Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Análisis automático de radiografías torácicas por medio de redes neuronales profundas

Tesis de Licenciatura en Ciencias de la Computación

Esteban Fernandez

Director: Dr. Enzo Ferrante

Buenos Aires, 2019

RESUMEN

Las imágenes de rayos X constituyen uno de los tipos de imágenes médicas utilizados con mayor frecuencia para el diagnóstico y seguimiento de enfermedades de la caja torácica. En particular, patologías cardiacas y pulmonares tales como la cardiomegalia, neumonía, neumotórax, pueden ser diagnosticadas por este medio. Dicho diagnóstico requiere la intervención de un especialista médico o radiólogo, que interprete las imágenes e informe sobre la existencia de anomalías en las mismas. La experiencia es un factor clave en este proceso: estudios realizados sobre una población de médicos con distintos grados de experiencia y diversas especialidades han demostrado alta variabilidad en la tasa de error de diagnóstico de unos y otros [14]. Por esta razón, el desarrollo de métodos automáticos que simplifiquen la tarea y provean indicios sobre la existencia de patologías determinadas, resulta fundamental para disminuir dichas probabilidades de error.

Durante las últimas décadas, el advenimiento de nuevos métodos de aprendizaje automático ha permitido el desarrollo de herramientas computacionales de asistencia al diagnóstico (también conocidas como CAD por las siglas en inglés de Computer Aided Diagnosis) cuyo objetivo radica en detectar posibles anomalías en imágenes médicas, alertar al especialista y reducir de esta forma la cantidad de falsos negativos en la detección temprana de patologías [15] [18]. Más recientemente, la producción masiva de datos (razón por la cual se dice que transitamos la era "big data"), junto a los últimos avances en inteligencia artificial y el gran poder de cómputo paralelo alcanzado por las unidades de procesamiento gráfico (GPU), dieron lugar a la aparición de los llamados métodos de aprendizaje profundo [19]. Estos métodos alcanzaron, en cuestión de algunos años, precisión cuasi-humana en diversas tareas específicas tan diversas como la localización de objetos, el reconocimiento de dígitos numéricos [21] o la segmentación de tumores cerebrales [22]. Sin embargo, el uso de dichos métodos para el diagnóstico de patologías torácicas se encuentra aún en su etapa inicial.

A partir de la publicación de una base de datos masiva del National Institute of Health de los Estados Unidos en el año 2017 [23] que incluye más de 100.000 imágenes de rayos X diagnosticadas con diferentes patologías cardiacas y pulmonares, la comunidad internacional de análisis de imágenes médicas ha comenzado a estudiar este problema con mayor énfasis, y se han propuesto diversos métodos basados en aprendizaje profundo para resolverlo. A la luz de dichos desarrollos, en este trabajo se implementará una herramienta que permita realizar diagnóstico automático de patologías de la caja torácica, explorando algunas de las últimas técnicas propuestas en el campo del aprendizaje automático (particularmente, las redes neuronales convolucionales densas [21] y las redes de cápsulas [25]), evaluando su desempeño en la tarea propuesta e intentando dotar de mayor interpretabilidad los modelos aprendidos.

Palabras claves: CapsNet, DenseNet, ChestX, Radiografía, Torax, NIH

ABSTRACT

X-ray images are one of the most frequently used types of medical images for the diagnosis and monitoring of diseases of the thoracic cavity. In particular, cardiac and pulmonary pathologies such as cardiomegaly, pneumonia, pneumothorax, can be diagnosed by this means. This diagnosis requires the intervention of a medical specialist or radiologist, who interprets the images and reports on the existence of anomalies in them. Experience is a key factor in this process: studies conducted on a population of physicians with different degrees of experience and various specialties have shown high variability in the diagnostic error rate of some and others [14]. For this reason, the development of automatic methods that simplify the task and provide evidence on the existence of certain pathologies, is essential to reduce these error probabilities.

During the last decades, the advent of new methods of machine learning has allowed the development of computational tools for diagnosis assistance (also known as CAD for the acronym in English of Computer Aided Diagnosis) whose objective is to detect possible anomalies in medical images, alert the specialist and reduce in this way the amount of false negatives in the early detection of pathologies [15][18]. More recently, the massive production of data (which is why we say that we are going through the big data era), together with the latest advances in artificial intelligence and the great power of parallel computing achieved by the graphics processing units (GPU) , gave rise to the appearance of the so-called deep learning methods [19]. These methods achieved, in a matter of a few years, quasi-human precision in diverse specific tasks as diverse as the location of objects, the recognition of numerical digits [21] or the segmentation of brain tumors [22]. However, the use of such methods for the diagnosis of thoracic pathologies is still in its initial stage.

After the publication of a massive database of the National Institute of Health of the United States in the year 2017 [23] that includes more than 100,000 X-ray images diagnosed with different cardiac and pulmonary pathologies, the international community of Analysis of medical images has begun to study this problem with greater emphasis, and various methods based on deep learning have been proposed to solve it. In light of these developments, this work will implement a tool that allows automatic diagnosis of pathologies of the thoracic cavity, exploring some of the latest techniques proposed in the field of machine learning (particularly, dense neural networks [21] and the capsule networks [25]), assessing their performance in the proposed task and trying to provide greater interpretability of the models learned.

Keywords: CapsNet, DenseNet, ChestX, Chest, Radiograph, NIH

AGRADECIMIENTOS

Quiero agradecer a todos mis amigos: a los del barrio, los de la facultad y los que me dio la vida; por estar siempre durante todo este trayecto a lo largo de la carrera apoyándome y alimentándome siempre de buenas energías para afrontar cada reto y obstáculo que se me presentó.

Agradezco haber tenido la suerte de trabajar con Enzo Ferrante. Sin su ayuda esta tesis no se hubiera llevado a cabo y habría trabajado en otro tema que no me hubiera interesado tanto como el de la presente tesis. Gracias a él, recobré la inspiración y motivación para continuar esforzándome, sacar adelante esta tesis y llegar a obtener resultados que superaron mis expectativas. Enzo es un experto en el área, se aprende mucho de él.

Finalmente, quiero agradecer a mi familia: a mis hermanas y mis padres. Sin su amor no hubiera logrado mantener la firmeza suficiente para nunca bajar los brazos y siempre darle para adelante durante toda la carrera y el desarrollo de esta tesis. Quiero hacer mención especial a mi papá, se lo dedico especialmente a él porque le prometí que iba a conseguir mi título sin importar como golpee la vida: es suyo también, él me enseñó como llegar.

Índice general

1. INTRODUCCIÓN

Las imágenes médicas constituyen un recurso sumamente relevante para la medicina actual ya que proporcionan representaciones internas de los diferentes órganos del cuerpo humano. A partir de ellas es posible diagnosticar, evaluar y realizar seguimiento de diferentes tipos de patologías o lesiones. Los tipos de imágenes médicas más usuales son las radiografías, las resonancias magnéticas, las tomografías computadas, ecografías, entre otras. Los profesionales involucrados en la interpretación de cada tipo de imagen requieren formación específica.

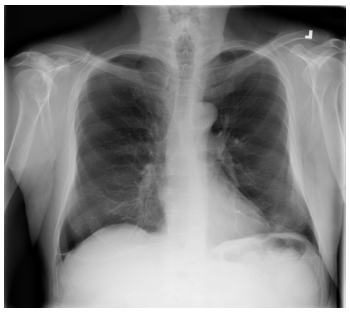

Fig. 1.1: Radiogr. torácica.

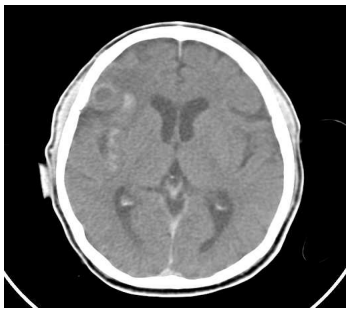

Fig. 1.2: Tomogr. cerebral.

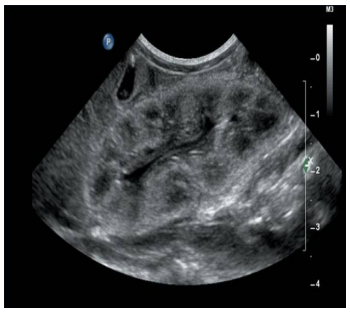

Fig. 1.3: Ecogr. abdominal.

Este es el motivo por el cual se realizan cada vez más investigaciones referidas al análisis de imágenes médicas mediante métodos computacionales capaces de automatizar un proceso de análisis que, además de ser lento y requerir personal especialmente capacitado, puede introducir errores humanos en un contexto que no admite margen de error como es el caso del diagnóstico de enfermedades. La literatura existente [27] sugiere que algunos de estos métodos automáticos alcanzan incluso a superar el desempeño de expertos en tareas acotadas.

Los análisis que se hacen a partir de estas imágenes son muy variados, desde detectar nódulos pulmonares en imágenes de rayos X hasta estimar el volumen de un órgano o medir la actividad cerebral frente a distintos estímulos. Una tarea de suma importancia dentro del análisis automático de imágenes médicas es la clasificación de las mismas, es decir, determinar a qué categoría pertenece una imagen dada. Este problema será central para esta tesis de licenciatura.

En este trabajo, nos centraremos en la tarea de clasificación de imágenes radiográficas. Es de suma importancia generar herramientas computacionales complementarias de *screening* para la comunidad médica. Contar con ellas ayudará a predecir y localizar enfermedades con una mayor eficacia. Dicho problema será abordado por medio de redes neuronales. Los últimos avances en materia de cómputo paralelo (especialmente en unidades gráficas de procesamiento o GPU) combinados con nuevos y diversos modelos de aprendizaje automático basados en redes neuronales profundas, han permitido el desarrollo de algoritmos de clasificación de imágenes con una alta tasa de acierto, impensable años atrás. El entrenamiento supervisado de este tipo de redes neuronales profundas no sólo requiere un gran número de imágenes, sino que también necesita las etiquetas corres-

pondientes a cada muestra. Es por esto que en el contexto de las imágenes médicas es necesario contar con profesionales que se encarguen de etiquetar manualmente las muestras, volviéndolas sumamente costosas.

En este contexto, se realizará un análisis comparativo de modelos capaces de resolver el problema de clasificación, utilizando una base de datos de imágenes radiográficas existente que incluye los diagnósticos médicos. El primer modelo que se estudiará es el de redes convolucionales densas en conexiones [21], que ha demostrado ser efectivo en problemas similares. En segundo lugar, se indagará sobre un nuevo tipo de redes neuronales propuestas recientemente por Geoffrey Hinton en [25] conocido como redes de cápsulas. Este tipo de redes, donde las neuronas son agrupadas en cápsulas, fue propuesto con el objetivo de mejorar el poder de representación de las redes convolucionales clásicas, especialmente en lo referido a forma, tamaño, posición y orientación de los objetos presentes en las imágenes a clasificar. Aquí, se indagará sobre su aplicación en el contexto del análisis de imágenes radiográficas.

1.1. Objetivos

El objetivo de esta tesis de licenciatura es estudiar, diseñar e implementar modelos de software basados en redes neuronales para la clasificación automática de imágenes médicas de radiografías torácicas. Para ello, se consideraran dos tipos diferentes de redes neuronales: redes neuronales convolucionales densas y redes de cápsulas.

El foco estará puesto en comprender los modelos propuestos y sus variantes, comparando su desempeño en la tarea de clasificación de imágenes de radiografía de tórax. La patología a considerar durante la experimentación será la cardiomegalia (agrandamiento del corazon), una patología cardíaca que puede ser diagnosticada utilizando este tipo de imágenes.

1.2. Estructura

Esta tesis está estructurada en un total de 6 capítulos. El capítulo 1 incluye una introducción a la problemática estudiada. En el capítulo 2, se presentan los conceptos básicos para comprender el contexto médico de aplicación de este trabajo, se discuten brevemente otras aproximaciones existentes en la literatura para el abordaje de la problemática, y se introduce el marco teórico sobre redes neuronales. El capítulo 3 plantea los modelos implementados para la tarea de clasificación de imágenes. En el capitulo 4 se incluye la descripción del soporte implementado para ejecutar las experimentaciones, validaciones y comparaciones de modelos. Luego, el capítulo 5 muestra una discusión sobre los resultados obtenidos durante las experimentaciones. Finalmente, la sección 6 incluye las conclusiones, puntos relevantes a destacar y posibles trabajos a futuro.

2. CLASIFICACIÓN DE IMÁGENES MÉDICAS

2.1. Imágenes radiográficas

La radiografía de tórax es uno de los exámenes radiológicos más accesibles para la detección y el diagnóstico de diversas enfermedades pulmonares. Este tipo de imágenes permite detectar patologías asociadas a distintos órganos como el corazón, los pulmones, las vías respiratorias, los vasos sanguíneos y los huesos de la columna y el tórax. Al igual que las tomografías computadas, las radiografías se obtienen a partir de la exposición del cuerpo a pequeñas dosis de radiación ionizante. En general, existen dos vistas posibles para la captura de una radiografía: lateral y frontal (ver Figura 2.1).

