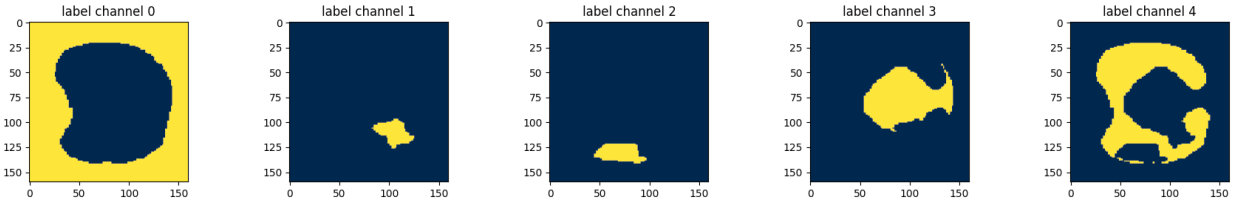


Figure 10.3.: Multi-channel ground truth segmentation mask after preprocessing associated with a patient from the ProstateX dataset. The yellow regions represent the segmentation masks and each channel corresponds to the following class: 0 - Background, 1 - CSPCa lesions, 2 - NCSPCa lesions, 3 - CG, 4 - PZ

10.3. Proposed Models

Several models have been designed with different purposes, but with the final common objective of segmenting clinically significant prostate cancer lesions in T2W MRI sequences. The same network architecture is used for all of them. Concretely, the U-Resnet architecture described in chapter 8 has been used (see Figure 8.7) with some modifications:

- The number of **filters** in both the contracting and expanding paths are $16, 32, 64$ and 128, while the transitional block contains 256 filters. That is, the network has 4 residual blocks in the part corresponding to the encoder and 4 upsampling blocks in the part corresponding to the decoder, as well as a transitional block.
- Each **residual block** contains 2 convolutional layers.
- **Dropout** is applied with a probability of 0.2.

- Due to the heterogeneity of our dataset, there is a high probability of inter-patient intensity variability. Therefore, to ensure that the contrast of an input image is not skewed by being batched with images having significantly different contrast ranges, **instance normalization** is applied after each convolution instead of batch normalization. Additionally, we will use a batch size of two, and, as we explained in section 7.5, with a small batch size instance normalization performs better.

We decided not to include a higher number of blocks in each path or a higher number of filters to keep the architecture as simple as possible. That is, a network with more layers would have more parameters to learn, and thus, it would be a more complex model which could lead to overfitting. Additionally, the time needed to train such a complex model would increase as well. What is more, if we added only one more block to each path of the network, the volume obtained after downsampling the input 5 times would be too small. In fact, with our volumes of 32 slices, it would end up being a 2D image, without depth dimension, so the information would be too compressed.

In the following, we introduce the different models used in our work.

1. **P-Seg**. This model learns to **segment the entire prostate gland**. For this purpose, it is trained with T2W images from all the available training datasets explained in section 9.5, since they all provide the segmentation masks for the entire prostate or the prostate zones (from which the segmentation mask of the prostate can be obtained as already mentioned). This data is preprocessed as explained above and in such a way that the resulting multi-channel one-hot encoded ground truth segmentation mask contains only 2 channels, corresponding to the background and prostate gland classes.

2. **Z-Seg**. This model is responsible for the **segmentation of prostate zones**, PZ and CG. The weights of this model are initialized with the weights learn by P-Seg, so that the training process of Z-Seg is faster and less data is needed to train it properly. Indeed, we have more data to train P-Seg than Z-Seg because there are more databases providing the segmentations of the whole prostate than databases that include the segmentation masks of both prostate zones. Z-Seg is trained with the data of all databases but PROMISE12.

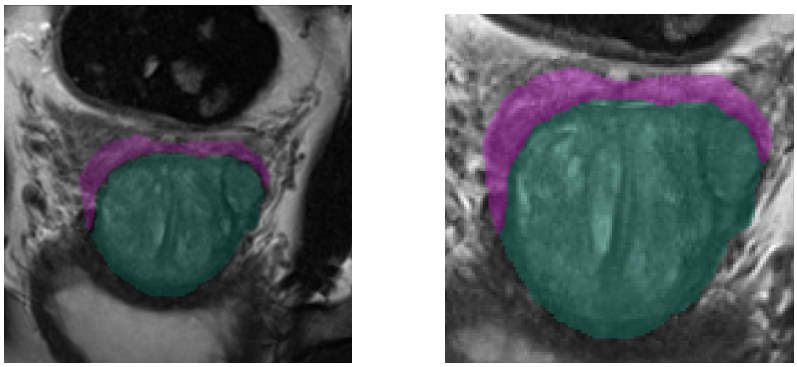
3. **PCa-Seg model 1**. It is designed to **segment the distinct prostate zones and the PCa lesions** at the same time. As the previous model, it uses weights that have been pre-trained, but in this case by Z-Seg. Since we only have 3 databases including PCa lesions and prostate zones segmentation masks, concretely ProstateX, I2CVB and Prostate158, the amount of data available to train this model is less than the data used to train Z-Seg. Therefore, the pre-training allows the PCa-Seg model 1 to have access to information learned from other databases as well.

   The T2W images used to train (and validate) this model are first introduced in the model Z-Seg, trained beforehand, which will segment the prostate zones present in the images. These predicted segmentation masks of the prostate zones are then post-processed (as we will explain in a later section) and used to calculate a **bounding box around the prostate**. Both T2W images and ground truth masks are cropped according to the calculated bounding box, but with an extra margin of 10 pixels at each border, so that possible lesions that outgrow the prostate are not cropped. Then, a resize operation is applied, using bilinear interpolation for the T2W images and nearest neighbor interpolation for the masks, to obtain a spatial resolution of 96×96 voxels

(note that 96 is a multiple of $2^4 = 16$). After these operations, the rest of the necessary preprocessing that was not applied to the images for their use with the Z-Seg model is carried out, such as the elimination of intersections, and the channels of the ground truth segmentation mask are adjusted accordingly.

This model can be trained, at the same time, to perform two different tasks:

- **CSPCa-Seg**. Only clinically significant lesions are segmented, that is, those with a GS$>$ 6 or PI-RADS$\geq$ 3. Therefore, in this case, the multi-channel ground truth segmentation map contains 4 channels, corresponding to the background, CSPCa lesions, CG and PZ. In the case a patient does not have any CSPCa lesion, the channel corresponding to that class will be empty.
- **CSPCa/NCSPCa-Seg**. Both clinically significant lesions and not clinically significant lesions (GS$\leq$ 6 or PI-RADS$<$ 3) are segmented, and each of them is classified according to its class. Here, the multi- channel ground truth segmentation mask has 5 channels, corresponding to the background, CSPCa lesions, NCSPCa lesions, CG and PZ. Since not all databases include segmentation masks for the NCSPCa lesions (in fact only the dataset of ProstateX does), an empty channel is included as the segmentation mask of this class in those cases.



(a) Before bounding box cropping (b) After bounding box cropping

Figure 10.4.: Effect of cropping the bounding box around the prostate. Example of a T2W slice from a patient in the ProstateX dataset. The segmentation masks shown here correspond to the ground truth, but the masks used for the bounding box cropping were the ones predicted by Z-Seg. The pink region represents the peripheral zone of the prostate while the green region is the central gland.

4. **PCa-Seg model 2**. This model learns to **segment only the PCa lesions**, as opposed to model 1, which also segments prostatic zones. It is not pre-trained as the previous models, but it is trained from scratch. The databases employed for the training and validation of this model have been the same used for PCa-Seg model 1, that is, ProstateX, I2CVB and Prostate158.

   Just as with PCa-Seg model 1, the images used to train (and validate) PCa-Seg model 2 are **bounding box cropped** following the same procedure explained before, and resized to 96×96 voxels. After that, the remaining necessary preprocessing is applied to the data. For example, intersections between lesions are eliminated when necessary

and the channels of the combined ground truth segmentation mask are adapted as appropriate.

This model receives as input not only the preprocessed T2W images but also the **probability maps of the prostate zones** predicted by Z-Seg cropped according to the calculated bounding box of the prostate and resized to 96×96 voxels by nearest neighbor interpolation. Therefore, the input to PCa-Seg model 2 has 3 channels.

Two submodels associated with PCa-Seg model 2 can be distinguished, depending on the task they perform:

- **CSPCa-Seg**. Only clinically significant lesions are segmented. The multi-channel ground truth segmentation map contains 2 channels, corresponding to the background and CSPCa lesions.
- **CSPCa/NCSPCa-Seg**. Both clinically significant lesions and not clinically significant lesions are segmented, each of them classified according to its class. The multi-channel ground truth segmentation mask has 3 channels, corresponding to the background, CSPCa lesions and NCSPCa lesions.

Observation 10.1.

- We crop the images according to the bounding box of the prostate, so that the models that learn to segment PCa lesions can focus on learning information only from the region of interest where PCa lesions are present, and forget about the unimportant background information. This operation is not applied to the data for the first two models (P-Seg and Z-Seg), because it is indeed their task to learn to differentiate the prostate in a larger image. Additionally, note that for the bounding box calculation, the ground-truth segmentation masks are not used, but the masks predicted by Z-Seg are used instead, so that we are not incurring in data snooping.
- The strategy of initializing the weights of a network with the weights learned by the same network trained to perform a similar task is known as *transfer learning* and is a common technique used in deep learning. Deep learning models require a long time to be trained and a huge amount of data, but with transfer learning effective models able to learn complex tasks can be trained faster and with less amount of data than they would have needed if they were trained from scratch.
- Recall that PCa lesions vary in frequency, malignancy and appearance in different prostate zones. Therefore, to increase lesion detection performance, we decided to include zonal information in the models responsible for segmenting PCa lesions in different ways. In model 1 this information is introduced by transfer learning and by forcing the network to learn to segment prostate zones together with PCa lesions, whereas in model 2 predicted prostate zonal probabilistic segmentations are given as input to the model. This last approach has shown good results in the literature and the best among the different methods studied in [72].

To sum up, we follow two strategies to segment PCa lesions.

1. The **first strategy** divides the task of segmenting PCa lesions into three parts of increasing complexity. First, P-Seg is trained to segment the entire prostate, and the information learned is transferred to Z-Seg, which is further trained to segment the prostate zones. Then, the learned weights are given to PCa-Seg model 1, which learns

to segment PCa lesions together with prostatic zones. The amount of data available to train the models decreases as the task complexity increases, which makes it even more difficult for the models to obtain good results. This is one of the reasons why transfer learning is used, to alleviate the problem of data scarcity.

2. In the **second strategy**, PCa-Seg model 2 is trained from scratch to segment only PCa lesions. To this end, both preprocessed T2W images (with their corresponding ground truth segmentation masks) and prostate zonal probabilistic segmentations predicted by Z-Seg (trained beforehand as in the first strategy) are provided to the model. As opposed to PCa-Seg model 1, this model is not pre-trained because it is not in charge of segmenting the prostate anatomy, as model 1 does, and the information about prostatic zones that would be transmitted to it by transfer learning is already provided as input.

10.4. Training process

Our models have been trained using an Adam optimizer, with an initial learning rate of 10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, with weight decay and a minibatch size of 2. Due to GPU memory constraints, a larger batch size was not possible, but we consider a minimum of 2 to ensure stability in the learning process since noise in gradients increases with fewer examples in the minibatch. A weight decay factor[1] of 10^{-4} was considered for P-Seg and Z-Seg models, while a decay factor of 10^{-3} was used for the training of the PCa-Seg models. These values for the weight decay, as well as the initial learning rate, were chosen empirically using 3-fold cross-validation (which will be explained in the next chapter). Z-Seg and PCa-Seg model 1 were trained for 400 epochs[2], while the models trained from scratch, i.e., P-Seg and PCa-Seg model 2, were trained for 500 epochs.

Additionally, a *cosine annealing scheduler* [84] is used to schedule the learning rate during the training process. With this scheduler, the learning rate starts with a high value η_{max} and it is relatively rapidly decreased to a minimum predefined value η_{min} before being increased rapidly again to the maximum value. This process is restarted every T_{max} epochs, which is a hyperparameter specified in advance. The resetting of the learning rate can be seen as a restart of the learning process, although not from scratch but with the weights learned until that moment as the starting point. This restart is known as a *warm restart*. The learning rate at each epoch t is calculated as follows:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right),$$

where η_{min} and η_{max} are ranges for the learning rate, T_{cur} is the number of epochs since the last restart, and T_{max} accounts for the number of epochs between two consecutive restarts. In our case η_{max} corresponds with the initial learning rate (10^{-3}), η_{min} is equal to 0 and T_{max} is set to 100 epochs.