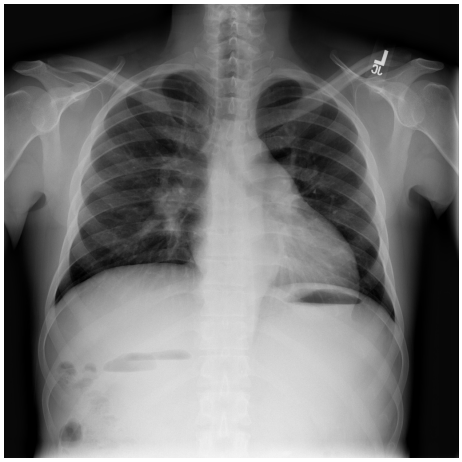

Fig. 2.1: Frontal.

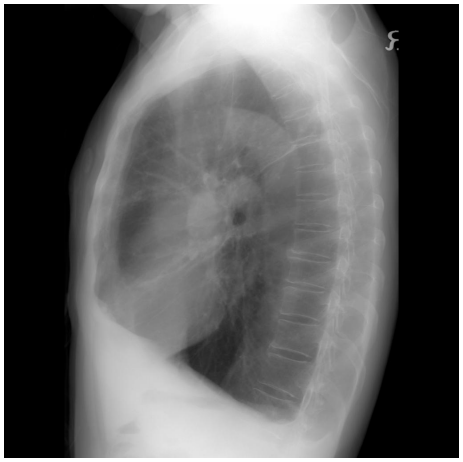

Fig. 2.2: Lateral.

Los radiólogos dependen directamente de la experiencia para ser correctos en sus análisis. A partir de estas imágenes, en conjunto con los correspondientes reportes médicos de los pacientes, es posible determinar un amplio espectro de patologías tales como la neumonía, cardiomegalia, cancer, atelectasia, fibrosis, etc.

Fig. 2.3: Nódulo.

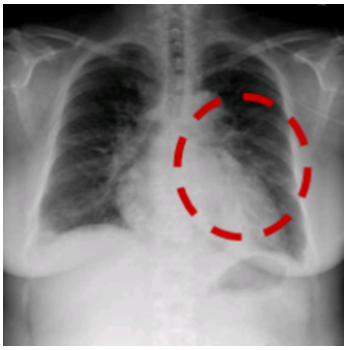

Fig. 2.4: Neumonía.

Fig. 2.5: Cardiomegalia.

Fig. 2.6: Ejemplos de patologías que pueden ser detectadas utilizando imágenes de radiografía de torax.

2.2. Aproximaciones clásicas a la clasificación de imágenes

Las imágenes radiográficas, al igual que cualquier otro tipo de imagen, pueden ser clasificadas siguiendo un criterio de interés para el problema a resolver. En el caso que se estudiará en esta tesis de licencitura, el interés radica en poder distinguir entre imágenes radiográficas de pacientes sanos y pacientes con cardiomegalia (ver Figura 2.5). Para llevar a cabo esta tarea, se utilizan algorítmos de clasificación de imágenes.

Los métodos clásicos de clasificación de imágenes poseen dos etapas principales y desacopladas: la extracción de características y la clasificación. Partiendo de un conjunto inicial de muestras, la primer etapa consiste en generar valores derivados de las mismas (*features* o *características*), que permitan distinguir cuándo una imagen corresponde a una u otra categoría. Estas características deberán ser discriminativas, informativas y no redundantes. En general, se busca que dichas características posean baja dimensionalidad (o al menos más baja que la dimensionalidad de las imágenes de entrada) pero que mantengan un poder discriminativo suficiente como para facilitar el aprendizaje y generalización de los métodos de clasificación que serán entrenados a partir de ellas.

Los métodos de clasificación mapean los datos de entrada x a etiquetas categóricas de salida y. Dentro del paradigma de aprendizaje supervisado, dado un conjunto de entrenamiento formado por pares (x,y), los métodos de clasificación son entrenados y luego utilizados para predecir la categoría asociada a nuevas muestras.

Algunos de los métodos de clasificación más utilizados en la literatura son:

- **Decision tree:** estructura que, como su nombre lo indica, tiene forma arbórea. En cada nodo se realiza la prueba de una condición sobre un atributo. Las ramas salientes de un nodo representan la dirección que se tomará en base al resultado de la prueba correspondiente. Los caminos de la raíz hasta una hoja representan reglas de clasificación.

- **Minimum distance:** este método supervisado clasifica una imagen desconocida mediante el cálculo de la distancia (por ejemplo euclídea) respecto a las observaciones conocidas (imágenes ya etiquetadas que pertenecen a la base de conocimiento). La que brinde la menor distancia definirá la etiqueta de la muestra desconocida.

- **Support Vector Machines:** Este método supervisado utiliza un hiperplano para separar las clases presentes en la distribución de observaciones. El espacio que existe entre las muestras de una clase y otra (también conocido como margen) debe ser lo más amplio posible. Los vectores que definen el hiperplano, se los conoce como *vectores soporte* o *support vectores*.

Diversos trabajos han empleado estos enfoques en el contexto de la clasificación de imágenes médicas.

En [28], los autores proponen el uso de *random forest* (composición de múltiples árboles de decisión [39]) para clasificar imágenes de rayos X de diferentes partes del cuerpo como el cráneo, la columna cervical, columna torácica, espina lumbar, manos, entre otras. Se propone el método Wavelet-Based CS-LBP para la extracción de features de manera local en diferentes zonas de las imágenes. Estas features refieren a diferentes aspectos de las texturas de las imágenes.

En [32] [33] [34] se propone el uso de vectores soporte para el diagnóstico del cáncer de pulmón, enfermedades del corazón y cáncer de mama, respectivamente. En el primer trabajo citado, el diagnóstico del cáncer de pulmón [32] se realiza sobre imágenes radiográficas. Para la extracción de features, se utiliza una segmentación de los pulmones para considerar sólo la zona abarcada por los mismos. Luego, se definen features utilizando técnicas de procesamiento de imágenes tales como el cálculo del área de la zona patológica candidata y la intensidad media de los píxeles de la misma. Para el diagnóstico de enfermedades cardíacas [33], los autores proponen la extracción de features mediante el uso de algoritmos genéticos [35]. Finalmente, para la tarea de diagnóstico del cancer de mama [34], se considera un dataset cuyas muestras se componen de vectores de 9 features, incluyendo características tales como el espesor del tumor, uniformidad del tamaño sus células, uniformidad de la forma de las mismas, entre otras.

En [37] también se realiza un estudio comparativo de diferentes métodos de clasificación para obtener una caracterización tisular cuantitativa de enfermedades hepáticas a partir de imágenes de ultrasonido. Entre los métodos propuestos por los autores, se encuentra la distancia mínima. El conjunto de features a considerar se obtiene calculando parámetros estadísticos de ciertas regiones seleccionadas de las imágenes. Este conjunto es filtrado, utilizando mediciones de correlación de parámetros [38], para poder descartar los que no son importantes y solo mantener las características que presentan mayor correlación con las diferentes patologías.

En esta tesis de licenciatura se utilizará un enfoque diferente, basado en redes neuronales profundas, donde ambas etapas (la de extracción de características y la de clasificación) formarán parte del mismo modelo.

2.3. Redes Neuronales Artificiales

Una red neuronal artificial es un modelo computacional inspirado en el funcionamiento de las células del sistema nervioso, las neuronas. Las neuronas biológicas reciben información (estímulos) de otras neuronas por medio de las dendritas. Es en su núcleo (cuerpo celular) donde se concentran dichos estímulos recibidos. Dependiendo de su intensidad, si se supera un cierto *umbral* energético propio de la neurona entonces la misma se *activa*, emitiendo un impulso de salida a través del axón. Este impulso saliente será recibido por otras neuronas. Las redes neuronales artificiales imitan este principio de funcionamiento

por medio de funciones matemáticas simples que, al componerlas, permiten crear arquitecturas de redes neuronales más complejas, capaces de aprender a representar una función arbitraria a partir de un conjunto de datos de entrenamiento.

2.3.1. Perceptrón simple

El perceptrón simple es el modelo basado en redes neuronales más básico. Formalmente, sea $x \in \mathbb{R}^n$ un vector de números reales, $f : \mathbb{R}^n \to \mathbb{R}$ será un *mapeo* de la forma:

$$f(x) = \mathrm{a}(\mathrm{w} \cdot \mathrm{x} + \mathrm{b}), \tag{2.1}$$

donde $w \in \mathbb{R}^n$ es el vector de pesos que pondera cada valor de x (estímulo proveniente a través de las dendritas), $w \cdot x$ es el producto punto entre dos vectores w y x ($\sum_n w_i x_i$ donde n es el tamaño del vector de entrada). b se conoce como *bias*, y define el umbral que debe superarse para que la neurona se active. Finalmente, *a()* es la función de activación. En un problema de clasificación binario, suele utulizarse la función Heaviside *Hs*, particularmente útil en este tipo de problemas. Dicha función se define como:

$$Hs(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0. \end{cases} \tag{2.2}$$

Si $Hs(w \cdot x + b) < 0$ entonces la clase a asignar a la muestra x será 0, de lo contrario será 1.

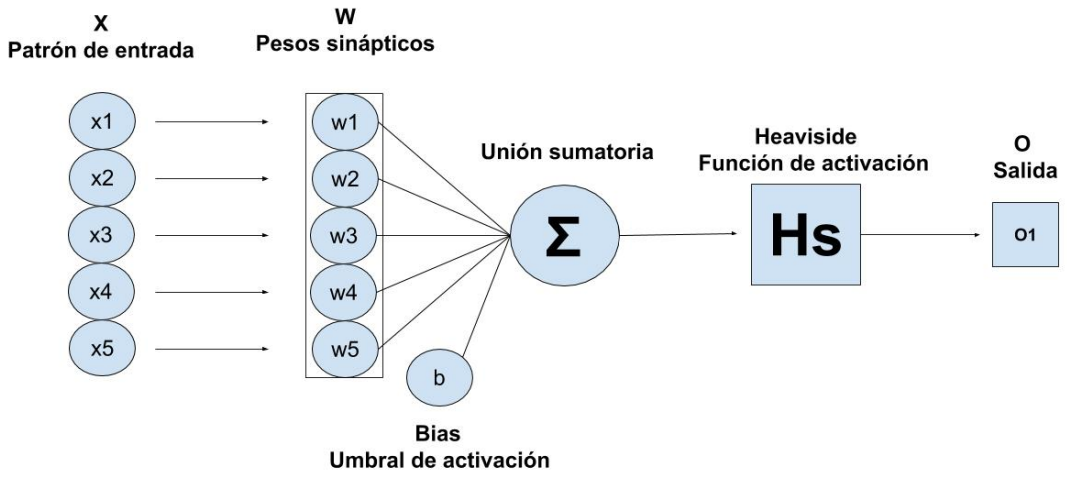

Fig. 2.7: Perceptrón simple

Siguiendo la misma idea del perceptrón simple, se extiende el problema de clasificación a más de dos clases mediante el perceptrón multiclase. La función que se implementa en este caso es $f : \mathbb{R}^n \to \mathbb{R}^m$, siendo m la cantidad de clases posibles para catalogar una muestra. Notar que el vector de entrada x no cambia de dimensiones, lo cual hace notar que ahora W es una matriz de $\mathbb{R}^{n \times m}$. La salida de $(Wx + b)$ (el producto escalar es ahora reemplazado por un producto matricial), en este caso, será un vector de tamaño m que

brindará un puntaje para cada clase. Respecto a la función de activación, ésta suele ser reemplazada por la función *Softmax*, que se define como:

$$Softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{m} e^{z_k}}. \tag{2.3}$$

Esta función de activación interpreta el vector de entrada z_j como los logaritmos de las probabilidades no normalizadas de que la muestra pertenezca efectivamente a la clase j. De esta forma, la función Softmax genera una distribución de probabilidades a partir de dicho vector de puntajes; la sumatoria de sus valores es 1. Una vez obtenido este vector, se le atribuye a la muestra la clase que mayor probabilidad haya obtenido. Es decir:

$$y = \underset{j \in 1 \ldots m}{\operatorname{argmax}} Softmax(z)_j \tag{2.4}$$

Notar que solamente una clase resultará seleccionada como ganadora para clasificar la muestra. Otra función de activación comunmente utilizada es la función *Sigmoide*, que asigna a cada salida un valor entre 0 y 1. En este caso no se genera una distribución de probabilidad, sino que la sumatoria de los valores del vector se encontrará en el rango $[0, m]$. Existen otras funciones de activación como ReLU, tangencial, etc. Todas las mencionadas son no-lineales y tienen la ventaja de que poseen primer derivada sencilla de calcular.

Para entrenar un perceptrón es necesario un conjunto de datos de entrenamiento (de tamaño N) de la forma $(x_i, y_i)_{i \in 1 \ldots N}$, donde x_i es un patrón de entrada e y_i es la clase correspondiente. Dado este conjunto, se calcula $f(x_i)$ y se computa una función de pérdida $\sum_i^N E(f(x_i), y_i)$, mediante el uso de diversas funciones de error. En este trabajo se utilizará *Cross Entropy* (entropía cruzada) y *Mean Square Error* (error cuadrático medio), definidas en la Sección 3.6. El perceptrón es entrenado utilizando el gradiente de la función de pérdida para actualizar el valor de los parámetros w y b, siguiendo el algoritmo de gradiente descendiente (ver Sección 2.3.3).