As loss functions (see section 8.3), due to the imbalanced representation of our classes in the images, we tried soft dice loss, weighted cross-entropy loss and an average of both of them, but the best segmentation results were obtained with the dice loss function alone. For the weighted cross-entropy loss, the inverse of each class frequency in the training dataset

[1] The decay factor refers to the regularization parameter λ. See section 4.5 for more details on this.
[2] The number of epochs determines the number of times that the neural network will pass through the entire training dataset.

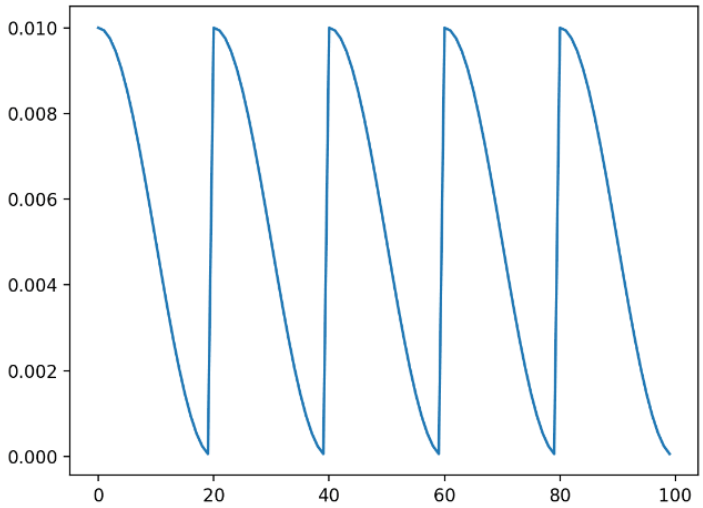


Figure 10.5.: Cosine annealing learning rate schedule with $\eta_{max} = 0.01$, $\eta_{min} = 0$ and $T_{max} = 20$. Source https://machinelearningmastery.com/snapshot-ensemble-deep-learning-neural-network/.

was used as its associated weight. More precisely, denoting by V_T the total number of voxels in the entire training dataset and by V_c the number of voxels of each class c, the frequency of the class c in the training dataset was calculated as $f_c = V_c / V_T$. Then, the weight associated to class c was chosen to be

$$w_c = \frac{1}{W f_c},$$

where $W = \sum_c \frac{1}{f_c}$, so that the weights add up to 1.

To reduce overfitting and to expand the amount of available training data, 3D data augmentation was employed during training. All augmentations were applied stochastically according to a predefined probability. For the training of P-Seg and Z-Seg models, the following data augmentation techniques were applied:

- Random **flip** along all 3 axes with a probability of 0.7 for each axis.
- **Translation** in the first two spatial dimensions by a random factor in the range $[-20, 20]$ with a probability of 0.7.
- **Zoom** in all dimensions by a random factor within the range $[0.9, 1.1]$ with a probability of 0.7.
- **Rotation** in the plane defined by the second and third axes by a random angle uniformly sampled from $U(-20°, 20°)$ with a probability of 0.7.
- **3D elastic deformations** with a probability of 0.5, using random displacement vectors on a coarse grid with a random size uniformly sampled from $[10, 20]$ in each dimension. The displacements are sampled from a Gaussian distribution with a random standard deviation uniformly sampled from the range $[3, 5]$. Per-voxel displacements are computed using interpolation.

- Random **non-linear intensity transformation** with a probability of 0.5. The image intensities are scaled to the range $[0, 1]$. Then, intensity values are raised to the power of a random value $\gamma \sim U(0.5, 2.0)$. Voxel intensities are subsequently scaled back to their original value range. That is, each voxel intensity is modified as follows:

$$i = \left(\frac{i - \min}{\text{original intensity range}} \right)^{\gamma} \times \text{original intensity range} + \min$$

- Zero centered additive **Gaussian noise** with a standard deviation of 0.1 is added to each voxel independently with a probability of 0.5.

For all transformations where interpolation is needed (translation, rotation, zoom and elastic deformations) bilinear or trilinear interpolation was applied, as appropriate, for the images, while nearest neighbor interpolation was used for the corresponding segmentation masks. Border padding[3] was added, when necessary, to keep the original image size.

Since the images used to train the PCa-Seg models are cropped to the region of interest, it makes no sense to apply some of the spatial transformations mentioned above to augment the data used to train those models. Concretely, translation and zoom transformations are not used. In addition, the task of segmenting the lesions is more complex and requires learning finer details from the image. Therefore, we tried not to disrupt too much the augmented data, so for the elastic deformations, a smaller grid was considered, with a size ranging in $[5, 10]$ in each dimension. For the same reason, random Gaussian noise was not added. The rest of the transformations were applied to augment the data for the training of the PCa-Seg models in the same way.

Data from all databases were augmented 2 times for each training process, except for the data of the I2CVB database, which was augmented 6 times to compensate for the lower number of patients in this dataset.

10.5. Postprocessing

We can use prior knowledge about our task to further process the segmentation masks output by the models and improve the segmentation results. For example, we know that there can be only one instance of each prostate zone in a T2W image. To this end, we have performed several operations that are described in the following.

The segmentation probability maps predicted by our models are converted to a multi-channel one-hot encoded segmentation mask by selecting for each voxel the channel with the maximum probability.

Once we have a deterministic segmentation mask, **possible holes inside the segments are filled**. By hole we mean a voxel labeled with a certain class that is enclosed by voxels of a different single class in all three dimensions. These holes are then assigned the same class as their surrounding voxels.

Only one segment corresponding to each prostatic zone can appear in an image, as we have already mentioned, and a patient from our databases can present at most 3 PCa lesions of each type (clinically significant and not clinically significant), which is the case only in the data from the ProstateX challenge (patients from other datasets have only one lesion). Therefore,

[3] With border padding, the borders of the image are extended by replicating the values of the voxels of the closest corresponding border.

we run a **connected component analysis** with a 3-connectivity rule (of full connectivity) [4] to determine the number of instances of each class that appear in the predicted segmentation mask. In case the task at hand consists in segmenting the entire prostate gland, we remove all the instances in the predicted segmentation mask but the largest connected component. For the prostate zones, the same procedure is applied, that is, only the largest component is kept for each zone. In contrast, for the PCa lesions, we consider the 3 largest components of each type and remove the rest. If less than 3 connected components of a PCa lesion type are present, no removal takes place.

Finally, **small connected components associated with PCa lesions are removed** to reduce possible false positives. More precisely, connected components smaller than 40mm^3 are removed. UK National Institute for Health and Care Excellence guidelines [85] recommend a minimum size of 200mm^3 for CSPCa lesions, but we consider a size of 40mm^3 (20% of 200mm^3) to account for partially segmented lesions.

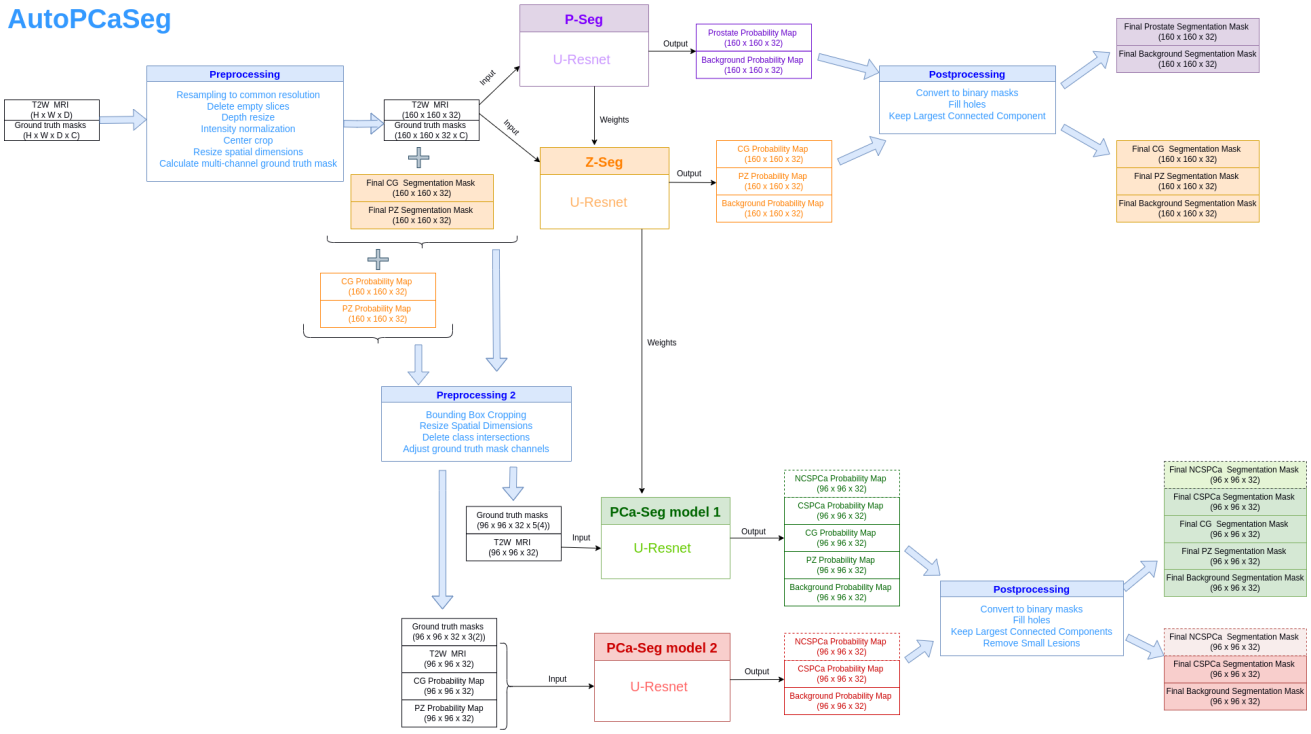


Figure 10.6.: AutoPCaSeg diagram where the different models involved in the task of segmenting PCa lesions are shown as well as their input and output data. Data processing steps are shown in the blue boxes. The dimensions of the images before and after each process are indicated in the data boxes. The letter C in the dimensions of the ground truth mask obtained after the first preprocessing indicates that it contains the number of channels corresponding to each case. This number is later adjusted appropriately to train each model. Boxes with dotted borders in the outputs of the PCa-Seg models denote that the data they contain may be or not be output by the model, depending on the task it performs.

[4] Two voxels are considered to be connected when they are neighbors in any of the three spatial dimensions and have the same associated label.

11. Experimental Results

In this chapter we describe the experimental framework, present the results obtained by our models (quantitative and qualitative), both during validation and on the different test sets, and discuss the main findings, comparing the performance of our models with state-of-the-art results and describing their limitations.

A wide variety of metrics has been included to evaluate the performance of our models, although the main metric considered throughout our work is the Dice Similarity coefficient since our primary objective is to segment different elements inside the prostate. A three-fold cross-validation technique was employed for the evaluation of the models, and an ensemble of the three instances of each model obtained after cross-validation was considered for testing. Tes-time augmentation was applied using random flip and random rotation transformations to improve the segmentation results.

During three-fold cross-validation, P-Seg obtained a mean DSC of 0.9049 for the segmentation of the entire prostate gland, Z-Seg achieved a DSC of 0.8506 and 0.7283 for the CG and PZ, respectively, and the best DSC (not enhanced) obtained by the PCa-Seg models for the segmentation of clinically significant PCa lesions was 0.16.

During testing on different test sets, the DSC values obtained by the P-Seg model ranged from 0.858 to 0.917, while the DSC values achieved by Z-Seg were in the range from 0.67 to 0.877 for the central gland, and 0.637 to 0.7731 for the peripheral zone. For the segmentation of not clinically significant lesions on the ProstateX dataset, PCa-Seg model 2 reached a DSC of 0.118. Concerning malignant lesions, on the test data from the I2CVB dataset acquired with the Siemens scanner, the best DSC value achieved by the PCa-Seg models was 0.496 and the best AUC 0.737, while on the test images from the same dataset but acquired with the GE scanner the highest DSC obtained was 0.223 and the highest AUC 0.724. On the Prostate158 test set, the best results were a DSC of 0.1679 and an AUC of 0.771, and for the ProstateX dataset, the highest values were 0.116 for the DSC and 0.897 for the AUC.

11.1. Experimental Framework

11.1.1. Evaluation Metrics

In this section we introduce the metrics employed to quantitatively evaluate the segmentation results of the different models. In general, metrics are a number in the range from 0 to 1, with a higher value representing a better result. All classes are evaluated individually on a one vs all manner. More precisely, every metric is calculated independently for each class and each patient by comparing voxel-by-voxel the deterministic multi-channel segmentation mask obtained after post-processing the probability maps predicted by the model and the one-hot encoded ground truth segmentation mask. Then, the average is computed over all patients in the dataset for each class.

First of all, the number of true positive (TP), false positive (FP), true negative (TN) and

false negative (FN) predictions are calculated for each class on a voxel-by-voxel basis. Once these values are known, they are used to calculate the following metrics:

- **Accuracy.** It measures the proportion of correctly classified voxels among the total number of voxels in the image.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Precision/Positive predictive value.** It is the ratio of correctly classified voxels to the total number of voxels predicted as a positive class (i.e. as the class whose precision we are calculating). The more false positives the model predicts for a class, the lower its associated precision.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall/Sensitivity/True positive rate.** It determines the proportion of voxels belonging to a class that is actually labeled with that class. The more false negatives the model predicts for a class, the lower its associated recall.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Specificity/Selectivity/True negative rate.** It measures the fraction of negative voxels (i.e. voxels from a different class) that are correctly classified as negative (i.e. as not belonging to the class whose specificity is being calculated).

$$\text{Spec} = \frac{TN}{TN + FP}$$

- **Area under the ROC curve (AUC).** AUC measures the ability of the model to distinguish between classes. The higher the AUC, the better the performance of the model at separating different classes. It is used as a summary of the ROC curve, which is obtained by plotting the sensitivity against $1-$specificity for all possible probability threshold values used to consider a voxel as belonging to the positive class (i.e. as the class whose ROC curve we are plotting). [1]That is, AUC summarizes the sensitivity and specificity for all possible threshold values. As its name indicates, AUC is computed as the area under the ROC curve. For calculating this area, we use the trapezoidal rule [86]:

$$\text{AUC} = 1 - \frac{1}{2}\left(\frac{FP}{FP + TN} + \frac{FN}{FN + TP}\right)$$

- **F1-score/Dice Similarity Coefficient.** It is the coefficient explained in section 8.3. Apart from the formulation introduced in that section, it can also be calculated as follows:

$$\text{DSC} = \frac{2TP}{2TP + FP + FN} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

[1]For more information about this metric and the ROC curve see AUC-ROC Curve.

- **Intersection Over Union (IoU) / Jaccard Index.** Just as the Dice Similarity Coefficient, this metric measures the overlap between the ground truth segmentation mask (G) and the predicted one (P). IoU is the ratio of the intersected area (G∩P) to the combined area (G∪P) of predicted and ground truth masks. The intersection contains the voxels that are present in both the predicted mask and the ground truth mask, while the union consists of all voxels found in either of the masks. It is calculated as follows:

$$\text{IoU} = \frac{TP}{TP + FP + FN}$$

The denominator of some of these metrics may take the value zero. Indeed, many patients in our datasets do not have any PCa lesion and if the model predicts correctly that no voxel of the image belongs to that class, the number of associated TP, FP and FN would be equal to zero. In those cases, we set the metric to be equal to 1 because the model acts correctly. However, in the particular case of the dice similarity coefficient, we follow two strategies when the denominator happens to be equal to zero for a certain patient: that patient is not considered when computing the average DSC value over the entire dataset, or the value of the metric is set to 1 for that patient, and, in this case, we call the resulting metric **Dice enhanced** to distinguish it from the other strategy.

Although we calculate all the above metrics to be able to compare our models with other works in the literature, the main metric used for our experiments has been the Dice coefficient.

11.1.2. 3-fold Cross validation

A 3-fold cross-validation strategy has been followed for the evaluation of our models. More concretely, the training dataset was randomly divided into 3 disjoint parts (folds). Each time two of these parts were selected as the training set for all models, that is, all models used the same training data each time, and the remaining one part was used as a validation set (the same for all models). The loss value was monitored during each training and the model checkpoint with the lowest loss value on the corresponding validation set was saved. This process was repeated 3 times, each using a different validation set. In this way, we obtained 3 instances of each model, each trained and evaluated on different data, and the average metric value of the 3 times was used as the final evaluation metric.

Folds were selected in such a way that the same percentage of data from each considered dataset was present in each fold, so that possible differences in the evaluation metrics between folds are not a result of differences in the acquisition methods and quality of the data present in different validation sets.

By using the same folds to train and evaluate all models each time we make sure that no data snooping occurs in the validation process. If different folds were used each time to train and evaluate different models, the information learned by previous models, which is transferred to subsequent models, could contain details from the data used to validate the other models, since that data could be part of the training examples used to train the previous models.

Thanks to cross-validation, the final evaluation metrics do not depend on how the validation set is split and we can have a better idea of the performance of the models on unseen data.

11.1.3. Testing

Predictions on the test sets were made using **cross-validation ensemble models**. For each model, the 3 instances trained using cross-validation were used to construct a single ensemble model. In general, ensembling consists in training different models for the same task, using them to make predictions on a test set, and combining the predictions of the different models using a certain method to obtain a single prediction of better quality than that of each individual model. In our case, we average the probability maps predicted on the test set data by each of the 3 models obtained during cross-validation. Cross-validation ensembling has been shown to be an effective ensembling strategy [79] that helps to prevent overfitting on the training set.

Furthermore, we used **test-time augmentation**[2] to model aleatoric uncertainty. During testing, we augmented the input images using random flip and random rotation transformations with the same probability and same parameters used for the training set. Each modified image, as well as the original image, were processed by the model which computed their respective probability maps. The predicted outputs were then rectified by undergoing the inverse of the transformations applied so that we could match each predicted pixel class with the ground truth labels. Finally, the average of the multiple outputs generated was calculated and used as the final prediction.

We only applied random flip and random rotation to augment the test images because these are the only transformations used to train every model (remember that during training of the PCa-Seg models some augmentation techniques were not used, so it makes no sense to use them during testing) that can be easily inverted, and each test image was augmented 5 times. Different numbers of augmentations ($2, 5, 7, 10, 12, 15, 17$ and 20) have been tried by making predictions on the training set but the best results were obtained for 5 augmentations.

Test-time augmentation was applied before ensembling the cross-validation models. That is, first the predictions of each cross-validation instance of a model on the augmented test data were computed. Then, the average of the predictions of each of the 3 cross-validation instances was calculated and taken as the model's final prediction for the corresponding test image.

11.2. Results

11.2.1. Validation Results

During 3-fold cross-validation the P-Seg model obtained a mean DSC of 0.9049, with a standard deviation of 0.0008, for the segmentation of the entire prostate gland, while the Z-Seg model achieved a DSC of 0.8506 ± 0.0079 and 0.7283 ± 0.0047 for the CG and PZ, respectively. Values of the different metrics achieved by the PCa-Seg models in cross-validation for the segmentation of the different classes are shown in the following tables.

[2] Hoar et al. [87] investigated the effect of test-time augmentation and transfer learning in the segmentation performance of PCa lesions in mpMRI and found that the addition of test-time augmentation improved significantly the Dice scores.

	CSPCa	std	NCSPCa	std	CG	std	PZ	std
accuracy	0.9868	0.0018	0.9947	0.0006	0.9373	0.0015	0.9368	0.0024
precision	0.2455	0.0224	0.2693	0.0920	0.8171	0.0084	0.6703	0.0220
recall	0.6578	0.0188	0.6224	0.0240	0.8236	0.0153	0.7136	0.0250
specifity	0.9919	0.0006	0.9973	0.0006	0.9634	0.0024	0.9629	0.0046
AUC	0.8249	0.0095	0.8099	0.0122	0.8935	0.0067	0.8382	0.0106
Dice	0.1526	0.0053	0.0273	0.0093	0.8115	0.0079	0.6815	0.0096
Dice_Enhanced	0.2063	0.0078	0.1984	0.0760	0.8115	0.0079	0.6815	0.0096
IoU	0.1587	0.0057	0.1895	0.0760	0.6944	0.0099	0.5281	0.0089

Table 11.1.: Results obtained by the **CSPCa/NCSPCa-Seg model 1** in three-fold cross-validation.

	CSPCa	std	CG	std	PZ	std
accuracy	0.9878	0.0037	0.9390	0.0022	0.9391	0.0026
precision	0.3201	0.0742	0.8171	0.0089	0.6821	0.0199
recall	0.6578	0.0122	0.8336	0.0251	0.7295	0.0150
specifity	0.9928	0.0026	0.9631	0.0034	0.9641	0.0034
AUC	0.8253	0.0074	0.8983	0.0109	0.8468	0.0065
Dice	0.1664	0.0065	0.8172	0.0103	0.6957	0.0104
Dice_Enhanced	0.2496	0.0452	0.8172	0.0103	0.6957	0.0104
IoU	0.2000	0.0483	0.7046	0.0131	0.5452	0.0099

Table 11.2.: Results obtained by the **CSPCa-Seg model 1** in three-fold cross-validation.

	CSPCa	std	NCSPCa	std
accuracy	0.9855	0.0017	0.9925	0.0008
precision	0.1880	0.0247	0.1275	0.0580
recall	0.6676	0.0189	0.6500	0.0240
specifity	0.9909	0.0008	0.9949	0.0011
AUC	0.8293	0.0094	0.8224	0.0117
Dice	0.1495	0.0165	0.0430	0.0049
Dice_Enhanced	0.1636	0.0201	0.1069	0.0365
IoU	0.1132	0.0143	0.0905	0.0367

Table 11.3.: Results obtained by the **CSPCa/NCSPCa-Seg-Seg model 2** in three-fold cross-validation.

	CSPCa	std
accuracy	0.9830	0.0021
precision	0.1856	0.0329
recall	0.7111	0.0312
specifity	0.9877	0.0012
AUC	0.8494	0.0157
Dice	0.1666	0.0196
Dice_Enhanced	0.1775	0.0272
IoU	0.1242	0.0212

Table 11.4.: Results obtained by the **CSPCa-Seg model 1** in three-fold cross-validation.

We can observe that the value of the dice coefficient and the IoU is low in general, especially for the not clinically significant PCa lesions. However, the value of the AUC for this class is not low, probably because of the small amount of NCSPCa lesions contained in the training set (recall that only the ProstateX dataset includes patients with segmented NCSPCa lesions), so for the majority of the patients the models correctly predict that no NCSPCa is present and a value of 1 is assigned to the AUC.

Note also that for the PCa lesion classes the values of the *Dice Enhanced* metric are generally higher than for the *Dice* metric. This is because in the cases with no PCa lesions where the models predict that no lesion is present the *Dice Enhanced* metric takes the value 1. Therefore, the higher the difference between the *Dice Enhanced* metric and the *Dice* value, the more negative cases the model predicts correctly.

Another metric that shows a small value for the PCa lesion classes is the precision, which indicates that there are many false positives. In contrast, the specificity is high, so the amount of false positives in comparison with all the voxels that are correctly predicted as negative is small, but we have to consider that there are many more voxels in the training dataset that correspond with healthy tissue than voxels of PCa lesions. Since the recall is not so high, we can hypothesize that there is a considerable amount of voxels that are not correctly identified as positive classes (false negatives).

11.2.2. Test Results

Our models have been tested in the test set of each considered dataset separately. However, data from the Medical Segmentation Decathlon and the NCI-ISBI 2013 challenge have been merged into a unique test set that we call *zones test set*. The I2CVB dataset has been divided into two subsets, each containing the images acquired with a different scanner (3T Siemens scanner and 1.5T GE scanner).

11.2.2.1. Quantitative Results

Values of the dice similarity coefficient achieved by the P-Seg and Z-Seg models on the different test sets are shown in Table 11.5 and Table 11.6, respectively. The best segmentation performance was obtained for the whole prostate, followed by the central gland and the peripheral zone. This fact suggests an ease of distinguishing the prostate gland from background tissue, but a difficulty in differentiating between PZ and CG, being at the same time the segmentation of PZ a more difficult task than the segmentation of CG. The worst segmentation results were obtained for the data in the I2CVB dataset acquired with the Siemens scanner. For the data in the same dataset acquired with the GE scanner, the segmentation of the whole prostate offered a high DSC value but the results for the segmentation of the two prostatic zones were worse and similar to the ones obtained for the images acquired with the Siemens scanner.