2.3.2. Perceptrón multicapa

Una limitación de los perceptrones simples es que únicamente pueden aprender a reconocer patrones que son linealmente separables en el espacio donde están definidos [2]. En este contexto surge el perceptrón multicapa, extensión del perceptrón simple donde se introducen capas de neuronas ocultas, aumentando de esta forma el poder expresivo del modelo así como la cantidad de parámetros a aprender (ver Figura 2.8).

El perceptrón multicapa propaga hacia adelante los valores del vector de entrada, desde la entrada hasta la salida del mismo (paso *forward*). Cada neurona de la red procesa la información recibida por sus entradas y produce una activación que se propaga hacia las neuronas de la capa siguiente.

Sea un perceptrón multicapa de C capas de las cuales C-2 son capas ocultas, n_c es la cantidad de neuronas de la capa $c = 1 \ldots C$. Sea $W^c = (w_{ij}^c)$ la matriz de pesos donde cada w_{ij}^c representa el peso de la conexión de la neurona i con la entrada j de la capa c para $c = 2 \ldots C$. Denotaremos a_i^c a la activación de la neurona i de la capa c. Dependiendo de la capa en la cual nos encontremos dichas activaciones se calculan de diferente manera.

- Activación de las neuronas correspondientes a la entrada a_i^1.

$$a_i^1 = x_i, i = 1, 2, \ldots, n_1, \tag{2.5}$$

donde $X = (x_1, \ldots, x_{n1})$ es el vector de entrada a la red.

- Activación de las neuronas correspondientes a la capa oculta c (a_i^c). Las neuronas ocultas procesan la información recibida aplicando la función de activación f al producto punto con sus correspondientes pesos.

$$a_i^c = f(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + b_i^c), \tag{2.6}$$

  para i = 1,2 ..., n_c y c = 2,3, ..., C-1, donde a_j^{c-1} son las activaciones de las neuronas de la capa c-1.

- Activación de las neuronas de la capa de salida (a_i^C). Al igual que en el caso anterior, la activación viene dada por la función de activación f aplicada al producto punto entre los pesos de dichas neuronas con la entrada que recibe de la capa inmediatamente anterior:

$$y_i = a_i^C = f(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + b_i^C), \tag{2.7}$$

  para i = 1,2 ..., n_C, donde $Y = (y_1, \ldots, y_{nC})$ es el vector de salida de la red.

Las cantidad de capas ocultas y de neuronas que compondrán cada una de ellas puede ser definida arbitrariamente, en función del problema a resolver. Las neuronas de las capas ocultas también utilizan una función de activación para introducir no-linearidades en la red. Una de las funciones más utilizadas es la *Rectified Linear Unit* (o ReLU) que se define como:

$$ReLU(z) = max(0, z). \tag{2.8}$$

Esta función de activación es usualmente utilizada porque mitiga el problema del gradiente que se desvanece. Este problema es una dificultad que surge al entrenar redes neuronales utilizando algoritmos basados en el gradiente descendiente y *backproapagation* (ver Sección 2.3.3). En este contexto, cada conexión (peso) de las neuronas recibe una actualización proporcional a la derivada parcial de la función de error con respecto al valor actual de dicha conexión en cada iteración del entrenamiento. El problema es que en algunos casos el gradiente, el cual depende de la función de activación utilizada, es tan pequeño que pasa desapercibido y evita que el valor de la conexión cambie. En el peor caso, esto puede impedir completamente que la red neuronal continúe su entrenamiento. Dado que el algoritmo de backpropagation (ver Sección 2.3.3) computa dichos gradientes mediante la regla de la cadena, un ejemplo de función de activación problemática es la función sigmoide, que tiende a desvanecerse para valores de entrada muy grandes o muy pequeños. El uso de la regla de la cadena en el cómputo del gradiente implica una multiplicación de n números pequeños al computar los gradientes correspondientes a las capas más cercanas a la entrada en una red de *n-capas*. El gradiente se decrementa exponencialmente con n, derivando en un entrenamiento muy lento de estas capas frontales.

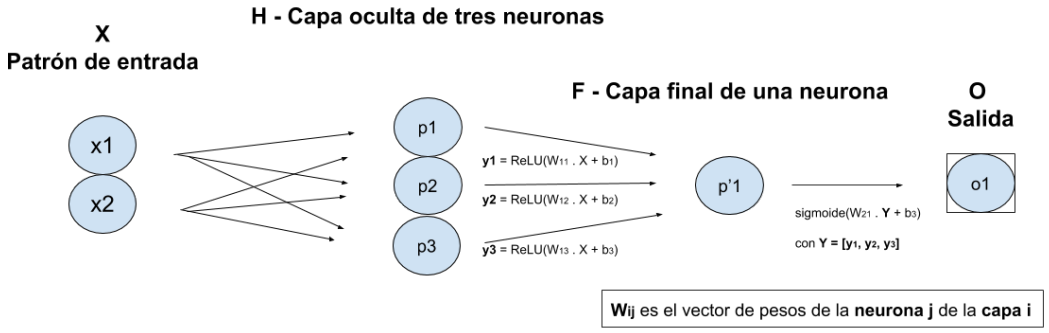

Fig. 2.8: Perceptrón multicapa de tres capas

Hay dos maneras sencillas de implementar un clasificador de imágenes mediante el uso de perceptrones. Una es extrayendo features de las imágenes con métodos definidos ad-hoc para esa tarea, generando así un vector representativo para cada muestra. En ese caso, la eficiencia del modelo dependerá no sólo de la selección apropiada de parámetros sino de la cantidad y calidad de las features extraídas.

La segunda manera es utilizar las imágenes completas como entrada para la red, es decir, transformándolas a vectores de una única dimensión. En este caso, la desventaja radica en que se pierden las referencias espaciales intra-imagen de las características que hacen a la clasificación, además de generar vectores de grandes dimensiones (tantas como píxeles en la imagen). Dichos vectores de entrada de alta dimensión producen modelos de mayor complejidad, dado que los perceptrones utilizan capas totalmente conectadas (es decir, cada neurona de una capa está conectada a todas las neuronas de la capa anterior, definiendo un peso sináptico a aprender para cada conexión). Las redes neuronales convolucionales (CNNs) que serán presentadas en la Sección 2.4, implementan mecanismos que permiten reducir la complejidad del modelo y se adaptan naturalmente a la estructura regular de las imágenes.

2.3.3. Gradiente Descendiente

En el contexto del entrenamiento de redes neuronales se busca minimizar una función de error E cuyo valor depende de los parámetros W de la red, los datos de entrada x_i y la correspondientes etiqueta asociada a cada uno de ellos y_i. Notar que para simplificar la notación, escribiremos $E(W)$ en lugar de $E(W, x_i, y_i)$ para referirnos a la función de error evaluada. Estos parámetros W son los pesos sinápticos de las neuronas como así también los *bias* asociados a estas.

Luego, podemos escribir $E(W)$ para indicar que el valor del error que comete la red neuronal depende de los parámetros mencionados. El objetivo es encontrar W^*, mínimo global de la función E.

Generalmente la función de error es una función no convexa, por lo que no disponemos de algoritmos que garanticen encontrar su mínimo global. La idea es entonces realizar una búsqueda a través del espacio de parámetros para que, idealmente, se aproxime de forma iterada a dicho mínimo global. El método utilizado para llevar a cabo esta búsqueda es el método del gradiente descendiente, que consiste en actualizar los pesos de la red siguiendo

la dirección opuesta al gradiente de la función de pérdida.

Se comienza con un conjunto inicial de parámetros (usualmente elegidos al azar) y a continuación se genera un nuevo vector de parámetros, buscando reducir la función de error. Este proceso se repite hasta haber reducido el error por debajo de un cierto umbral definido o cuando se cumpla cierta condición específica de convergencia.

Si E cumple con las condiciones de derivabilidad, es posible computar el *gradiente* ∇E de la función E respecto a cada peso de la red y usarlo para guiar el proceso de optimización. Dicho vector se compone de las derivadas parciales $\frac{\partial E}{\partial w_{ij}}$.

Denotaremos entonces:

- W_t los parámetros del modelo en la iteración t.
- $E = E(W_t)$, es el valor del error en el paso t-ésimo de la iteración.
- $g_t = \nabla\ E(W_t)$, es el valor del gradiente de la función de error en el paso t-ésimo de la iteración.

Gradiente descendiente es uno de los algoritmos de optimización más simple y extendido. Sólo hace uso del vector gradiente y por esto se dice que es un método de primer orden.

Para actualizar W_{t+1} a partir de W_t, se traslada este punto en la dirección de entrenamiento $d_t = -g_t$, justamente en la dirección contraria al gradiente, hacia el mínimo.

$$W_{t+1} = W_t - g_t v \tag{2.9}$$

donde v se denomina *tasa de entrenamiento* (o learning rate). Puede fijarse a priori o calcularse mediante un proceso de búsqueda de grilla (usualmente combinado con validación cruzada). Una implementación muy utilizada es la provista por el optimizador Adam [43] el cual utilizaremos en posteriores experimentaciones. Dicho optimizador define un learning rate particular para cada parámetro a ajustar. De esta manera se acelera el proceso de convergencia y se simplifica la búsqueda del valor óptimo para v.

Existen diferentes variaciones del algoritmo de Descenso por Gradiente, una usual es la versión estocástica. Este método calcula una actualización de los parámetros entrenables por cada muestra del conjunto de entrenamiento durante una misma época: esta es la principal diferencia con el método clásico, el cual requiere evaluar todas las muestras en su totalidad antes de realizar un único paso de actualización.

Para realizar el cálculo del gradiente se utiliza el algoritmo *backpropagation*. Este método realiza una propagación hacia atrás del error calculado a la salida de la red neuronal, utilizando la regla de la cadena para calcular los gradientes de cada capa. Una descripción completa del método queda fuera del alcance de esta tesis. El lector interesado puede referirse a [40].

2.4. Redes Neuronales Convolucionales

Este tipo de redes se entrenan siguiendo la misma estrategia que el perceptrón multicapa, conservando varios aspectos como la manera en la cual se ajustan los parámetros del modelo, mediante Gradiente Descendente y *backpropagation* (ver Sección 2.3.3), con el fin de minimizar la función de error utilizada. Las redes neuronales convolucionales

(CNN) son un tipo específico de red neuronal diseñado para emular el comportamiento de la corteza visual primaria de los mamíferos.

Como se mencionó en la Sección 2.3.2, los perceptrones multicapa pueden ser utilizados para el análisis de imágenes. Sin embargo, debido a la conectividad total de las neuronas entre capas, al aumentar la cantidad de capas o la dimensión de la entrada, la cantidad de parámetros de la red alcanza una complejidad en ocasiones inmanejable.

Otra limitación de los perceptrones multicapa es que no tienen en cuenta la estructura espacial de los datos: los píxeles de la imagen que están alejados entre sí pueden ser considerados cercanos, implicando una pérdida de referencias espaciales entre estos. Además, la conectividad completa entre neuronas no resulta adecuada para fines tales como el reconocimiento de imágenes que están determinadas por patrones de entrada que se definen en base a referencias espaciales entre píxeles.

En contraposición, los modelos de redes neuronales convolucionales aprovechan la localización espacial de los píxeles, presente en la imagen. A su vez, tienen las siguientes características:

- Volúmenes 3D de neuronas. Las capas de una CNN tienen neuronas dispuestas en 3 dimensiones: ancho, altura y profundidad. Las neuronas dentro de una capa están conectadas sólo a una pequeña región de la capa anterior, llamada *campo receptivo*. Dicho campo receptivo es definido por un *kernel convolutivo*, el cual se extiende a lo largo y ancho de la entrada, de manera restringida.

- Conectividad local: Cuando se trata de entradas de alta dimensión, como imágenes, no es práctico conectar neuronas a todas las neuronas en el volumen anterior porque dicha arquitectura de red resulta muy compleja. Las redes convolucionales explotan la correlación espacial local al imponer un patrón de conectividad local disperso entre las neuronas de las capas adyacentes: cada neurona está conectada solo a una pequeña región del volumen de entrada.

  El alcance de esta conectividad es un hiperparámetro llamado campo receptivo de la neurona. Las conexiones son locales en el espacio (a lo largo de ancho y alto), pero siempre se extienden a lo largo de toda la profundidad del volumen de entrada. Dicha arquitectura garantiza que los filtros aprendidos produzcan la respuesta más fuerte a un patrón de entrada localizado espacialmente. El apilamiento de muchas de estas capas en este tipo de redes conduce a una amplificación del campo receptivo de las neuronas que se encuentran en las últimas capas de la red (es decir, responden a una región más grande de píxeles). De esta manera, la red crea primero representaciones de pequeñas partes de la entrada y luego, a partir de ellas, ensambla representaciones de áreas más grandes y abstractas.

- Pesos compartidos: en las CNN, un mismo kernel convolutivo es utilizado sobre todo el campo visual de la entrada. Eso implica que las neuronas de una capa convolucional comparten la misma parametrización (vectores de pesos y bias) y generan un mapa de características. Por esto, todas las neuronas en una capa convolucional dada responden a la misma característica: considerando esto se reduce la cantidad de parámetros a entrenar ya que no es necesario tener un kernel por neurona de salida, únicamente se entrena un único kernel. Si un kernel es útil para reconocer una característica en una posición espacial, entonces también será útil para reconocerla en otras posiciones. Así, es posible detectar características, independientemente

de su posición en el campo visual. Esta propiedad se conoce como *invarianza a la traslación*. Notar que un misma capa convolucional puede estar dotada de varios kernels convolucionales. En este caso, cada kernel producirá un mapa de características diferente.

En conjunto, estas propiedades permiten que las CNN logren un mejor desempeño en problemas de visión computacional. Utilizar pesos compartidos, reduce drásticamente el número de parámetros a aprender, lo cual reduce los requisitos de memoria para ejecutar la red y permite el entrenamiento de redes más grandes y potentes.

La profundidad de una CNN está dada por la cantidad de capas ocultas. Existen diferentes tipos de capas, las cuales serán definidas a continuación, junto con algunos otros conceptos relevantes para el entrenamiento de dichas redes.

Capa convolucional

Como se ha mencionado, una de las particularidades de este tipo de redes es que se adaptan naturalmente a la estructura regular de las imágenes. Para lograr esto, cada neurona obtiene información de un campo receptivo restringido.

Los parámetros entrenables de este tipo de capa están en el conjunto de kernels convolutivos (también llamados filtros convolucionales), los cuales definen el *campo receptivo* de las neuronas a lo largo y ancho de la entrada, extendiéndose completamente en la profundidad del volumen de la misma.

Durante el paso *forward* (mencionado en la Sección 2.3.2), los kernels convolutivos serán desplazados a lo ancho y largo de la entrada, es decir, el kernel se desplazará y activará mediante el cálculo del producto punto y la posterior aplicación de la función de activación elegida (por ejemplo, la función ReLU). De esta manera se genera un mapa de activaciones de dos dimensiones (también conocido como *mapa de características*). A continuación se formaliza el funcionamiento de una capa convolucional:

Una imagen, en lo que respecta a la visión por computadora, es una matriz de valores de intensidad. Por ejemplo, para una imagen color, son necesarias tres matrices de intensidades, relacionadas con los canales de color rojo, verde y azul respectivamente (RGB). Un canal es un mapa I, definido en una región Ω de superficie bidimensional, cuyos valores son números reales positivos. Así, el canal I se define como:

$$I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}_+; (i, j) \to I_{i,j} \tag{2.10}$$

de esta manera, un canal de la imagen puede representarse mediante una matriz de tamaño $n_1 \times n_2$.

Supongamos el caso en que la imagen tiene un sólo canal I. La *convolución bidimensional discreta* se puede describir matemáticamente como sigue. Dado el filtro $K \in \mathbb{R}^{2h_1+1 \times 2h_2+1}$, la convolución discreta de la imagen I con el filtro K viene dada por

$$(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} * I_{r-u,s-v} \tag{2.11}$$

donde el filtro K es una matriz de la forma:

$$K = \begin{bmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{bmatrix} \tag{2.12}$$

Por lo tanto, el mapa de características resultante proporciona información sobre qué tan bien el *kernel* se ajusta a cada área local restringida de la entrada que se está analizando. Notar que si la imagen o mapa de caracteristicas de entrada a la red convolucional posee más de un canal, el kernel convolutivo poseerá tres dimensiones, donde la última dimensión corresponde a la cantidad de canales en la entrada.

Cada *kernel* toma entonces un volumen de mapas de características como entrada y genera un nuevo mapa de características, aplicando además una función de activación no lineal a cada elemento en la salida. Al implementar la convolución definida en la ecuación 2.12, se suele incorporar un término adicional de bias B, que consiste en un único escalar asociado a cada kernel convolutivo.

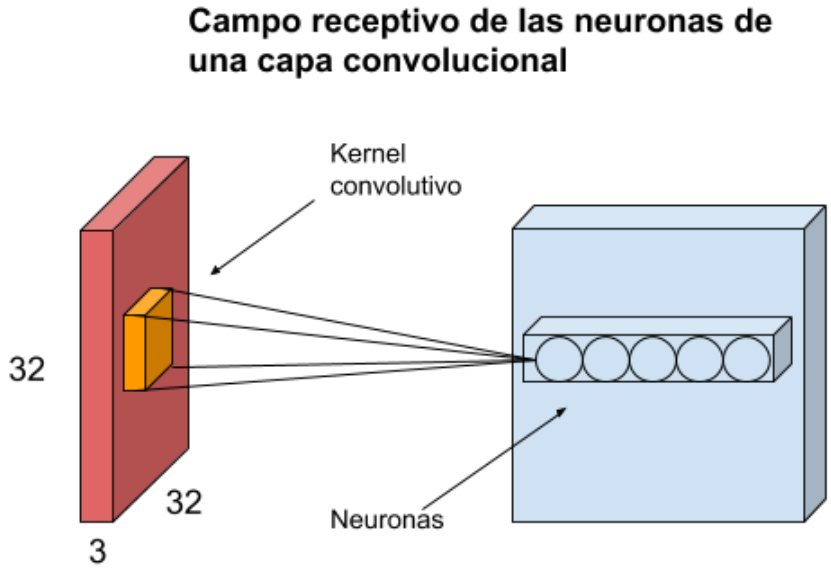

Fig. 2.9: Cada kernel convolutivo genera un mapa de características que luego es sometido a una función de activación (por ejemplo, la función ReLu).

El tamaño de los *kernels* de una capa es definido por el parámetro K'. Luego, existen otros parámetros a configurar tales como el *padding* y el *stride*. El concepto de *padding*, está relacionado a la idea de agregar P píxeles alrededor de la imagen, permitiendo a un kernel convolutivo centrarse en bordes y puntas de la imagen. De esta forma, es posible alterar el tamaño de la salida de una capa convolucional, y mantenerlo igual al tamaño de la entrada.

El parámetro *stride*, determina el paso (cantidad de pixeles, S) que es desplazado el kernel al realizar la convolución. Dependiendo de cómo combinemos dichos parámetros, obtendremos diferentes tamaños de *mapas de características* en lo que refiere al alto y ancho de los mismos. Para calcular el tamaño del volumen de salida O dado un mapa de características de entrada I, se utiliza la siguiente fórmula:

$$O = \frac{I - K' + 2P}{S} + 1 \tag{2.13}$$

El entrenamiento derivará en un conjunto de kernels que se activarán en mayor grado cuando detecten sus correspondientes features aprendidas. Ver ejemplos (ver Figura 2.10).

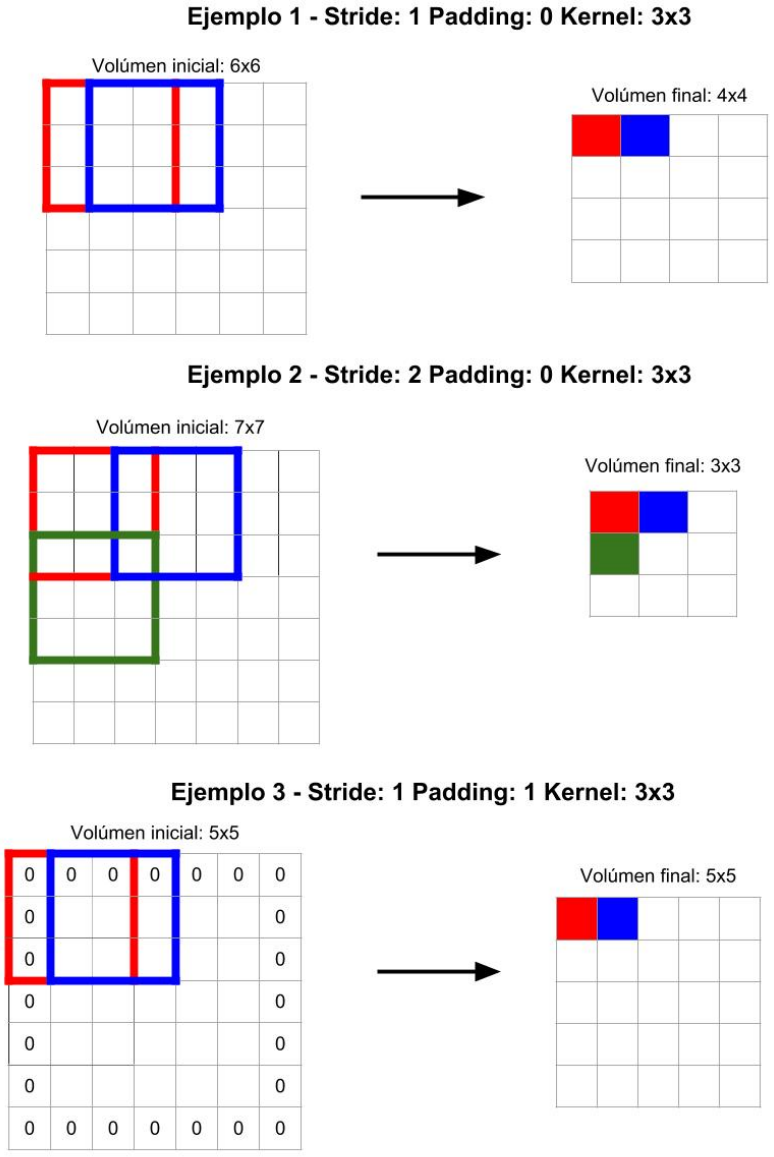

Fig. 2.10: Ejemplo de kernels convolutivos aplicados con distintos parámetros de stride, padding y tamaño del kernel

Capa de pooling

Las capas de pooling permiten reducir la resolución de los datos a medida que estos fluyen por la red, utilizando operaciones de agrupamiento. De esta forma, particionan la entrada en un conjunto de regiones (usualmente no se solapan, y son definidas por un *kernel*) y, por cada uno de ellas, se obtiene un valor representativo el cual depende del tipo de pooling que se utilice. Hay diferentes funciones para implementar este tipo de capa, generalmente se utiliza *max-pooling* (ver Figura 2.11) o *average pooling* (ver Figura 2.12).

Este tipo de capa se coloca, en general, después de la capa convolucional. No afecta a la dimensión de profundidad del volumen. La operación realizada por esta capa también se conoce como reducción de muestreo, ya que la reducción de tamaño implica una pérdida de información. Sin embargo, una pérdida de este tipo puede ser beneficiosa para la red por dos razones:

- La disminución de la resolución conduce a una menor carga de cálculos computacionales por parte de las capas subsiguientes de la red neuronal.
- Introduce mayor invarianza respecto a la posición de los objetos de interés en la imagen.

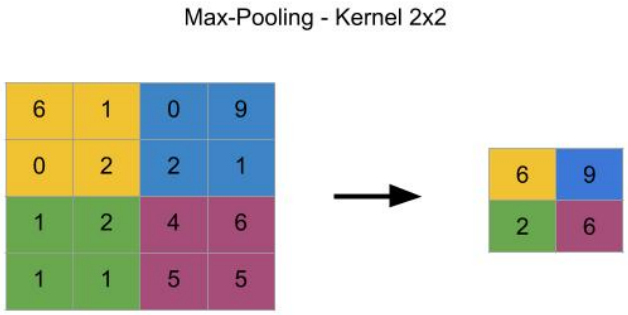

Fig. 2.11: Ejemplo de kernel max-pooling, aplicado para reducir la resolución de la entrada tomando el máximo del área abarcada por el kernel

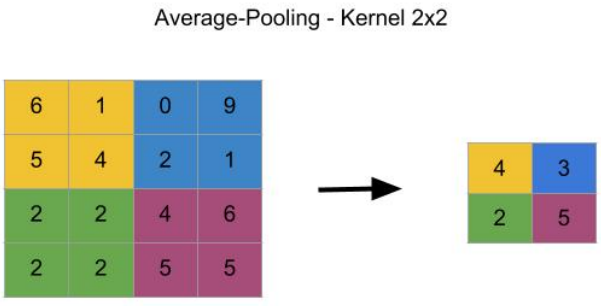

Fig. 2.12: Ejemplo de kernel average pooling, el cual toma un promedio del área abarcada por el kernel

Batch Normalization

Es una técnica utilizada para normalizar las activaciones de las capas intermedias de una red neuronal. Suele acelerar el proceso de entrenamiento porque, de no utilizarse, cada capa deberá aprender a adaptarse a una nueva distribución en los valores de entrada (problema conocido como cambio de covarianza o *covarience shift* en inglés).

Luego, para mejorar la velocidad, el rendimiento y la estabilidad durante el entrenamiento de las redes neuronales, se normaliza la salida de la capa de activación previa a la actual restando la media del batch y dividiendo por su desviación estandar.

Introduciendo no linealidad

Como ya se ha mencionado en la Sección 2.8, ReLU es una operación *element-wise* (aplicada píxel a píxel) y reemplaza todos los valores de los píxeles negativos del *mapa de características* por cero. En este contexto, el propósito de ReLU es introducir no linealidad a la red convolucional y ayudar a combatir el problema del gradiente que se desvance, discutido en la sección Sección 2.3.2.

Capa totalmente conectada

En general, las arquitecturas de redes neuronales convolucionales utilizadas para realizar clasificación de imágenes, incluyen una última capa totalmente conectada. Esta capa se corresponde con un perceptrón multicapa que utiliza una función de activación softmax o sigmoide en la capa de salida para clasificar una imagen, tal y como se introdujo en la Sección 2.3.2.

Cuando una imagen es procesada por todas las capas convolucionales y de pooling, se obtiene un vector de *features* extraídas automáticamente a partir de la entrada. Eso es sumamente importante ya que, al utilizar los métodos clásicos de la literatura de clasificación de imágenes, era necesario extraer las features de manera ad-hoc. En el caso de las CNNs, dichas características son aprendidas automáticamente y se suministra dicho vector de *features* al perceptrón multicapa para obtener una clasificación de la imagen. Todo el modelo es entrenado en conjunto, por medio de la retropropagación del error de predicción, en lo que se conoce como un proceso de entrenamiento *end-to-end*.