PROMISE12	Zones test set	Prostate158	ProstateX	I2CVB GE	I2CVB Siemens
0.9055	0.9126	0.9064	0.9172	0.9105	0.8578

Table 11.5.: Dice similarity coefficient values obtained by **P-Seg** for the segmentation of the whole prostate gland on different test datasets.

	CG	PZ
Zones test set	0.8731	0.6993
Prostate158	0.8771	0.7143
ProstateX	0.8667	0.7731
I2CVB GE scanner	0.6866	0.6951
I2CVB Siemens scanner	0.6708	0.6369

Table 11.6.: Dice similarity coefficient values obtained by **Z-Seg** for the segmentation of the prostate zones on different test datasets.

Regarding the PCa-Seg models, the results for all the datasets are collected in the following tables.

Prostate 158	**CSPCa**	**NCSPCa**	**CG**	**PZ**
accuracy	0.9896	0.9993	0.9455	0.9515
precision	0.3816	0.6786	0.8652	0.6543
recall	0.4748	1.0000	0.8539	0.7551
specifity	0.9947	0.9993	0.9674	0.9681
AUC	0.7347	0.9997	0.9106	0.8616
Dice	0.1189	0.0000	0.8567	0.6909
Dice_Enhanced	0.2762	0.6786	0.8567	0.6909
IoU	0.2469	0.6786	0.7526	0.5355
ProstateX	**CSPCa**	**NCSPCa**	**CG**	**PZ**
accuracy	0.9890	0.9897	0.9403	0.9425
precision	0.2021	0.4550	0.8325	0.7460
recall	0.7480	0.2091	0.8390	0.7285
specifity	0.9916	0.9986	0.9640	0.9694
AUC	0.8698	0.6039	0.9015	0.8490
Dice	0.0912	0.0329	0.8311	0.7305
Dice_Enhanced	0.1611	0.0329	0.8311	0.7305
IoU	0.1330	0.0180	0.7143	0.5820
I2CVB GE scanner	**CSPCa**	**NCSPCa**	**CG**	**PZ**
accuracy	0.9916	1.0000	0.9183	0.8824
precision	0.1282	1.0000	0.7483	0.6468
recall	0.3380	1.0000	0.7055	0.6614
specifity	0.9951	1.0000	0.9585	0.9268
AUC	0.6665	1.0000	0.8320	0.7941
Dice	0.1044		0.7173	0.6519
Dice_Enhanced	0.1044	1.0000	0.7173	0.6519
IoU	0.0660	1.0000	0.5604	0.4838
I2CVB Siemens scanner	**CSPCa**	**NCSPCa**	**CG**	**PZ**
accuracy	0.9645	0.9972	0.9274	0.8982
precision	0.5768	0.2500	0.6969	0.4060
recall	0.4837	1.0000	0.6919	0.7428
specifity	0.9896	0.9972	0.9582	0.9192
AUC	0.7366	0.9986	0.8250	0.8310
Dice	0.4768	0.0000	0.6857	0.4882
Dice_Enhanced	0.4768	0.2500	0.6857	0.4882
IoU	0.3201	0.2500	0.5813	0.3356

Table 11.7.: Results obtained by the **CSPCa/NCSPCa-Seg model 1** on different test datasets.

Prostate 158	**CSPCa**	**CG**	**PZ**
accuracy	0.9911	0.9477	0.9524
precision	0.3847	0.8596	0.6520
recall	0.4854	0.8701	0.7631
specifity	0.9960	0.9656	0.9681
AUC	0.7407	0.9179	0.8656
Dice	0.1279	0.8615	0.6942
Dice_Enhanced	0.2836	0.8615	0.6942
IoU	0.2498	0.7602	0.5398
ProstateX	**CSPCa**	**CG**	**PZ**
accuracy	0.9911	0.9448	0.9454
precision	0.2587	0.8446	0.7735
recall	0.7464	0.8560	0.7431
specifity	0.9938	0.9659	0.9726
AUC	0.8701	0.9110	0.8578
Dice	0.0987	0.8463	0.7509
Dice_Enhanced	0.2143	0.8463	0.7509
IoU	0.1875	0.7368	0.6079
I2CVB GE scanner	**CSPCa**	**CG**	**PZ**
accuracy	0.9951	0.9197	0.8842
precision	0.3866	0.7454	0.6319
recall	0.2928	0.7128	0.7001
specifity	0.9990	0.9584	0.9202
AUC	0.6459	0.8356	0.8102
Dice	0.0652	0.7214	0.6633
Dice_Enhanced	0.0652	0.7214	0.6633
IoU	0.0375	0.5648	0.4971
I2CVB Siemens scanner	**CSPCa**	**CG**	**PZ**
accuracy	0.9682	0.9225	0.8880
precision	0.6426	0.6410	0.3785
recall	0.4638	0.7152	0.7327
specifity	0.9931	0.9459	0.9126
AUC	0.7285	0.8305	0.8227
Dice	0.4959	0.6735	0.4582
Dice_Enhanced	0.4959	0.6735	0.4582
IoU	0.3384	0.5813	0.3081

Table 11.8.: Results obtained by the **CSPCa-Seg model 1** on different test datasets.

Prostate 158	**CSPCa**	**NCSPCa**
accuracy	0.9888	0.9981
precision	0.2682	0.2500
recall	0.5036	1.0000
specifity	0.9944	0.9981
AUC	0.7490	0.9990
Dice	0.1301	0.0000
Dice_Enhanced	0.1922	0.2500
IoU	0.1505	0.2500
ProstateX	**CSPCa**	**NCSPCa**
accuracy	0.9880	0.9886
precision	0.1214	0.2812
recall	0.7792	0.2678
specifity	0.9904	0.9967
AUC	0.8848	0.6322
Dice	0.1095	0.1178
Dice_Enhanced	0.1095	0.1178
IoU	0.0720	0.0697
I2CVB GE scanner	**CSPCa**	**NCSPCa**
accuracy	0.9941	0.9989
precision	0.5156	0.2500
recall	0.4505	1.0000
specifity	0.9969	0.9989
AUC	0.7237	0.9995
Dice	0.2226	0.0000
Dice_Enhanced	0.2226	0.2500
IoU	0.1432	0.2500
I2CVB Siemens scanner	**CSPCa**	**NCSPCa**
accuracy	0.9627	0.9974
precision	0.5869	0.2500
recall	0.4336	1.0000
specifity	0.9915	0.9974
AUC	0.7125	0.9987
Dice	0.4237	0.0000
Dice_Enhanced	0.4237	0.2500
IoU	0.2920	0.2500

Table 11.9.: Results obtained by the **CSPCa/NCSPCa-Seg model 2** on different test datasets.

	Prostate158	ProstateX	I2CVB GE scanner	I2CVB Siemens scanner
accuracy	0.9878	0.9850	0.9912	0.9643
precision	0.2687	0.1063	0.1755	0.5709
recall	0.5497	0.8065	0.4163	0.4511
specifity	0.9923	0.9871	0.9943	0.9885
AUC	0.7710	0.8968	0.7053	0.7198
Dice	0.1679	0.1157	0.1652	0.4549
Dice_Enhanced	0.2570	0.1157	0.1652	0.4549
IoU	0.2114	0.0767	0.1011	0.3220

Table 11.10.: Results obtained by the **CSPCa-Seg model 2** for the segmentation of clinically significant PCa lesions on different test datasets.

In general, the values obtained for all metrics on the different test datasets are in agreement with the values obtained during cross-validation, as expected.

The segmentation performance of the PCa-Seg model 1 for the prostate zones is worse than the performance offered by the Z-Seg model in almost all datasets because PCa-Seg model 1 has to learn to differentiate between normal tissue and a prostate cancer lesion apart from distinguishing the two prostatic zones. This is a more difficult task than the one executed by Z-Seg, which simply has to segment the two prostate zones independently of whether they contain malignant tissue or not.

Since only the dataset from the ProstateX-2 challenge contains data with segmentation masks of not clinically significant lesions the models that learn to segment NCSPCa lesions should not detect this type of lesions in images from the other datasets. However, this is not the case as the evaluation metrics show. If no NCSPCa lesions were detected in patients that do not present these lesions, the value of the evaluation metrics should be 1 (except for the Dice metric which is not defined in this case). Therefore, we can assure that these models predict incorrectly some NCSPCa lesions on datasets where they should not. The higher the value of the evaluation metrics for the NCSPCa class, the fewer incorrect NCSPCa lesions the model predicts.

Three bar charts showing the values of the Dice, Dice Enhanced and AUC metrics obtained by the different PCa-Seg models on all datasets are presented in Figure 11.2, Figure 11.1 and Figure 11.3, to be able to visually compare the quantitative results.

From the graphs for the Dice and AUC metrics we can deduce that the differences between models are not significant and do not follow a clear pattern. Nevertheless, the difference between the values of the dice and dice enhanced metrics is higher for the PCa-Seg model 1 than for the PCa-Seg model 2, so we can say that PCa-Seg model 1 produces fewer false positive predictions than PCa-Seg model 2.

More considerable are the differences between datasets. The best DSC value by far is achieved on the test set formed by the images from the I2CVB dataset obtained with the Siemens scanner, followed by the Prostate158 test set and the test images from the I2CVB dataset obtained with the GE scanner. The worst DSC value is obtained on the ProstateX dataset. In contrast, the best AUC is reached on this dataset, followed by the Prostate158 dataset and the I2CVB dataset.

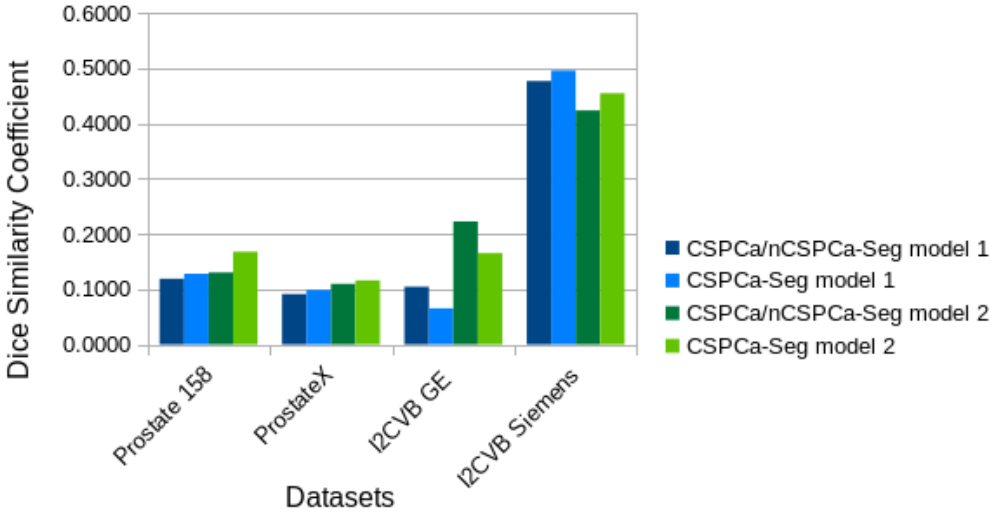


Figure 11.1.: Bar chart showing the DSC values obtained by the different PCa-Seg models on all test sets.

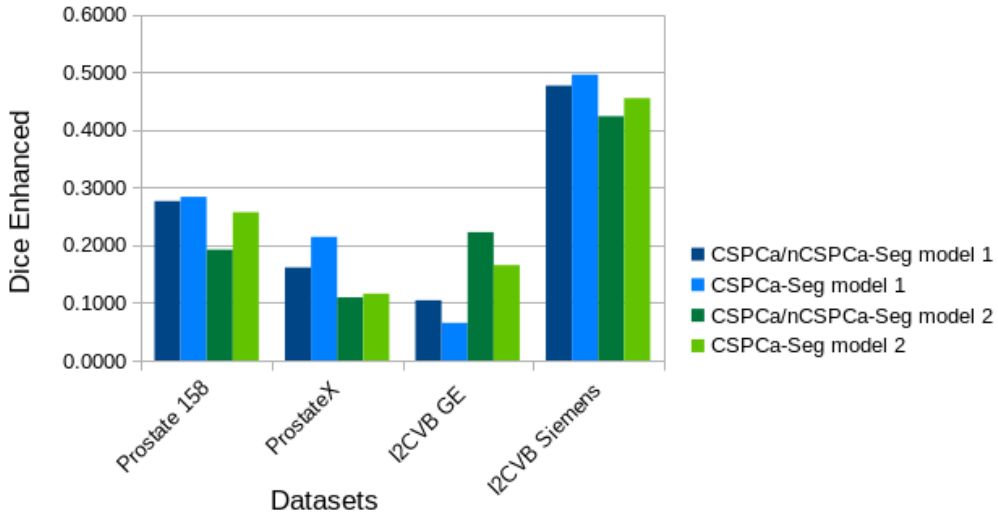


Figure 11.2.: Bar chart with the values of the Dice Enhanced metric obtained by the different PCa-Seg models on all test sets.

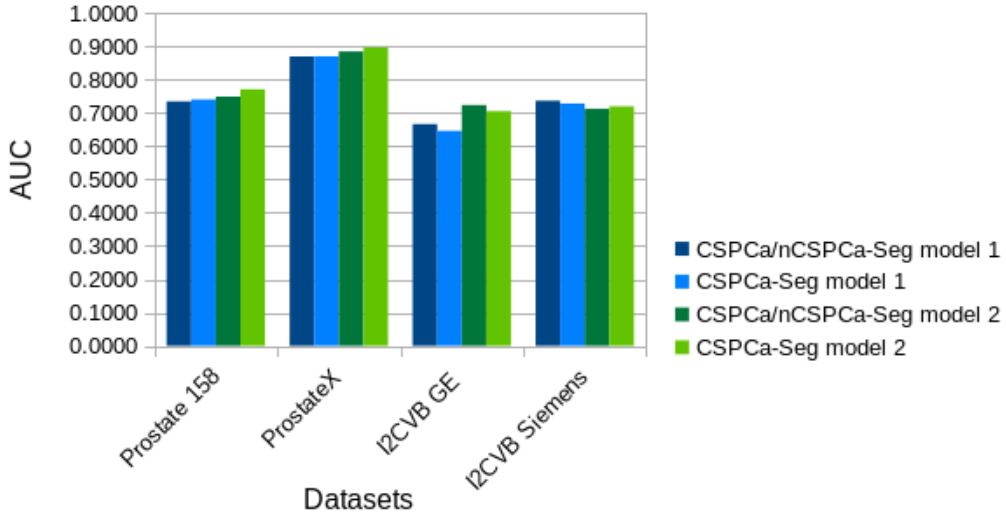


Figure 11.3.: Bar chart showing the AUC values achieved by the different PCa-Seg models on all test sets.

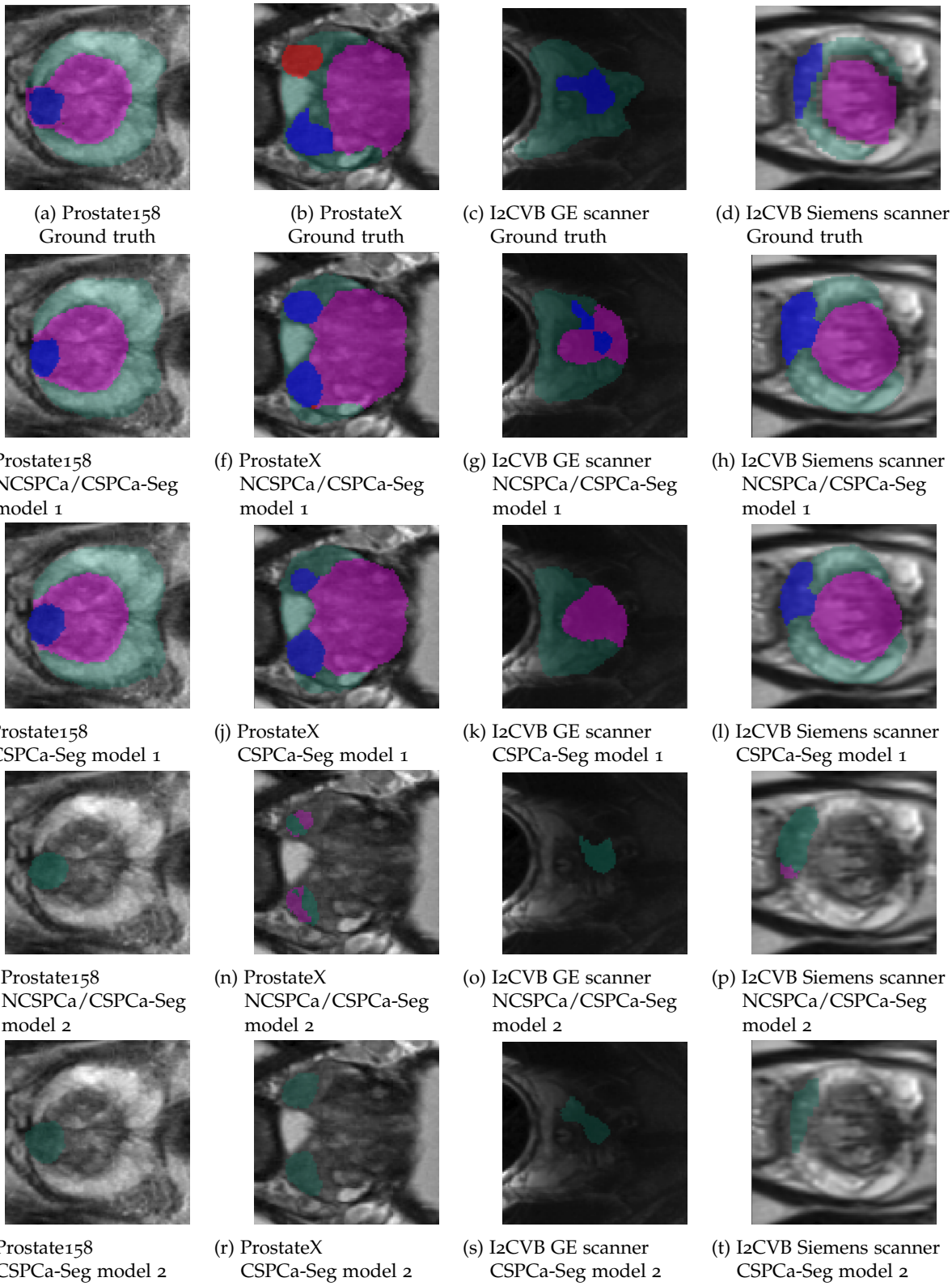
11.2.2.2. Qualitative Results

Predictions of the PCa-Seg models on two patients from each test set are shown in Figure 11.4 and Figure 11.5 together with the corresponding ground truth mask. In the outputs of the PCa-Seg model 1 and the ground truth masks the blue region represents a CSPCa lesion, the red area denotes a NCSPCa lesion and the PZ and CG are represented with a green and pink color, respectively. In the segmentation masks predicted by the PCa-Seg model 2, the green color denotes a CSPCa lesion while the pink color is used to refer to NCSPCa lesions.

We can see that almost all lesions are correctly detected by the different models, although most of them are only partially segmented. Models that learn to detect both clinically significant and not clinically significant lesions tend to mix both classes in these particular examples, as can be seen in Figure 11.4n and Figure 11.4p. The patient from the ProstateX dataset in Figure 11.4 have one clinically significant and one not clinically significant lesion, but the models are not able to classify that lesion as not clinically significant. Even the models that are not trained to segment NCSPCa lesions detect that lesion and segment it correctly but classify it as clinically significant. We can hypothesize that it looks similar to a CSPCa lesion, so the models get confused.

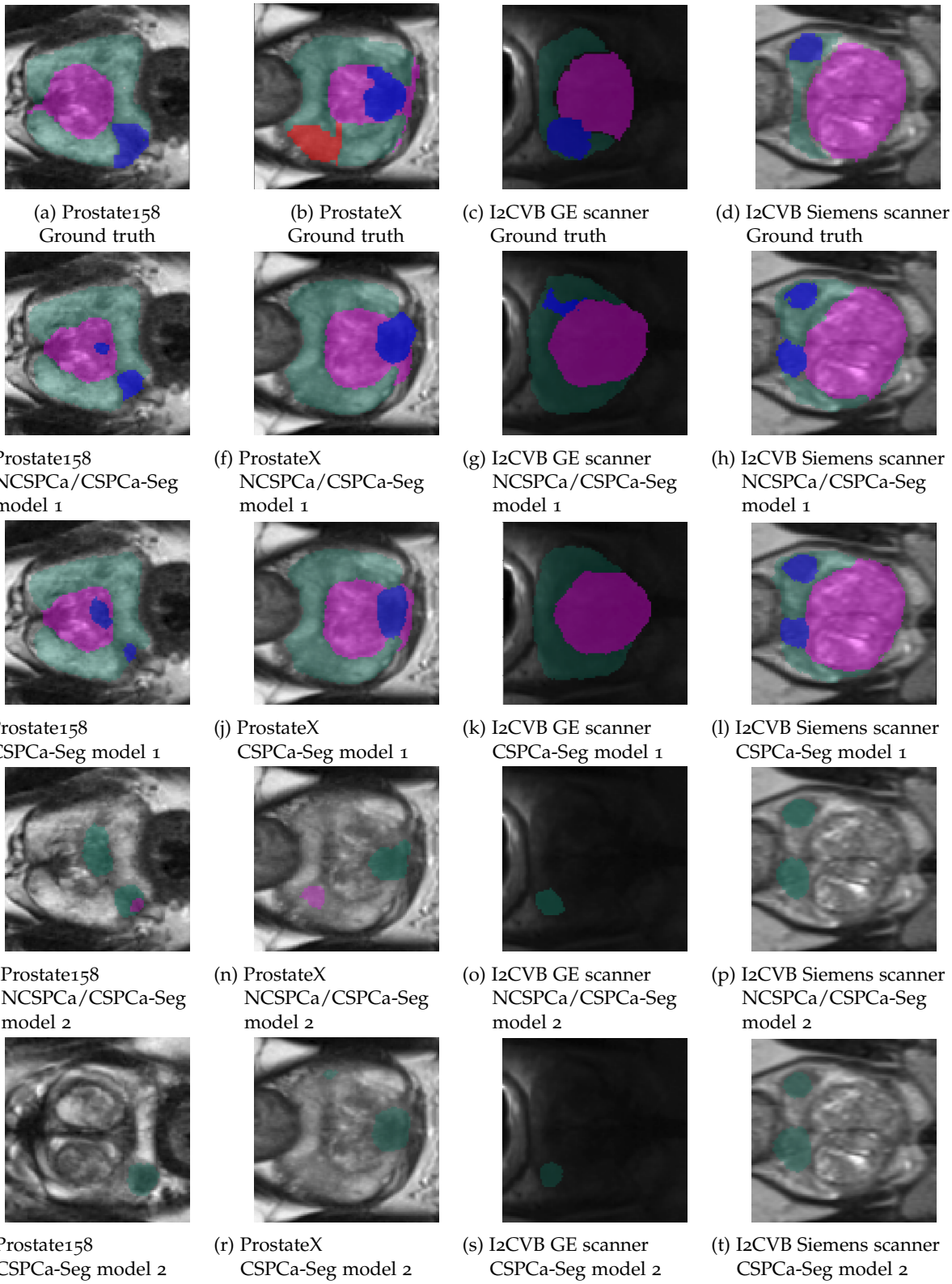
There are also some false positive predictions, such as in the image from the Prostate158 dataset and the patient from the I2CVB dataset acquired with the Siemens scanner, both in Figure 11.5, and false negatives, for example Figure 11.4k and Figure 11.5k. Note that the NCSPCa lesion in the patient from the ProstateX dataset in Figure 11.5 is only detected (but not completely segmented) by the PCa-Seg model 2.

Regarding the prostate zones, we can observe that in all cases they are correctly segmented, except for the patient from the I2CVB dataset acquired with the GE scanner in Figure 11.4, where only the PZ appears in the ground truth mask.