2.4.1. Clasificación de imágenes médicas por medio de redes neuronales convolucionales

Las arquitecturas básicas de redes neuronales convolucionales destinadas a clasificación de imágenes, se componen apilando una cantidad arbitraria de capas de convolución, seguidas de la aplicación de ReLU e incorporando análisis en múltiples resoluciones mediante pooling: dicha composición, en conjunto, actúa como extractor de *features* a partir de las imágenes de entrada. Al final de esta composición se conecta un perceptrón multicapa el cual toma el vector de features extraídas y actúa como clasificador. Uno de los ejemplos clásicos de este tipo de arquitectura es LeNet, propuesto por [44].

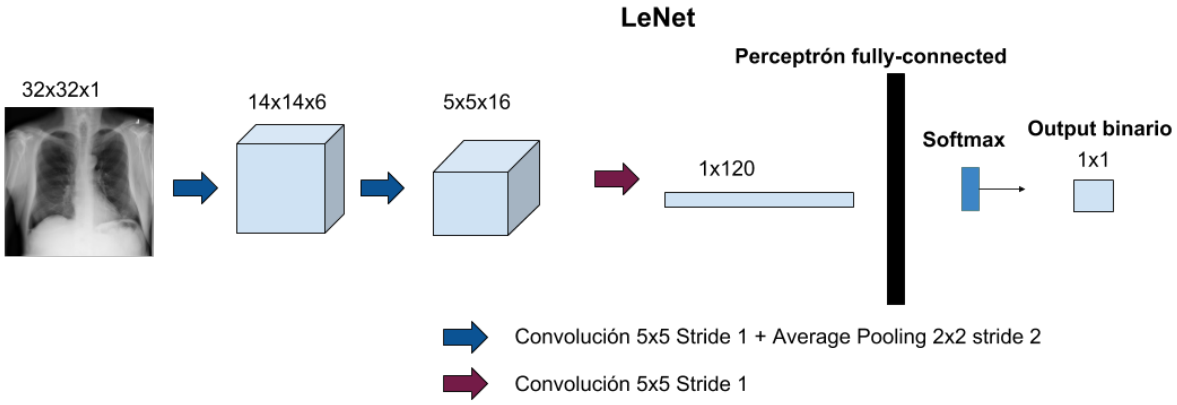

Fig. 2.13: Arquitectura básica de red neuronal profunda LeNet. [44]

El uso de este tipo de arquitecturas ha producido un gran progreso en una variedad de problemas de visión modelados por medio de aprendizaje profundo, y utilizando grandes bases de datos de imágenes anotadas como [1] [3] [4] [5]. En el contexto médico, existe una importante cantidad de aplicaciones donde este tipo de métodos ha sido aplicado con éxito. Algunos ejemplos son la detección y clasificación de ganglios linfáticos y enfermedades pulmonares intersticiales [6] [7]; detección de microbios cerebrales [16]; detección de nódulos pulmonares en las imágenes de TC [17]; segmentación automatizada del páncreas [8]; segmentación y seguimiento de imagenes de células [9]; predicción de las puntuaciones radiológicas espinales [10] y extensiones de segmentación de imágenes multimodal [11] [12].

2.5. Redes Densas

Diversos trabajos recientes [20] determinaron que el uso de una cantidad excesiva de capas en la arquitectura de una red no provee necesariamente mejores resultados debido al problema del *gradiente que se desvanece* (ver Sección 2.3.2). Este problema puede ser aliviado utilizando distintas estrategias. Una de ellas consiste en crear conexiones entre las capas cercanas a la entrada y las capas próximas a la salida de la red. De esta forma, se vuelve viable la utilización de redes más profundas capaces de propagar la información mitigando el problema del gradiente que se desvanece. Basándose en este razonamiento, surge la idea de las redes densas en conexiones (conocidas como DenseNet [21]), que hacen uso de esta estrategia agregando conexiones adicionales intra bloques de capas. Estos bloques son llamados *bloques densos*.

El nombre DenseNet puede resultar ambiguo, dado que el mismo podría dar la idea de que se utilizan capas totalmente conectadas en la red. Sin embargo, los bloque densos no incorporan capas totalmente conectadas, sino que la *densidad* del bloque viene dada por las conexiones extra que se realizan entre capas convolucionales intra-bloque sucesivas.

Una red DenseNet se divide en *bloques densos* donde cada uno está formado por capas convolucionales y, dentro de ese bloque, la salida de cada capa l es concatenada a la entrada de las capas $l+1, l+2$, etc. Es decir, los mapas de características de salida de una capa dada no solamente son conectados a la capa siguiente, sino a todas las capas consecutivas dentro del bloque, permitiendo así la reutilización de los mismos y colaborando en la

mitigación del problema del del gradiente que se desvanece. La idea es maximizar el flujo de información entre capas.

Entre bloques densos, se encuentran las *capas de transición*. Cada capa de transición incorpora un módulo de convolución cuyo kernel es de 1×1 y un módulo de pooling. Están destinados a reducir a la mitad la cantidad de mapas de características y la resolución de los mismos. Con esto, se va armando entonces un conocimiento colectivo a medida que se propagan, capa a capa, los conjuntos de mapas de características. La Figura 2.14 muestra como se unen los diferentes bloques densos, intercalados con capas de transición.

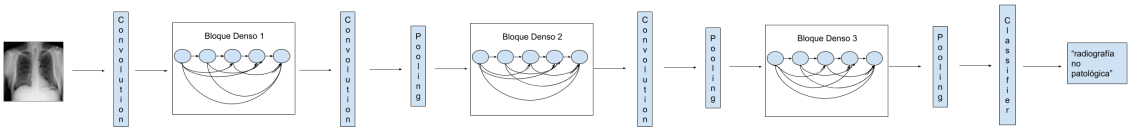

Fig. 2.14: DenseNet de tres bloques densos. Las capas entre bloques densos conforman las capas de transición, destinadas a reducir a la mitad la resolución y cantidad de mapas de características mediante convoluciones y pooling.

Formalmente, sea I_0 una imagen de entrada, $H_1(I_0)$ será el resultado de aplicar a I_0 una composición de operaciones batch normalization, convolución 3×3, ReLU y pooling. Utilizaremos la notación $I_l = H_l(I_{l-1})$ para referirnos a dicha secuencia de operaciones. Una sucesión de bloques H_l compondrán entonces un bloque denso. Una capa I_l de un bloque denso recibirá como entrada la concatenación de los mapas de características de las capas anteriores $[I_0, I_1, \ldots, I_{l-1}]$, es decir:

$$I_l = H_l([I_0, I_1, \ldots, I_{l-1}]). \tag{2.14}$$

El modelo DenseNet incorpora además una capa de convolución 1×1 entre cada par de capas consecutivas de un bloque denso, llamada *cuello de botella*, encargada de fijar la cantidad de mapas de características resultantes de la primer capa de este par, con el fin de mejorar la eficiencia computacional al aplicar la convolución de 3×3 de la segunda capa.

Finalmente, la tasa de crecimiento k de un modelo DenseNet determina la cantidad de nuevos mapas de características a agregar a la salida de cada capa de un bloque denso (ver Figura 2.15). Es decir, si la función H_l produce k mapas de características, la capa $(l+1)$ tendrá $k_0 + k * (l-1)$ mapas de características. En la Figura 2.15 puede observarse como la cantidad de feature maps de entrada aumenta al seleccionar una tasa de crecimiento $k = 4$. A su vez, existe el factor θ para cada bloque denso el cual determina la cantidad de capas dentro del mismo. La Figura 2.15 muestra el funcionamiento de un bloque denso.

2.6. Redes de Cápsulas

Para introducir a las redes de cápsulas, tenemos que mencionar primero las limitaciones que tienen las redes convolucionales clásicas. Como se mencionó en la Sección 2.4, los kernels aprenden diferentes tipos de características que pueden ser de menor o mayor complejidad (desde lineas o contornos hasta objetos o escenas completas). Dichas entidades pueden hallarse en cualquier posición espacial de la entrada y pueden ser reconocidas

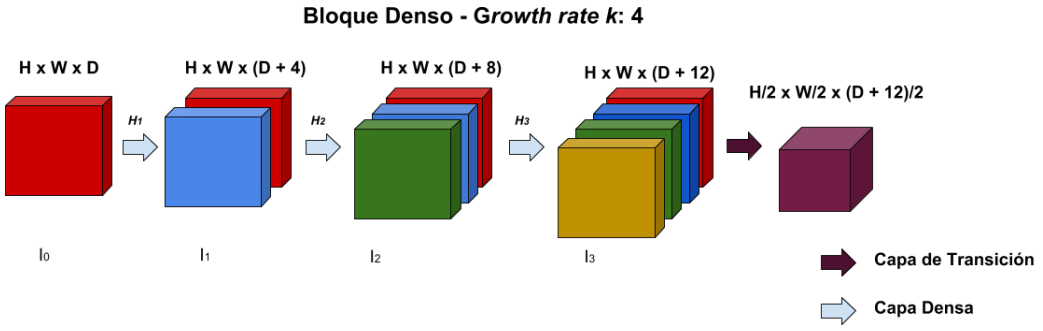

Fig. 2.15: Bloque denso en conexiones con growth rate k = 4 y cantidad de capas $\theta = 3$

siempre y cuando no cambien de orientación: esto se conoce como **invarianza a la traslación**.

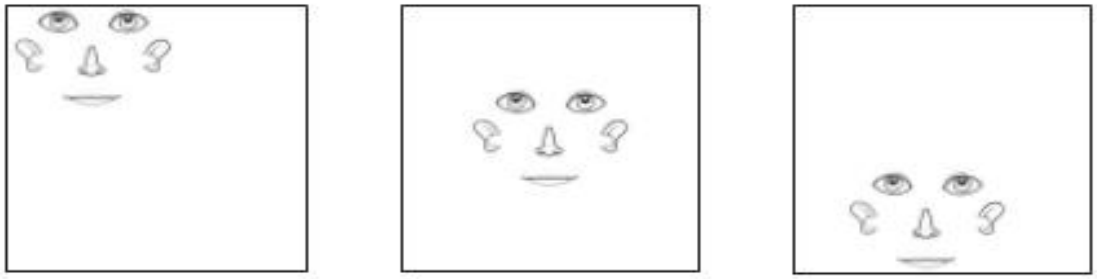

Fig. 2.16: Los kernels son invariantes a la traslación al reconocer features aprendidas

Esta propiedad de los kernels convolutivos es una limitación que genera dos grandes deficiencias a la hora de clasificar imágenes modificadas de manera sutil.

La primer deficiencia surge al rotar las imágenes a clasificar. En el ejemplo de la Figura 2.17 se espera que, al rotar la misma, la red siga reconociéndola como tal pero esto no ocurre ya que los kernels son sensibles a la rotación. Para que una red convolucional pueda aprender una característica con cierto grado de rotación, es necesario utilizar muestras que presenten dicha característica durante la fase de entrenamiento. En estos casos se obtienen falsos negativos, ver Figura 2.17.

La segunda deficiencia, relacionada con el uso de *max-pooling* es que, al *reducir* a un único valor distintas áreas de la entrada, se genera pérdida de información entre features, como por ejemplo, la posición relativa de uno respecto a otro. En consecuencia, los kernels convolutivos aprenden features aisladas, inconexas, que pueden resultar en falsos positivos. Para poner a prueba esta falencia es necesario tomar una imagen, separar las features que la componen para situarlas luego, **sin rotarlas**, en posiciones espaciales tales que las posiciones relativas entre sí no tengan sentido considerando la naturaleza del dataset (ver Figura 2.18).

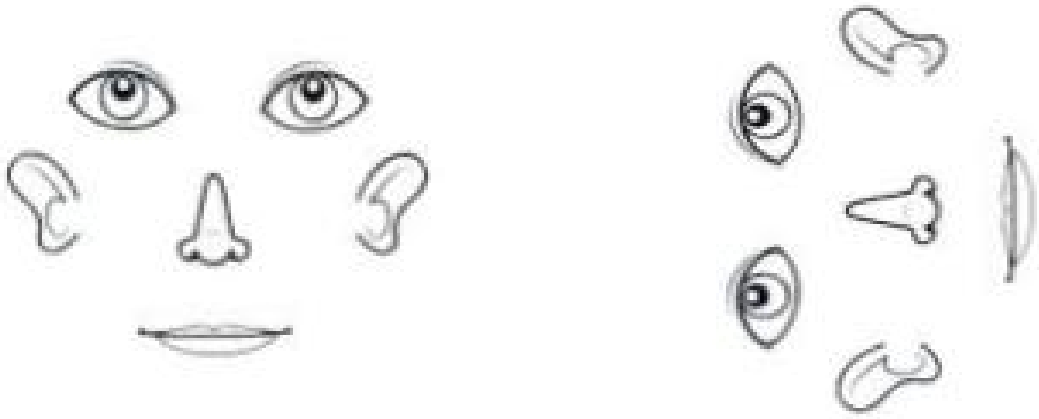

Fig. 2.17: La red convoucional no reconocerá caras rotadas, brindando falsos negativos.