(a) Prostate158 Ground truth
(b) ProstateX Ground truth
(c) I2CVB GE scanner Ground truth
(d) I2CVB Siemens scanner Ground truth
(e) Prostate158 NCSPCa/CSPCa-Seg model 1
(f) ProstateX NCSPCa/CSPCa-Seg model 1
(g) I2CVB GE scanner NCSPCa/CSPCa-Seg model 1
(h) I2CVB Siemens scanner NCSPCa/CSPCa-Seg model 1
(i) Prostate158 CSPCa-Seg model 1
(j) ProstateX CSPCa-Seg model 1
(k) I2CVB GE scanner CSPCa-Seg model 1
(l) I2CVB Siemens scanner CSPCa-Seg model 1
(m) Prostate158 NCSPCa/CSPCa-Seg model 2
(n) ProstateX NCSPCa/CSPCa-Seg model 2
(o) I2CVB GE scanner NCSPCa/CSPCa-Seg model 2
(p) I2CVB Siemens scanner NCSPCa/CSPCa-Seg model 2
(q) Prostate158 CSPCa-Seg model 2
(r) ProstateX CSPCa-Seg model 2
(s) I2CVB GE scanner CSPCa-Seg model 2
(t) I2CVB Siemens scanner CSPCa-Seg model 2

Figure 11.4.: Predictions of the PCa-Seg models on patients from different datasets together with the corresponding ground truth mask. Each column corresponds to a single patient from a dataset.



(a) Prostate158 Ground truth
(b) ProstateX Ground truth
(c) I2CVB GE scanner Ground truth
(d) I2CVB Siemens scanner Ground truth
(e) Prostate158 NCSPCa/CSPCa-Seg model 1
(f) ProstateX NCSPCa/CSPCa-Seg model 1
(g) I2CVB GE scanner NCSPCa/CSPCa-Seg model 1
(h) I2CVB Siemens scanner NCSPCa/CSPCa-Seg model 1
(i) Prostate158 CSPCa-Seg model 1
(j) ProstateX CSPCa-Seg model 1
(k) I2CVB GE scanner CSPCa-Seg model 1
(l) I2CVB Siemens scanner CSPCa-Seg model 1
(m) Prostate158 NCSPCa/CSPCa-Seg model 2
(n) ProstateX NCSPCa/CSPCa-Seg model 2
(o) I2CVB GE scanner NCSPCa/CSPCa-Seg model 2
(p) I2CVB Siemens scanner NCSPCa/CSPCa-Seg model 2
(q) Prostate158 CSPCa-Seg model 2
(r) ProstateX CSPCa-Seg model 2
(s) I2CVB GE scanner CSPCa-Seg model 2
(t) I2CVB Siemens scanner CSPCa-Seg model 2

Figure 11.5.: Predictions of the PCa-Seg models on patients from different datasets together with the corresponding ground truth mask. Each column corresponds to a single patient from a dataset.

11.3. Discussion

In this work, we trained and evaluated various U-Resnet models for the segmentation of clinically significant and not clinically significant PCa lesions in prostate T2W MRI of different public datasets.

Different outputs can be measured for our system. Firstly, the P-Seg and Z-Seg models, which were developed with the sole purpose of automatically generating zonal information to be included in the PCa-Seg models, achieved state-of-the-art results. The higher DSC was obtained on the ProstateX dataset, with a value of 0.917, 0.867 and 0.773 for the segmentation of the whole prostate gland, the central gland and the peripheral zone, respectively. Regarding the zones test set, where data from the NCI-ISB challenge and the Medical Segmentation Decathlon is included, our models obtained a DSC of 0.913, 0.873 and 0.699 for the entire prostate, the CG and the PZ, respectively. For comparison, in Qin et al. [88] the authors trained their model on an internal dataset and the NCI-ISBI train dataset independently, and evaluated it using cross-validation, achieving a DSC of 0.908 and 0.785 for the CG and PZ in their internal dataset, and a DSC of 0.901 and 0.806 in the NCI-ISBI dataset. In Mehta et al. [11], during ten-fold cross-validation on the ProstateX dataset, AutoProstate obtained a DSC of 0.86 and 0.78 for the CG and PZ, respectively. Pellicer-Valero et al. [10] reported a DSC of 0.941/0.894 for the segmentation of the whole prostate, 0.935/0.860 for the central gland, and 0.866/0.690 for the peripheral zone on a private test dataset/the NCI-ISBI train dataset (the latter used only as external evaluation, not for training). Finally, Adams et al. [12] trained their model (also a U-Resnet) on the Prostate158 dataset and tested it externally on the Medical Segmentation Decathlon and ProstateX datasets, obtaining a DSC of 0.82/0.86 for the central gland and 0.64/0.71 for the peripheral zone. We have to consider that the last two studies did not use the public datasets for the training of their models, they used them entirely only for testing, but we used a part of them for training and only a small portion for testing.

The worse results obtained for the segmentation of the prostate zones on the two subsets of the I2CVB dataset may be caused by a bad quality of the images from this dataset. It is also possible that the few images included in the test set are more difficult to segment, as it is the case in Figure 11.4c.

Secondly, the PCa-Seg models produce segmentation masks for both CSPCa and NCSPCa lesions or only for CSPCa lesions, depending on each case. Regarding NCSPCa lesions, it seems that segmenting this type of lesions on T2W MRI images is a more difficult task for the models to learn. The reason for that may be that NCPCa lesions are difficult to detect and differentiate from CSPCa lesions using only T2W MRI images. In fact, to the best of our knowledge, no previous study has tried to train a deep learning model to segment not clinically significant PCa lesions by using only T2-weighted MRI. Moreover, approximately only half of the patients in the employed training dataset contain the segmentation mask for this type of lesions (at most 157 patients, corresponding to the training data from the ProstateX dataset), while the other patients may have not aggressive lesions but their segmentation masks are not included. Therefore, the models do not have enough information to learn to detect NCSPCa lesions and become confused with images where these lesions are present but they do not contain the associated segmentation mask. Indeed, our models detected indolent PCa lesions in patients from all datasets, so some of these false positive predictions may correspond to actual NCSPCa lesions. Nevertheless, CSPCa/NCSPCa-Seg model 2 achieved a DSC of 0.1178 for the segmentation of non-aggressive PCa lesions on the ProstateX dataset, a value that is comparable with the 0.1095 DSC obtained by the same

model for the segmentation of CSPCa lesions on this test set.

As for clinically significant lesions, the best value of DSC obtained during three-fold cross-validation was 0.16. By observing Figure 11.1 and Figure 11.3, we come to the conclusion that the differences in results between models are not significant and not clearly interpretable. Thus, we can assert that the integration of zonal information by either of the two tested ways into the PCa-Seg models has the same effect on the segmentation performance of CSPCa lesions. However, we can affirm by observing the difference between the values of the Dice and Dice enhanced metrics that PCa-Seg model 1 produces fewer false positive predictions than PCa-Seg model 2.

Although our models were trained on different datasets, they still behave differently on each of them, which is not desirable, but probably unavoidable. On the test data from the I2CVB dataset acquired with the Siemens scanner the best DSC value achieved was 0.496 and the best AUC 0.737, while on the test images from the same dataset but acquired with the GE scanner the highest DSC obtained was 0.223 and the highest AUC 0.724. Concerning the Prostate158 test set, the best results were a DSC of 0.1679 and AUC of 0.771, and for the ProstateX dataset, the highest values were 0.116 for the DSC and 0.897 for the AUC.

Note that better results could be reached by increasing the number of epochs for training. However, this would require much more training time and computing resources in the cross-validation scheme.

One of the reasons why the segmentation performance is better on the Prostate158 dataset compared with the ProstateX dataset is that only lesions with a PI-RADS score$\geq$ 4 are included in the Prostate158 dataset, while ProstateX also contains lesions with a PI-RADS score of 3. PCa lesions with a PI-RADS score$\geq$ 4 are larger and, thus, easier to detect than PI-RADS 3 lesions, leading to a better segmentation performance of the deep learning models. For the I2CVB dataset, we do not have information about the grade of aggressiveness of the included PCa lesions, but the higher values of DSC obtained on the tests sets from this dataset make us think that these lesions are larger or at least easier to detect. Since the test sets from the I2CVB dataset are too small, the good results could also be a result of the selected test sets.

A fair comparison with the segmentation performance of prostate cancer of previous CAD systems is challenging as the datasets used for training and testing, the evaluation metrics selected and the ground truth annotations vary across studies. Furthermore, we trained our models to distinguish PCa lesions on a per-voxel basis, while most previous studies have evaluated their systems on a per-lesion basis. However, we tried to overcome these limitations by testing our models on different publicly available datasets and evaluating their performance using a variety of metrics adopted in the literature.

We begin our comparison with Alkadi et al. [9]. For the segmentation of aggressive PCa lesions, they obtained an average AUC of 0.995, an accuracy of 0.894, a recall of 0.928 and an IoU of 0.679 using a leave-one-patient out cross-validation scheme on the subset from the I2CVB dataset formed by the T2W images acquired with the Siemens scanner. Using the same evaluation method on the same subset, Liu et al. [80] reported an accuracy of 0.931, a recall of 0.911, an IoU of 0.702 and a DSC of 0.793. In our case, the best results achieved by our models for the test set selected from this subset (I2CVB Siemens scanner) were an AUC of 0.737, accuracy of 0.968, recall of 0.484, IoU of 0.338 and DSC of 0.496.

For the segmentation of CSPCa lesions on images from the ProstateX dataset, AutoProstate [11] achieved a DSC of 0.39 during ten-fold cross-validation (using the ground truth labels of Cuocolo et al. [62]), De Vente et al. [77] obtained a DSC of 0.37 with five-fold cross-validation and Pellicer-Valero et al. [10] reported an average DSC over all patients on their chosen test

set (from ProstateX) for segmenting any type of PCa lesions (including benign lesions) of 0.255 when using a 0.25 probability threshold and 0.244 with a 0.5 threshold. It is important to consider that the two last studies used automatically generated ground truth labels (grown from the lesion center) which are far from being the perfect ground truth. Adams et al. [12] used the entire dataset from the ProstateX challenge (with ground truth labels by Cuocolo et al.) as an external evaluation of their model and obtained a DSC of 0.11 for the segmentation of all CSPCa lesions in the dataset, which coincides with the highest DSC value achieved by our models on the subset from the ProstateX dataset that we selected for testing. For training, they used the Prostate158 dataset and a small subset of data from the same institution (that they kept private) was considered for testing. On this test subset, they reached a DSC of 0.45 for the CSPCa lesions compared with the ground truth labels annotated by the first radiologists, which are the labels that we used in our work.

It is worth mentioning that, as far as we know, this is the first study that makes use of the data from the I2CVB dataset obtained with the 1.5T GE scanner to train and evaluate a deep learning model for the segmentation of cancer tumors. These images are more difficult to segment since they are darker and of worse quality than the images from the other datasets used in this work.

We can see that the segmentation performance of PCa lesions achieved by our models is in general below the state-of-the-art. Nevertheless, we have to consider that all the aforementioned studies used different MRI modalities (except for Alkadi et al. [9]) to train their models while we only used T2W MRI images. The inclusion of different MRI modalities provides additional information on prostate tissues, especially in the lesion areas, that improves the segmentation performance. The use of a single MRI modality might ignore the different forms of mutual information, preventing the model from achieving clear segmentation performance. As we commented in section 9.3, the use of the T2W modality alone to detect cancer lesions has limitations that can be overcome with the aid of other modalities. Therefore, it is reasonable to expect lower results when only the T2W MRI modality is used to train deep learning networks to segment PCa lesions. Moreover, we utilized a quite simple model, namely the U-Resnet architecture, while most CAD systems in the literature have used more complex models, e.g. with attention mechanisms, for the same task.

Another aspect to take into account is that most of the above mentioned systems in the literature were trained and evaluated on the same unique dataset, often homogeneous, with data acquired with the same MRI scanner, the same parameters and annotated by the same radiologist. As a result, the optimistic performance reported by those systems may be dataset-specific, with no guarantee of good generalization on test data sampled from other distributions. In contrast, we trained and tested our models in a heterogeneous dataset, which ensures a good generalization ability to data from different institutions and makes the system suitable for clinical use, but at the same time it can hinder learning and worsen the segmentation performance. To the best of our knowledge, we are the first to use all the public databases that contain segmentation masks of PCa lesions to train and validate our models.

The segmentation of PCa lesions is undoubtedly a difficult task even for human experts. Bhayana et al. [89] compared the interrater agreement between radiologists with various years of experience using Cohen's kappa[3] and they found that for lesions with a PI-RADS score of 3, using PI-RADS v2.1., there was an agreement of only $\kappa = 0.17$. The best agreement was achieved for lesions with a PI-RADS ≥ 4 with $\kappa = 0.63$.