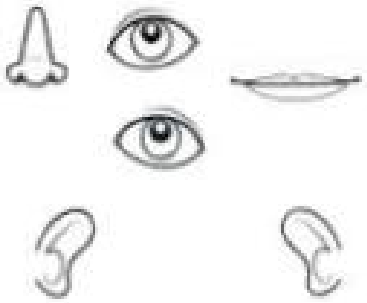

Fig. 2.18: La red convolucional reconoce una cara cuando no la hay, brindando así un falso positivo.

En este contexto, y con el objetivo de solucionar las falencias mencionadas anteriormente, fueron propuestas las redes de cápsulas [25]. La idea de estas redes es modelar de forma más explícita los cambios de orientación, de ubicación, etcétera, manteniendo información referida a la ubicación relativa de una *feature* con respecto a las demás.

Hay dos componentes principales en una arquitectura de cápsulas: por un lado un *encoder*, utilizado a su vez como clasificador, y por el otro un *decoder*, utilizado como regularizador. El encoder se encarga de generar una codificación representada mediante un vector, para cada imagen de entrada. Por otro lado, el decoder (conformado por un perceptrón multi-capa compuesto por neuronas totalmente conectadas) se encargará de tomar dicho vector para reconstruir la imagen de entrada. Tanto el vector de codificación como la imagen reconstruida, servirán para corregir los parámetros entrenables de la red, los cuales introduciremos a continuación.

En el encoder de una red de cápsulas, cada capa se conforma de grupos de neuronas denominados cápsulas. De esta forma, la salida de una cápsula ya no es un valor escalar como en las neuronas de una red neuronal clásica, sino que ahora es interpretada como un vector. Cada cápsula se corresponde con una determinada feature/entidad que puede estar presente en la imagen. La actividad de las neuronas dentro de una cápsula es interpretada como las propiedades de dicha entidad (por ejemplo su posición, tamaño, orientación, deformación, etcétera).

Las cápsulas se organizan de manera arbórea y mediante un proceso iterativo, cada cápsula que se activa en un determinado nivel del árbol escoge una cápsula del nivel inmediatamente superior para que sea su padre. Este proceso es el que se encargará de asignar partes a un todo.

Una propiedad especial que interesa modelar es la presencia, en alguna ubicación, de una entidad particular en la imagen. La manera de representar esto es mediante el uso de la magnitud del vector de activaciones de la cápsula (a mayor magnitud, mayor probabilidad de que dicha entidad o característica esté presente en la imagen de entrada). Una vez determinada la presencia, lo que se hace es reforzar la orientación del vector en cuestión.

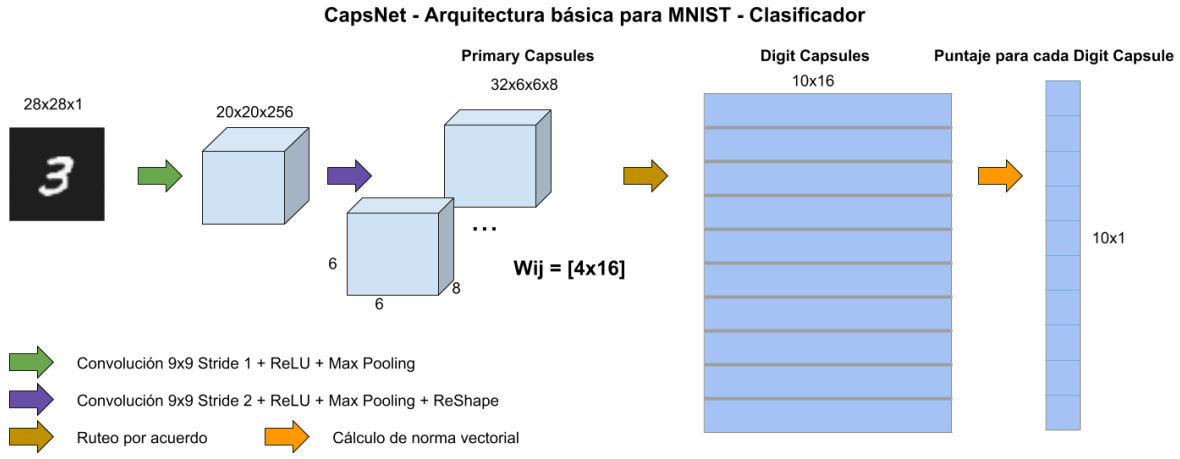

Fig. 2.19: Encoder/clasificador de una CapsNet destinada al reconocimiento de números manuscritos (MNIST) [25].

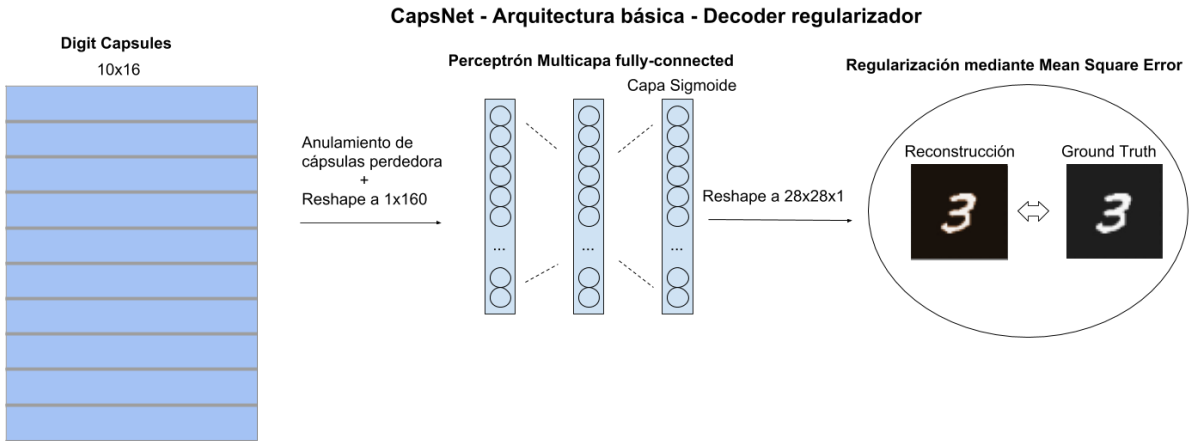

Fig. 2.20: Decoder/regularizador de una CapsNet destinada al reconocimiento de números manuscritos (MNIST) [25].

La magnitud del vector/cápsula estará entre 0 y 1, buscando así corresponderse con la

probabilidad de que la feature que captura la cápsula está presente. Para forzar los vectores unitarios, se utiliza una función no lineal de *squashing*: los vectores cortos se llevan cerca de 0, y los vectores largos cerca de 1. Notar que la misma no modifica la orientación del vector. La función de squashing es entonces definida como:

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||}, \tag{2.15}$$

donde v_j es el vector de salida de la cápsula j y s_j es la entrada de la misma.

La salida de una cápsula será entonces el vector tras la aplicación de *squashing*: esto hace posible el uso del mecanismo dinámico de ruteo que asegura que la salida de una cápsula es ruteada al padre correcto.

En un comienzo, la salida de una cápsula será ruteada a todas las posibles cápsulas padres. Para cada una de éstas, se calcula un vector de predicción a partir de una matriz de pesos. Luego, si el valor escalar que se obtiene a partir del producto entre el vector de predicción calculado y una posible cápsula padre es elevado, entonces se obtiene un *feedback top down* tal que se aumenta el nivel de acoplamiento con esa cápsula padre. De esta manera, se inhibe la activación de las demás posibles cápsulas padre. Esto se conoce como *ruteo por acuerdo*.

Para todas las capas de cápsulas, excepto la primera, la entrada de la cápsula s_j es una suma ponderada de los vectores de predicción $\hat{u}_{ij}$ de la capa anterior que se producen mediante la multiplicación entre la matriz de pesos W_{ij} y u_i, donde u_i es la salida de una cápsula de la capa inmediatamente inferior (notar que por cada cápsula primaria habrá al menos dos matrices de pesos, una por cada clase definida, para realizar el cálculo de los vectores de predicción). Dichos términos se definen entonces como:

$$\hat{u_{ij}} = W_{ij} u_i, \tag{2.16}$$

$$s_j = \sum_i c_{ij} \hat{u}_{ij}. \tag{2.17}$$

En la ecuación anterior c_{ij} es el coeficiente de acoplamiento entre una cápsula y una cápsula padre determinado durante la ejecución del algoritmo de enrutamiento utilizando la función softmax, donde cada b_{ij} correspondiente es considerado como el logaritmo de la probabilidad de que la cápsula i sea acoplada a la cápsula j:

$$c_{ij} = \frac{e^{b_{ij}}}{\sum_k k e^{b_{ik}}}, \tag{2.18}$$

donde b_{ij} depende de la localización y tipo de las cápsulas i y j, pero no de la imagen de entrada.

Estos coeficientes se van refinando midiendo el acuerdo entre la salida v_j de cada cápsula j en la capa siguiente y la predicción $\hat{u}_{ij}$, hecha por la cápsula i en la capa actual. El acuerdo es, simplemente, el producto escalar $a_{ij} = v_j \hat{u}_{ij}$. Este acuerdo se adiciona a b_{ij} antes de calcular los nuevos valores para todos los coeficientes de acoplamiento mediante softmax, que vincula la cápsula i con las cápsulas de nivel superior. La cantidad de iteraciones r, se fija en 3 y solventa problemas de estabilidad numérica.

Por ejemplo, para la configuración de la Figura 2.19, $i \in (1, 32 \times 104 \times 104)$ (donde $32 \times 104 \times 104$ es la cantidad de cápsulas primarias totales) y W_{ij} es una matriz de 8x16

con $j \in (0, 1, \ldots, 9)$, una por cada cápsula de clasificación. En esta configuración, cada cápsula primaria se compone de 8 neuronas, y cada cápsula de clasificación se compone de 16 neuronas: de aquí las dimensiones de W_{ij}.

A continuación se define el algoritmo de ruteo, cuyo funcionamiento fue descripto en los párrafos anteriores:

Algorithm 1 Algoritmo de ruteo por acuerdo

procedure ENRUTAMIENTO($\hat{u}_{ji}$, r, l)
  para todas cápsula i en la capa l y cáspula j en la capa (l + 1): $b_{ij} \leftarrow 0$
  for `r iterations` **do**
    para toda cápsula i en la capa l: $c_i \leftarrow softmax(b_i)$ 2.18
    para toda cápsula j en la capa (l + 1): $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
    para toda cápsula j en la capa (l + 1): $v_j \leftarrow squash(s_j)$ 2.15
    para toda cápsula i en la capa l y cápsula j en la capa (l + 1): $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i}v_j$
  return v_j.

Como se dijo, utilizamos la magnitud del vector de instanciación para representar la probabilidad de que la entidad asociada a la cápsula exista en la imagen. La magnitud del vector será grande sólo si la entidad de la clase k está presente en la imagen.

Para entrenar este modelo, consideremos a modo de ejemplo el problema de clasificación de imágenes de números manúscritos (MSNIST) en diez clases, que se corresponden con diez cápsulas de salida: la cápsula de la clase cero (ej. cero, clase 0), la cápsula de la clase uno (clase 1), etc. Llamaremos a estas cápsulas de la última capa Digit Capsules, dado que representan la presencia o ausencia de un dígito. Dada una imagen, durante el entrenamiento, se calculará un valor de error para cada Digit Capsule. Luego, dichos valores se sumarán y así se computará el error final para esa muestra en particular.

Para el entrenamiento, se utiliza la siguiente función de error, la cual es similar a la que se utiliza para entrenar una máquina de soporte vectorial. Siendo v_c la salida correspondiente a la Digit Capsule de la clase c, m un margen pre-establecido y T_c una variable indicadora de clase ($T_c = 1$ si la imagen corresponde a la clase c), entonces la función de error se define como:

$$L_c = T_c \,\text{máx}(0, m^+ - ||v_c||)^2 + \lambda(1 - T_c).\,\text{máx}(0, ||v_c|| - m^-)^2 \tag{2.19}$$

T_c será 1 (con $c \in \{0,1,2,\ldots,9\}$) cuando la entidad de la clase c esté presente en la muestra actual. Este valor se obtiene a partir de la etiqueta de la muestra (recordar que estamos en un entrenamiento supervisado).

Si la muestra actual presenta un cero, entonces $T_0 = 1$ y $T_i = 0$ con i $\in (1, \ldots, 9)$. Esto implica que el primer término de la función de error es el único que se mantiene y el segundo se vuelve 0. Se toma luego la salida de la Digit Capsule correcta $||v_c||$, y se substrae este valor a m^+ cuyo valor fijo es 0.9. Nos quedaremos con el resultado sólo si es mayor que 0 y lo elevaremos al cuadrado, caso contrario se devuelve 0. Lo que se consigue con esto, es obtener un error de 0 en caso de que se clasifique correctamente con probabilidad mayor a 0.9, y distinto de 0 en caso de que la probabilidad sea menor que 0.9.

En el caso de la Digit Capsule que no se corresponde con la etiqueta correcta de la muestra, $T_c = 0$ y se evaluará el segundo término de la función de error ya que (1 - T_c)

valdrá 1. Se puede ver que la pérdida será 0 si la cápsula predice con una probabilidad menor a 0.1 la existencia de la entidad correspondiente y será distinto de 0 si predice con probabilidad mayor a 0.1. El coeficiente λ es incluido para mantener estabilidad numérica durante el entrenamiento, su valor es fijo: 0.5.