Some reasons limiting the improvement of the segmentation performance achievable by

[3]See footnote 3.

deep learning models are the great variability of the shape and size of PCa lesions among patients, the difficulty of retaining feature information from small target areas in MRI images during the downsampling process, the interference of irrelevant background noise, and variations in the ground truth definitions (e.g. comparison to whole-mount histopathology, targeted biopsy, or use of radiologist reports).

Part V.

Conclusion

12. Conclusion and Future Work

12.1. Future work

Our work still has a lot of room for improvement. First of all, the addition of different MRI modalities should be studied. As we already commented, the use of several MRI modalities, instead of a single one, could improve the segmentation performance of our models. Therefore, we could use different modalities to train our models and analyze the results while comparing them with the current segmentation performance.

Furthermore, other more complex architectures with attention mechanisms, such as the models introduced in Duran et al. [78], Saha et al. [76] or Liu et al. [80], could be implemented, since they have shown better results for the task of segmenting PCa lesions than in the case where no attention mechanisms are included. Generative adversarial networks may also be considered, such as in Kohl et al. [70] and Zhang et al. [71]. Not many studies have used this type of network before for this task, so it would be interesting to further investigate their application to the segmentation of PCa lesions in prostate mpMRI images.

Since the data we worked with is anisotropic, i.e. the size of the images and voxels is different in different dimensions (recall that our images have a depth of 32 but width and height of 160), we could try using different kernel sizes and strides for different dimensions in the network architecture, as the nn-Unet [79] does.

As far as we know, until now no previous studies have developed an ensemble of different network architectures to segment prostate cancer lesions in MRI images, so it could be a new line of research.

Another possible modification is to use 2D network architectures, instead of 3D, and provide as input to the network only one slice each time from the T2W MRI sequences, or employ 2D architectures with three input branches for three consecutive slices, and compare the results with the ones of our models. 2D network architectures require much fewer computational resources than 3D networks and may offer better results for anisotropic data.

With regard to the datasets, we could try training and evaluating our models for the segmentation of PCa lesions on a single dataset (for each dataset considered) and observe whether the results obtained by our models on that particular dataset improve in comparison with the results of the models trained on several heterogeneous datasets. In that case, we could assert that the relatively poor performance of our models is in part due to the heterogeneity of the training data. We should also test our models on external datasets (not used for training) to evaluate their generalization ability.

Additionally, it is clear that a larger dataset is likely to improve the segmentation performance, especially for the case of not clinically significant PCa lesions, since the only dataset publicly available that contains the segmentation masks for this type of lesions is the ProstateX (provided by Cuocolo et al. [62]). Nevertheless, due to the privacy policy, hospitals cannot publicly share information about their patients, so a large expert-annotated dataset of prostate mpMRI is difficult to obtain in practice. Therefore, current research is limited by the lack of sufficiently large and comprehensive annotated datasets.

12.2. Conclusion

To conclude this work, we can say that all the initially proposed objectives have been successfully accomplished. Indeed, we studied the mathematical fundamentals of deep learning together with the most popular models used for computer vision tasks: convolutional neural networks. We came to the conclusion that feedforward neural networks (of which convolutional neural networks are a special kind) have great expressive power, so they can be applied to learn complex functions. Additionally, they have a finite VC dimension in general, so the hypothesis class that they implement is PAC learnable. Therefore, they are a good choice to solve our problem. In particular, the U-Net architecture has been proven to achieve the best results for the task of semantic segmentation in biomedical images, so we decided to use one of its variants in our work.

Furthermore, we implemented a deep learning based software that learned to segment PCa lesions in T2W images, as well as the prostate zones. We used all the publicly available datasets that include segmentation masks for PCa lesions in MRI images to avoid the overfitting of our models to a specific dataset. To the best of our knowledge, this is the first study to do that. For the segmentation of the prostate zones, our model achieved state-of-the-art results, while the segmentation performance of the models that learn to segment PCa lesions is below the state-of-the-art, being the differences between the results obtained by the different considered models not significant. However, the differences in the results achieved on the different datasets are meaningful, because different datasets contain images and segmentation masks of different quality and the level of aggressiveness (and hence the size) of the lesions differs between datasets.

We hypothesize that the lower performance of our models for the task of segmenting PCa lesions is partly due to the simple network architecture that we used, being necessary experimentation with more complex architectures. Moreover, it is possible that the use of a single MRI modality is not sufficient to detect PCa lesions, since each modality is subject to its own limitations. The heterogeneity of the data used for training may be another reason for the poor results, but we cannot be sure unless we train and evaluate our models on a single homogeneous dataset (what we did not do because of a lack of time).

Anyways, it is well-known that the performance that any CAD system could achieve for the segmentation of prostate cancer in MRI images is upper bounded, being the main reasons the difficulty of the task itself, which is complex even for expert radiologists, and a lack of sufficiently large, comprehensive annotated and good quality datasets.

Nevertheless, with further improvements, these systems may have a major impact on patient care, helping radiologists to interpret prostate MRI, so that their performance is improved (for example avoiding missing PCa lesions), the process is speeded up and the inter-observer variability is reduced.

Bibliography

[1] Vladimir N.Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998. [Cited in page xvii, 6, 29, 35, 37, and 42]

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. http://www.vision.jhu.edu/teaching/learning/deeplearning18/assets/Cybenko-89.pdf. [Cited in page xvii and 86]

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. https://link.springer.com/content/pdf/10.1007/978-3-319-24574-4_28.pdf. [Cited in page xviii, 4, 115, and 117]

[4] Achim Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer, London, second edition, 2013. [Cited in page 6 and 11]

[5] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, 2014. [Cited in page 6, 29, 33, 43, 69, 75, and 85]

[6] Y. S. Abu-Mostafa, M. Magdon-Ismail and H.-T. Lin. *Learning from Data, a short course*. AMLBook, New York, USA, 2012. volume 4. [Cited in page 6, 29, 31, 41, 69, 70, 72, 74, and 88]

[7] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. [Cited in page 6 and 49]

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. [Cited in page 6, 49, 69, and 91]

[9] Ruba Alkadi, Fatma Taher, Ayman El-Baz, and Naoufel Werghi. A deep learning-based approach for the detection and localization of prostate cancer in T2 magnetic resonance images. *Journal of digital imaging*, 32(5):793–807, 2018. [Cited in page 6, 135, 167, and 168]

[10] Oscar J Pellicer-Valero, José L Marenco Jiménez, Victor Gonzalez-Perez, Juan Luis Casanova Ramón-Borja, Isabel Martín García, María Barrios Benito, Paula Pelechano Gómez, José Rubio-Briones, María José Rupérez, and José D Martín-Guerrero. Deep Learning for fully automatic detection, segmentation, and Gleason Grade estimation of prostate cancer in multiparametric Magnetic Resonance Images. *Scientific reports*, 12(1):1–13, 2022. [Cited in page 6, 128, 137, 166, and 167]

[11] Pritesh Mehta, Michela Antonelli, Saurabh Singh, Natalia Grondecka, Edward W Johnston, Hashim U Ahmed, Mark Emberton, Shonit Punwani, and Sébastien Ourselin. AutoProstate: Towards Automated Reporting of Prostate MRI for Prostate Cancer Assessment Using Deep Learning. *Cancers*, 13(23):6138, 2021. [Cited in page 6, 137, 166, and 167]

[12] Lisa C Adams, Marcus R Makowski, Günther Engel, Maximilian Rattunde, Felix Busch, Patrick Asbach, Stefan M Niehues, Shankeeth Vinayahalingam, Bram van Ginneken, Geert Litjens, et al. Prostate158-An expert-annotated 3T MRI dataset and algorithm for prostate cancer detection. *Computers in Biology and Medicine*, page 105817, 2022. [Cited in page 6, 118, 119, 131, 138, 166, and 168]

[13] Vijay K. Rohatgi and A. K. MD. Ehsanes Saleh. *An Introduction to Probability and Statistics*. Wiley series in probability and statistics. John Wiley & Sons, New York, second edition, 2001. [Cited in page 11]

[14] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1,1997. [Cited in page 29]

[15] Vladimir N.Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, second edition, 1999. [Cited in page 29]

[16] Mehryar Mohri, Afshin Rostamizadeh and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts, 2012. [Cited in page 29 and 41]

[17] Robert M. Gower. Convergence theorems for gradient descent, 2022. https://gowerrobert.github.io/pdf/M2_statistique_optimisation/grad_conv.pdf. [Cited in page 49 and 56]

[18] Micol Marchetti-Bowick. Lecture 6: September 12, 2013. https://www.stat.cmu.edu/~ryantibs/convexopt-F13/scribes/lec6.pdf. [Cited in page 49]

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. https://arxiv.org/pdf/1412.6980.pdf. [Cited in page 49, 63, and 64]

[20] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization. 2020. https://arxiv.org/pdf/2004.06632.pdf. [Cited in page 70 and 77]

[21] Fernando Berzal. *Redes Neuronales & Deep Learning*. Independently published, 2018. [Cited in page 77 and 91]

[22] Standford University. Cs231n: Deep learning for computer vision, Spring 2022. https://cs231n.github.io/. [Cited in page 77, 91, 115, and 116]

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer,. [Cited in page 83]

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. https://arxiv.org/pdf/1502.03167.pdf. [Cited in page 83]

[25] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993. https://www.sciencedirect.com/science/article/pii/S0893608005801315. [Cited in page 87]

[26] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017. https://arxiv.org/pdf/1709.02540.pdf. [Cited in page 87]

[27] P. L. Bartlett and W. Maass. Vapnik-chervonenkis dimension of neural nets. https://igi-web.tugraz.at/PDF/139.pdf. [Cited in page 88]

[28] Eric Baum and David Haussler. What size net gives valid generalization? *Advances in neural information processing systems*, 1, 1989. https://proceedings.neurips.cc/paper/1988/file/1d7f7abc18fcb43975065399b0d1e48e-Paper.pdf. [Cited in page 88]

[29] Akito SAKURAI. Polynomial Bounds for the VC-Dimension of Sigmoidal, Radial Basis Function, and Sigma-pi Networks. 1995. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.1761&rep=rep1&type=pdf. [Cited in page 88]

[30] Peter Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear VC dimension bounds for piecewise polynomial networks. *Advances in neural information processing systems*, 11, 1998. https://proceedings.neurips.cc/paper/1998/file/bc7316929fe1545bf0b98d114ee3ecb8-Paper.pdf. [Cited in page 88]

[31] Peter L Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *The Journal of Machine Learning Research*, 20(1):2285–2301, 2019. https://arxiv.org/pdf/1703.02930.pdf. [Cited in page 89]

[32] Armando Reyes Villena. *Análisis de Fourier, Tema 2: Convolución*. Departamento de Análisis Matemático, Universidad de Granada, 2020. [Cited in page 91]

[33] Isidro Paulino Basurto. *Convolución de sucesiones sumables*. Instituto Politécnico Nacional, Escuela Superior de Física y Matemáticas, México, January 2021. http://esfm.egormaximenko.com/analysisPaulinoBasurto_2020_convolucion_en_l1_presentacion.pdf. [Cited in page 92]

[34] Egor Maximenko. *Convolución sobre los enteros*. Instituto Politécnico Nacional, Escuela Superior de Física y Matemáticas, México, July 2015. http://esfm.egormaximenko.com/presentations/Maximenko_presentation_2015_convolution_on_integers.pdf. [Cited in page 92]

[35] Lars Hörmander. *The Analysis of Linear Partial Differential Operators I*. Grundlehren der mathematischen Wissenschaften 256. Springer-Verlag, 1998. [Cited in page 92]

[36] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018. https://link.springer.com/article/10.1007/s13244-018-0639-9. [Cited in page 98]

[37] Jefkine. Backpropagation in convolutional neural networks, 2016. https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/. [Cited in page 100]

[38] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016. https://arxiv.org/pdf/1603.07285.pdf. [Cited in page 102 and 103]

[39] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016. https://distill.pub/2016/deconv-checkerboard/?utm_source=researcher_app&utm_medium=referral&utm_campaign=RESR_MRKT_Researcher_inbound. [Cited in page 104 and 105]

[40] Akbar Karimi. Instance vs batch normalization, 2021. https://www.baeldung.com/cs/instance-vs-batch-normalization. [Cited in page 106]