Como ya mencionamos, algo importante en este tipo de redes es el uso del decoder para reconstruir la imagen de entrada, utilizado como **método de regularización**. Utilizar el error de la reconstrucción respecto a la entrada, asegura que las cápsulas de cada clase ajustarán sus parámetros de instanciación correspondientes. Se utiliza la actividad del vector correspondiente a la Digit Capsule ganadora (la que presente mayor magnitud) para reconstruir la imagen de entrada. Este decoder está formado por tres capas totalmente conectadas. Se minimiza la suma de los cuadrados de las diferencias entre la salida de una función sigmoidea (situada al finalizar el perceptrón multicapa) y la intensidad de los píxeles de entrada (error cuadrático medio).

Esta regularización fuerza a que la información guardada en los vectores correspondientes a cada Digit Capsule, sea útil para reconstruir la imagen de entrada. Esto resulta en que dichos vectores codificarán una representación bajo-dimensional de las imágenes de dicha clase, donde cada parámetro tiende a estar asociado a alguna característica de la imagen, como la orientación del dígito, la relación de aspecto, etc.

2.7. Métrica de evaluación

Este capítulo introdujo conceptualmente los dos modelos que utilizaremos para realizar las experimentaciones referidas al problema de clasificación de radiografías torácicas. Es necesario medir el desempeño de cada uno de los modelos entrenados para luego realizar una comparación entre los mismos.

La métrica que utilizaremos es **accuracy** (o tasa de acierto), muy usual en problemas de clasificación binario como el actual, donde se clasificará imágenes radiográficas como patológicas o no patológicas. La tasa de acierto se define como:

$$accuracy = \frac{TPR + TNR}{TPR + TNR + FPR + FNR} \tag{2.20}$$

donde TPR es True Positive Rate, TNR es True Negative Rate, FNR es False Negative Rate y FPR es False Positive rate.

La principal desventaja que presenta esta métrica es que requiere clases balanceadas en el dataset ya que, de lo contrario, la eficiencia del modelo para la clase que es minoría puede pasar desapercibida. Se hará un balanceo del dataset a la hora de realizar la experimentación para que sea viable utilizar esta métrica.

3. MODELOS DE CLASIFICACIÓN PROPUESTOS

A continuación se proponen diversos modelos propuestos en esta tesis de licenciatura en base a los fundamentos teóricos introducidos en la sección anterior, que serán evaluados en la tarea de clasificación de imágenes de radiografía torácica.

3.1. Base de datos y planteo del problema

Se cuenta con la base de radiografías torácicas provista por el National Institute of Health de los Estados Unidos [23] la cual constituye, en la actualidad, uno de los datasets de radiografías con etiquetas asociadas a patologías más grande del mundo. Cada imagen presenta las etiquetas correspondientes a las enfermedades asociadas al paciente (alcanzando un total de catorce enfermedades/cuadros médicos, entre ellas la cardiomegalia, neumonía, infiltración, efusión, presencia de nódulo o masa, entre otras). Contiene 112,120 imágenes de rayos X, vista frontal, de 30,805 pacientes distintos.

Cada muestra fue etiquetada mediante la utilización de técnicas de procesamiento de lenguaje natural sobre los informes/reportes médicos asociados a cada una de las muestras del dataset en cuestión [23].

Como se mencionó al comienzo de este trabajo en la Sección 1.1, aquí nos centraremos en una única patología: la **cardiomegalia** o agrandamiento del corazón. Es por esto que, el problema que se intentará resolver es el de determinar si una radiografía torácica presenta o no cardiomegalia. En otros términos, la imagen que presente cardiomegalia será considerada patológica y la que no tiene la enfermedad será considerada control.

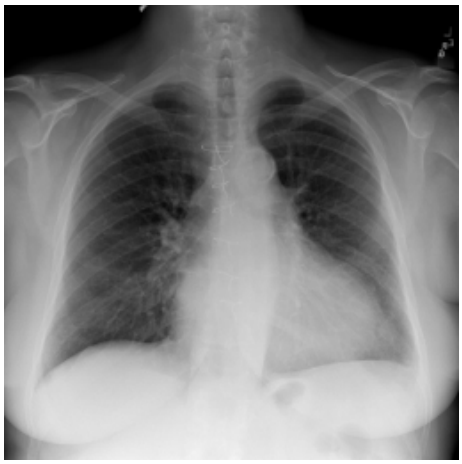

Fig. 3.1: Con cardiomegalia.

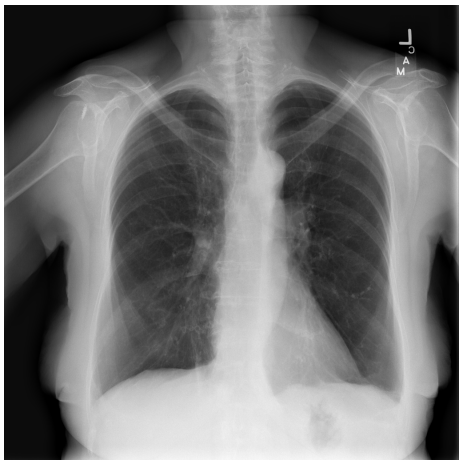

Fig. 3.2: Sin cardiomegalia.

3.2. Modelo basado en DenseNet

En el trabajo original que introdujo el concepto de DenseNets, diferentes arquitecturas densas en conexiones fueron propuestas en el contexto de clasificación de imágenes sobre

el dataset ImageNet [21]. La diferencia entre las mismas radica en la cantidad de capas densas intra bloque denso. Siempre se cuenta con 4 bloques densos y 3 de transición, esto es constante en las arquitecturas densas en conexiones (ver Figura 3.3). Las arquitecturas más destacadas son las siguientes:

	Tamaño salida	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolución	112x112	7×7 conv, strd 2			
Pooling	56×56	3×3 max pool strd 2			
Bloque Denso 1	56x56	1x1 conv. 3×3 conv. × 6	1x1 conv. 3×3 conv. × 6	1x1 conv. 3×3 conv. × 6	1x1 conv. 3×3 conv. × 6
Bloque Transición 1	28×28	1×1 conv. 2×2 avg pool strd 2			
Bloque Denso 2	28x28	1×1 conv. 3x3 conv. × 12	1×1 conv. 3×3 conv. × 12	1x1 conv. 3×3 conv. × 12	1x1 conv. 3×3 conv. × 12
Bloque Transición 2	14×14	1x1 conv. 2×2 avg pool strd 2			
Bloque Denso 3	14×14	1x1 conv. 3x3 conv. × 24	1x1 conv. 3x3 conv. × 32	1x1 conv. 3x3 conv. × 48	1x1 conv. 3x3 conv. × 64
Bloque Transición 3	7×7	1×1 conv. 2×2 avg pool strd 2			
Bloque Denso 4	7×7	1×1 conv. 3x3 conv. × 16	1×1 conv. 3x3 conv. × 32	1x1 conv. 3x3 conv. × 32	1x1 conv. 3x3 conv. × 48
Capa de Clasificación 3	1×1	7×7 avg pool. 1000D FC, sigmoide			

Tab. 3.1: Arquitecturas DenseNet para ImageNet, growth rate $k = 32$, cada capa *conv* hace referencia a la secuencia BatchNorm-ReLU-Convolución. FC hace referencia a un perceptrón multicapa.[21]

Se puede notar que las configuraciones de DenseNets difieren en la cantidad de capas en los Bloques Densos 3 y 4, el resto se mantiene fijo (ver Tabla 3.1). Las redes DenseNet-121 y DenseNet-264 obtienen resultados similares en la tarea de clasificación sobre el dataset ImageNet [21], aún cuando la DenseNet-264 tiene más del doble de capas entrenables que DenseNet-121 [1] .

Entrenar una DenseNet-264 requiere tener más de un GPU para instanciarla por la cantidad de memoria que insume y, en adición, más tiempo de entrenamiento: esta es la razón por la cual en la literatura se puede ver que, a la hora de elegir una red densa en conexiones, se suele utilizar la DenseNet-121 como referencia ya que es entrenable utilizando un único GPU y, por lo mencionado al comienzo del párrafo, sus resultados pueden considerarse como referencia para los modelos densos en general.

En este trabajo utilizaremos la arquitectura **DenseNet-121**. Esta red tiene un *growth rate* $k = 32$ y la cantidad de capas por bloque $\theta = 6$, 12, 24 y 16 para cada bloque denso, respectivamente (ver Tabla 3.1). A continuación desglosaremos y detallaremos el modelo a utilizar.

Primero, a modo de preparación, se aplican 64 kernels convolutivos sobre la imagen de entrada, cada *kernel* define un campo receptivo de 7×7, con *stride* de 2. Luego, se aplica una capa de *max-pooling* cuyo *kernel* es de 3×3 con *stride* de 2. Se agrega a continuación la composición de 4 bloques densos, los cuales se intercalan con las 3 capas de transición.

Como se mencionó, este modelo utilizará un *growth rate* de 32. Esto implica que si tomamos el primer **bloque denso** y lo expandimos, veremos 6 *capas densas* donde cada una agregará 32 nuevos *mapas de características*. Es por esto que luego de aplicar el primer

[1] Los sufijos 121 y 264 de las DenseNets refieren a la profundidad de la red en términos de capas.

bloque denso, pasaremos de tener 64 a 256 (64 + 6×32) mapas de características como se puede observar en la Figura 3.3.

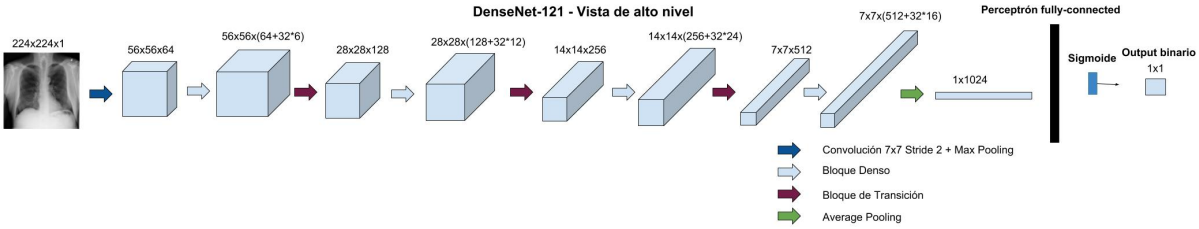

Fig. 3.3: Vista global de una DenseNet-121

Una vez atravesadas todas las capas del bloque denso, el **bloque de transición** se encarga de aplicar una convolución de 1x1 con 128 kernels lo cual implica que, no importa el *growth rate* elegido, la salida siempre será de 128 mapas de características (afectando a la profundidad de la misma). A continuación, se aplica una capa de *average pooling* de 2x2 con stride de 2 la cual reduce a la mitad el alto y ancho de la entrada.

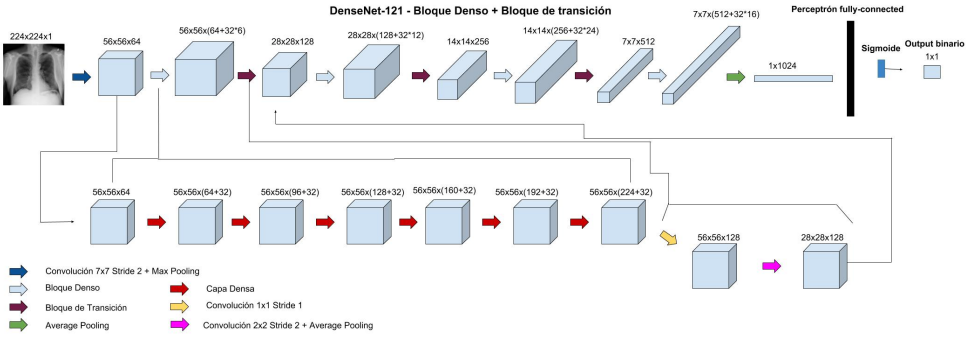

Fig. 3.4: Visualización de un Bloque de Transición y de un Bloque Denso (este último compuesto por Capas Densas).

Desglosando una **capa densa** se puede observar como agrega los 32 nuevos mapas de características: se aplica una convolución de 1x1, idéntica a la de la capa de transición, generando así 128 mapas de características con la entrada (esta operación solo modifica la profundidad resultante). A continuación, se aplican 32 (*growth rate*) convoluciones de 3x3 con stride 1 y padding 1.

Luego, el volumen inicial de la entrada y el resultado de estas dos convoluciones de la capa densa, son concatenados. De esta manera es que se logra agregar nueva información al conocimiento global que se propaga desde la entrada de la red. Ver Figura 3.5.

Finalmente, la salida del último bloque denso se somete a una capa de *average pooling* con un *kernel* es de 7x7, cuyo resultado se redimensiona a un vector unidimensional. Éste será la entrada a un perceptrón multicapa cuyas neuronas de las capas ocultas utilizan la función de activación ReLU introducida en la ecuación 2.8.

A la salida del perceptrón, se obtendrá únicamente un valor el cual será sometido a una función de activación *sigmoide* con el fin de obtener un valor entre 0 y 1, que representará la probabilidad de que la imagen sea patológica.