[41] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016. https://arxiv.org/pdf/1607.08022.pdf. [Cited in page 106]

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. https://arxiv.org/pdf/1512.03385.pdf. [Cited in page 107]

[43] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv:2106.11342*, 2021. [Cited in page 109]

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. https://arxiv.org/pdf/1603.05027.pdf. [Cited in page 109]

[45] Abbas Ismail. Semantic segmentation, 2019. https://medium.com/hackabit/semantic-segmentation-8f2900eff5c8. [Cited in page 113]

[46] Jeremy Jordan. An overview of semantic image segmentation, 2018. https://www.jeremyjordan.me/semantic-segmentation/. [Cited in page 113]

[47] Hmrishav Bandyopadhyay. An introduction to image segmentation: Deep learning vs. traditional [+examples], 2021. https://www.v7labs.com/blog/image-segmentation-guide. [Cited in page 113 and 114]

[48] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. https://arxiv.org/pdf/1405.0312.pdf. [Cited in page 113]

[49] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016. https://link.springer.com/chapter/10.1007/978-3-319-46723-8_49. [Cited in page 117]

[50] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018. [Cited in page 118]

[51] Eric Kerfoot, James Clough, Ilkay Oksuz, Jack Lee, Andrew P King, and Julia A Schnabel. Left-ventricle quantification using residual u-net. In *International Workshop on Statistical Atlases and Computational Models of the Heart*, pages 371–380. Springer, 2018. [Cited in page 118]

[52] Guillaume Lemaıtre. *Computer-Aided Diagnosis for Prostate Cancer using Multi-Parametric Magnetic Resonance Imaging*. PhD thesis, Universitat de Girona, Girona, Spain, 2016. [Cited in page 123, 127, and 130]

[53] Young Jun Choi, Jeong Kon Kim, Namkug Kim, Kyoung Won Kim, Eugene K Choi, and Kyoung-Sik Cho. Functional MR imaging of prostate cancer. *Radiographics*, 27(1):63–75, 2007. [Cited in page 123 and 124]

[54] John E McNeal. The zonal anatomy of the prostate. *The prostate*, 2(1):35–49, 1981. [Cited in page 123]

[55] Nicolas Mottet, Roderick CN van den Bergh, Erik Briers, Thomas Van den Broeck, Marcus G Cumberbatch, Maria De Santis, Stefano Fanti, Nicola Fossati, Giorgio Gandaglia, Silke Gillessen, et al. EAU-EANM-ESTRO-ESUR-SIOG guidelines on prostate cancer—2020 update. Part 1: screening, diagnosis, and local treatment with curative intent. *European urology*, 79(2):243–262, 2021. [Cited in page 125 and 126]

[56] Hyuna Sung, Jacques Ferlay, Rebecca L Siegel, Mathieu Laversanne, Isabelle Soerjomataram, Ahmedin Jemal, and Freddie Bray. Global cancer statistics 2020: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: a cancer journal for clinicians*, 71(3):209–249, 2021. [Cited in page 125]

[57] Giorgio Brembilla, Paolo Dell'Oglio, Armando Stabile, Anna Damascelli, Lisa Brunetti, Silvia Ravelli, Giulia Cristel, Elena Schiani, Elena Venturini, Daniele Grippaldi, et al. Interreader variability in prostate MRI reporting using Prostate Imaging Reporting and Data System version 2.1. *European radiology*, 30(6):3383–3392, 2020. [Cited in page 126]

[58] Ashkan A Malayeri, Riham H El Khouli, Atif Zaheer, Michael A Jacobs, Celia P Corona-Villalobos, Ihab R Kamel, and Katarzyna J Macura. Principles and applications of diffusion-weighted imaging in cancer detection, staging, and treatment follow-up. *Radiographics*, 31(6):1773–1791, 2011. [Cited in page 128]

[59] Maryellen L Giger and Kenji Suzuki. Computer-aided diagnosis. In *Biomedical information technology*, pages 359–XXII. Elsevier, 2008. [Cited in page 128]

[60] Geert Litjens, Oscar Debats, Jelle Barentsz, Nico Karssemeijer, and Henkjan Huisman. ProstateX Challenge data. *The Cancer Imaging Archive*, 2017. [Cited in page 129]

[61] Geert Litjens, Oscar Debats, Jelle Barentsz, Nico Karssemeijer, and Henkjan Huisman. Computer-aided detection of prostate cancer in MRI. *IEEE transactions on medical imaging*, 33(5):1083–1092, 2014. [Cited in page 129]

[62] Renato Cuocolo, Arnaldo Stanzione, Anna Castaldo, Davide Raffaele De Lucia, and Massimo Imbriaco. Quality control and whole-gland, zonal and lesion annotations for the PROSTATEx challenge public dataset. *European Journal of Radiology*, 138:109647, 2021. [Cited in page 130, 137, 167, and 173]

[63] Guillaume Lemaître, Robert Martí, Jordi Freixenet, Joan C Vilanova, Paul M Walker, and Fabrice Meriaudeau. Computer-aided detection and diagnosis for prostate cancer based on mono and multi-parametric MRI: a review. *Computers in biology and medicine*, 60:8–31, 2015. [Cited in page 130]

[64] Michela Antonelli, Annika Reinke, Spyridon Bakas, Keyvan Farahani, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, Olaf Ronneberger, Ronald M Summers, et al. The medical segmentation decathlon. *Nature communications*, 13(1):1–13, 2022. [Cited in page 131]

[65] Geert Litjens, Robert Toth, Wendy van de Ven, Caroline Hoeks, Sjoerd Kerkstra, Bram van Ginneken, Graham Vincent, Gwenael Guillard, Neil Birbeck, Jindang Zhang, et al. Evaluation of prostate segmentation algorithms for MRI: the PROMISE12 challenge. *Medical image analysis*, 18(2):359–373, 2014. [Cited in page 131]

[66] Nicholas Bloch, Anant Madabhushi, Henkjan Huisman, John Freymann, Justin Kirby, Michael Grauer, Andinet Enquobahrie, Carl Jaffe, Larry Clarke, and Keyvan Farahani. NCI-ISBI 2013 challenge: automated segmentation of prostate structures. *The Cancer Imaging Archive*, 370(6):5, 2015. [Cited in page 131]

[67] Quande Liu, Qi Dou, Lequan Yu, and Pheng Ann Heng. Ms-net: Multi-site network for improving prostate segmentation with heterogeneous MRI data. *IEEE Transactions on Medical Imaging*, 2020. https://liuquande.github.io/SAML/. [Cited in page 131]

[68] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram Van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *arXiv:1902.09063*, 2019. http://medicaldecathlon.com/. [Cited in page 131]

[69] Geert Litjens, Oscar Debats, Jelle Barentsz, Nico Karssemeijer, and Henkjan Huisman. Computer-aided detection of prostate cancer in MRI. *IEEE transactions on medical imaging*, 33(5):1083–1092, 2014. [Cited in page 135]

[70] Simon Kohl, David Bonekamp, Heinz-Peter Schlemmer, Kaneschka Yaqubi, Markus Hohenfellner, Boris Hadaschik, Jan-Philipp Radtke, and Klaus Maier-Hein. Adversarial networks for the detection of aggressive prostate cancer. *arXiv:1702.08014*, 2017. [Cited in page 135 and 173]

[71] Guokai Zhang, Weigang Wang, Dinghao Yang, Jihao Luo, Pengcheng He, Yongtong Wang, Ye Luo, Binghui Zhao, and Jianwei Lu. A bi-attention adversarial network for prostate cancer segmentation. *IEEE Access*, 7:131448–131458, 2019. [Cited in page 135 and 173]

[72] Matin Hosseinzadeh, Patrick Brand, and Henkjan Huisman. Effect of adding probabilistic zonal prior in deep learning-based prostate cancer detection. *arXiv:1907.12382*, 2019. [Cited in page 136 and 145]

[73] Patrick Schelb, Simon Kohl, Jan Philipp Radtke, Manuel Wiesenfarth, Philipp Kickingereder, Sebastian Bickelhaupt, Tristan Anselm Kuder, Albrecht Stenzinger, Markus Hohenfellner, Heinz-Peter Schlemmer, et al. Classification of cancer at prostate MRI: deep learning versus clinical PI-RADS assessment. *Radiology*, 293(3):607–617, 2019. [Cited in page 136]

[74] Yizheng Chen, Lei Xing, Lequan Yu, Hilary P Bagshaw, Mark K Buyyounouski, and Bin Han. Automatic intraprostatic lesion segmentation in multiparametric magnetic resonance images with proposed multiple branch UNet. *Medical physics*, 47(12):6421–6429, 2020. [Cited in page 136]

[75] Arun Seetharaman, Indrani Bhattacharya, Leo C Chen, Christian A Kunder, Wei Shao, Simon JC Soerensen, Jeffrey B Wang, Nikola C Teslovich, Richard E Fan, Pejman Ghanouni, et al. Automated detection of aggressive and indolent prostate cancer on magnetic resonance imaging. *Medical physics*, 48(6):2960–2972, 2021. [Cited in page 136]

[76] Anindo Saha, Matin Hosseinzadeh, and Henkjan Huisman. End-to-end prostate cancer detection in bpMRI via 3D CNNs: Effects of attention mechanisms, clinical priori and decoupled false positive reduction. *Medical image analysis*, 73:102155, 2021. [Cited in page 136 and 173]

[77] Coen De Vente, Pieter Vos, Matin Hosseinzadeh, Josien Pluim, and Mitko Veta. Deep learning regression for prostate cancer detection and grading in bi-parametric MRI. *IEEE Transactions on Biomedical Engineering*, 68(2):374–383, 2020. [Cited in page 136 and 167]

[78] Audrey Duran, Gaspard Dussert, Olivier Rouvière, Tristan Jaouen, Pierre-Marc Jodoin, and Carole Lartizien. ProstAttention-Net: A deep attention model for prostate cancer segmentation by aggressiveness in MRI scans. *Medical Image Analysis*, 77:102347, 2022. [Cited in page 137 and 173]

[79] Fabian Isensee, Jens Petersen, Andre Klein, David Zimmerer, Paul F Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajitra, Sebastian Wirkert, et al. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. *arXiv:1809.10486*, 2018. [Cited in page 137, 154, and 173]

[80] Yatong Liu, Yu Zhu, Wei Wang, Bingbing Zheng, Xiangxiang Qin, and Peijun Wang. Multi-scale discriminative network for prostate cancer lesion segmentation in multiparametric MR images. *Medical Physics*, 2022. [Cited in page 138, 167, and 173]

[81] PyTorch documentation. https://pytorch.org/docs/stable/index.html. [Cited in page 139]

[82] MONAI documentation. https://monai.io/. [Cited in page 139]

[83] Segmentation models pytorch documentation. https://segmentation-modelspytorch.readthedocs.io/en/latest/. [Cited in page 140]

[84] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv:1608.03983*, 2016. [Cited in page 146]

[85] National Institute of Health and Care Excellence. NICE Prostate cancer: Diagnosis and Management. Guidelines 2019, updated in December 2021. https://www.nice.org.uk/guidance/ng131. [Cited in page 149]

[86] David MW Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv:2010.16061*, 2020. [Cited in page 152]

[87] David Hoar, Peter Q Lee, Alessandro Guida, Steven Patterson, Chris V Bowen, Jennifer Merrimen, Cheng Wang, Ricardo Rendon, Steven D Beyea, and Sharon E Clarke. Combined transfer learning and test-time augmentation improves convolutional neural network-based semantic segmentation of prostate cancer from multi-parametric MR images. *Computer Methods and Programs in Biomedicine*, 210:106375, 2021. [Cited in page 154]

[88] Xiangxiang Qin, Yu Zhu, Wei Wang, Shaojun Gui, Bingbing Zheng, and Peijun Wang. 3D multi-scale discriminative network with multi-directional edge loss for prostate zonal segmentation in bi-parametric MR images. *Neurocomputing*, 418:148–161, 2020. [Cited in page 166]

[89] Rajesh Bhayana, Aileen O'Shea, Mark A Anderson, William R Bradley, Ravi V Gottumukkala, Amirkasra Mojtahed, Theodore T Pierce, and Mukesh Harisinghani. Pi-rads versions 2 and 2.1: interobserver agreement and diagnostic performance in peripheral and transition zone lesions among six radiologists. *American Journal of Roentgenology*, 217(1):141–151, 2021. [Cited in page 168]