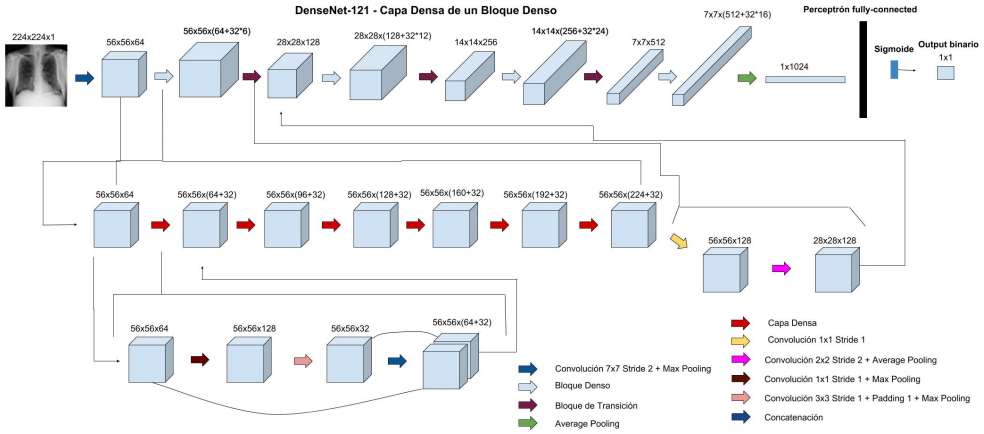

Fig. 3.5: Visualización de una Capa Densa

Se ha utilizado una tasa de aprendizaje inicial de 0.001, con un Optimizador Adam con los parámetros estandard $\beta_1 = 0.9$ y $\beta_2 = 0.999$.

3.2.1. Modelo preentrenado

Generalmente, los pesos sinápticos de una red son inicializados de manera *random* alrededor del 0. Sin embargo, existe una técnica que utilizaremos en nuestro modelo denso la cual consiste en inicializar los pesos sinápticos con los correspondientes a una arquitectura del mismo modelo, pero entrenado sobre el dataset ImageNet [1]. Este modelo tendrá diferente tipos de kernels aprendidos, que pueden ser útiles para guiar y facilitar el aprendizaje sobre nuestro dataset de imágenes médicas (por ejemplo, kernels que reconocen líneas, contornos, etc).

Entrenar redes densas puede resultar lento por la cantidad de parámetros que incluye. Es por esto que utilizar modelos pre entrenados es una opción para acelerar notoriamente este proceso, haciendo que el mismo tienda a converger en menos épocas. Notar que, pese a utilizar un modelo pre-entrenado, todas las capas de dicho modelo serán re-entrenadas (es decir, no se fijarán los pesos sinápticos de ninguna capa, sino que serán refinados durante el entrenamiento).

3.3. Modelo de cápsulas

Como ya se introdujo en la Sección 2.6, los modelo basados en cápsulas constan de dos partes: encoder (que realiza la clasificación) y decoder (que actúa como regularizador).

Si hacemos un desglose de la arquitectura, veremos que en total este tipo de redes están compuestas por 6 capas donde las primeras 3 se corresponden al *encoder/clasificador* y las demás al *decoder/regularizador*.

- Capa 1: Capa Convolucional
- Capa 2: Capa de cápsulas primarias
- Capa 3: Capa de Disease Capsules

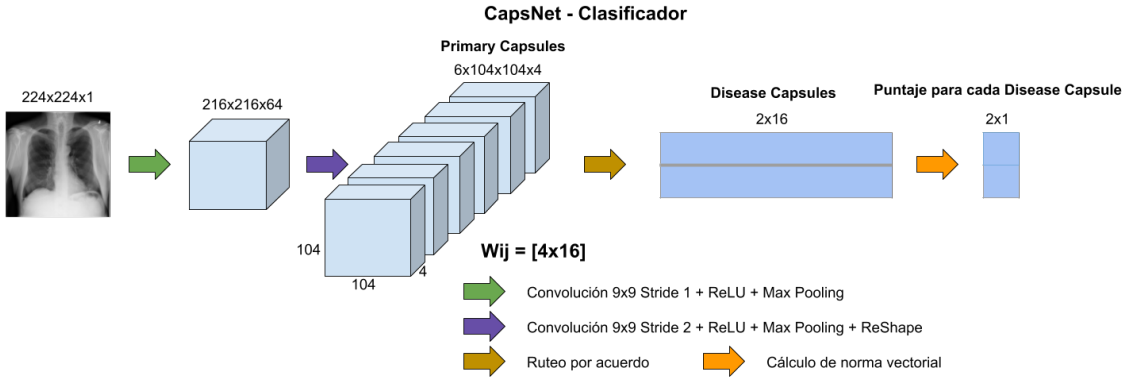

Fig. 3.6: Modelo base de CapsNet con 3 capas. Las longitudes de los vectores correspondiente a las cápsulas de enfermedad indican la presencia de un corazón normal o uno agrandado, esto es utilizado para calcular el error de clasificación.

- Capa 4: Fully-connected 1
- Capa 5: Fully-connected 2
- Capa 6: Fully-connected 3

Clasificador

En el contexto de este trabajo, el **encoder/clasificador** se encargará de tomar como entrada radiografías torácicas y aprenderá a codificar dichas imágenes a vectores de 16 dimensiones (cápsulas de 16 unidades a las que llamaremos Disease Capsules dado que estarán asociadas a la presencia o no de la patología en cuestión, en este caso, de cardiomegalia). Las imágenes originales son escaladas a 224x224 píxeles (ver Sección 3.8 para más detalles sobre el pre-procesamiento de las imágenes) y normalizadas siguiendo el método z-scores (ver Sección 3.8), es decir, centrando su valor medio en la intensidad 0 y escalando el desvío estandar a 1.

La primer capa (ver Figura 3.6) es una capa convolucional convencional cuya tarea es detectar features básicas en la imagen. La cantidad de mapas de características a generar por esta capa es un parámetro que variaremos a la largo de la experimentación (a modo de ejemplo, lo fijaremos en 64). Cada *kernel* de esta capa tendrá, de manera fija, tamaño 9x9 con *stride* de 1 seguido de una función de activación ReLU.

El resultado de esta primera capa alimentará la segunda capa del *encoder/clasificador*: capa de cápsulas primarias (la cantidad de cápsulas primarias también será un parámetro a explorar). Como se mencionó en la Sección 2.6, la tarea de estas cápsulas es tomar las features básicas generadas previamente y producir combinaciones entre las mismas. Cada neurona de estas cápsulas primarias tiene definido un campo receptivo, propio de una convolución, con *kernel* 9x9, stride de 2, con profundidad acorde al volumen resultante de la capa anterior. La cantidad de neuronas que tiene una cápsula será un parámetro a validar.

La tercera (y última) capa del *encoder/clasificador*, está compuesta por 2 cápsulas Disease Capsule. La salida de una Disease Capsule estará asociada a muestras patológicas,

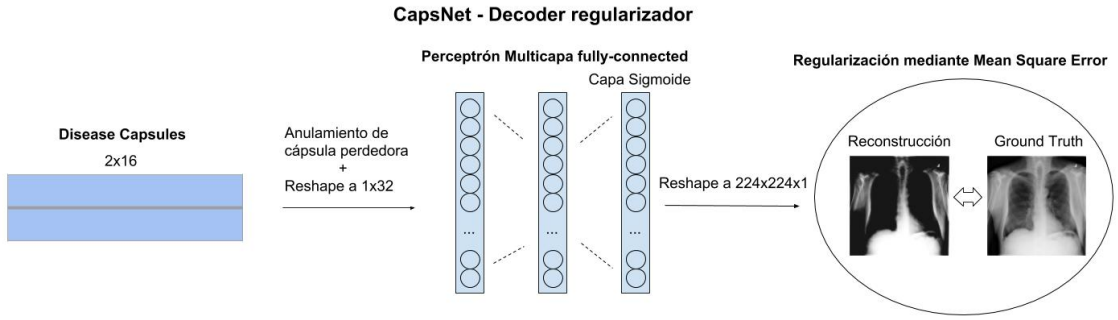

Fig. 3.7: Decoder utilizado para reconstruir la radiografía a partir de las Disease Capsules. La distancia euclídea se minimiza a partir de la salida de una capa final de activación Sigmoidea, durante el entrenamiento.

y la otra a imágenes de pacientes sanos. Se definen entonces 2 matrices (una por cada Disease Capsule) para cada cápsula primaria. Esto es requerido para calcular los vectores de predicción para cada cápsula primaria con respecto a cada Disease Capsule.

Una vez calculados los vectores de salida correspondientes a cada Disease Capsule, utilizando los vectores de predicción, se calculan las respectivas magnitudes (norma Euclídea) y se selecciona entonces una Disease Capsule como ganadora para determinar la clasificación.

A continuación se muestra como se compone la segunda parte de una CapsNet: el **decoder** (ver Figura 3.7). El mismo toma como entrada la salida de las Disease Capsules, dimensionadas de manera tal que sean un único vector unidimensional. Se anulan los activaciones correspondientes a la Disease Capsule perdedora.

Regularizador

Como se mencionó previamente, el *decoder* es utilizado como *regularizador* y aprende a reconstruir radiografías torácicas utilizando como función de error el error cuadrático medio. La idea es forzar a las cápsulas a aprender *features* que sean útiles para la reconstrucción de la imagen original.

La primer capa del *decoder/regularizador* está constituida por la salida de las Disease Capsules, las cuales son ponderadas y se utilizan como entrada de una capa *fully-connected* (segunda capa del *decoder/regularizador*). Esta capa recibe un vector de tamaño 32 (16x2), cuyas activaciones son dirigidas a cada una de las 512 neuronas internas del mismo. Se aplica la función de activación ReLU y luego se pasará el resultado a la tercer capa *fully-connected* de 1024 neuronas. La salida de esta capa se compone de 224x224 unidades (dimensiones de la entrada).

Como se mencionó, la entrada para este modelo serán las radiografías crudas (sólo preprocesadas por medio de downsampling y normalización z-score). El decoder/regularizador tendrá en su última capa una función sigmoide, y como función de pérdida, se utilizará el error cuadrático medio (*mean square error*). Tomará como referencia de ground truth para la regularización la misma radiografía que toma como entrada.

A esta configuración la llamaremos **RAW-RAW**, en contraposición a los modelos que serán presentados en la siguiente sección, los cuales utilizarán no sólo las imágenes crudas

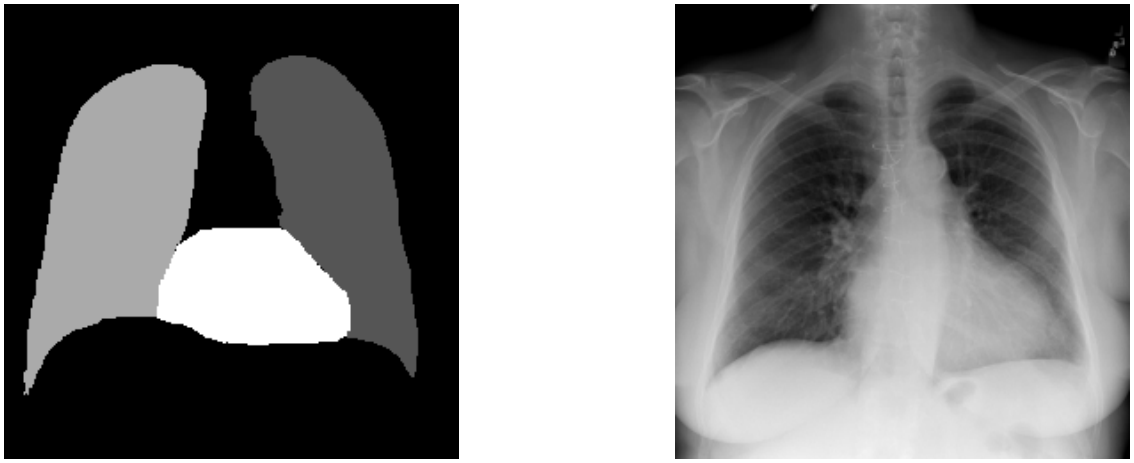

Fig. 3.8: Ejemplo de segmentación la cual distingue el corazón y ambos pulmones por separado. A la derecha se puede ver la radiografía correspondiente.

(raw), sino que incorporarán además segmentaciones anatómicas.

Se ha utilizado una tasa de aprendizaje inicial de 0.001, y se entrena de principio a fin utilizando el algorítmo de optimización Adam con los parámetros estándar $\beta_1 = 0.9$ y $\beta_2 = 0.999$.

3.4. Modelo de cápsulas con segmentaciones

Se tiene acceso a una segmentación del *dataset* que incluye, para cada muestra, una clasificación de los píxeles correspondientes a los pulmones y el corazón. Dicha segmentación fue generada utilizando métodos de segmentación automática basados en modelos multi-atlas. Para mayor información sobre la generación de dichas segmentaciones, ver [42].

Se aprovecharán las segmentaciones para plantear diferentes modelos de cápsulas con entradas de diferente tipo y variaciones en el decoder/regularizador. Nos basaremos en la hipótesis de que agregar información a la entrada o brindar atención a la red (para que se enfoque en ciertas regiones de interés) implica la obtención de mejores resultados de clasificación. Se espera que estas técnicas ayuden a la red a librarse del ruido que presentan las imágenes radiográficas (complejas por naturaleza) ya que las segmentaciones son más simples y contienen, particularmente, información referida a la estructura del corazón. Esta información es relevante para determinar si un paciente presenta o no cardiomegalia.

Las segmentaciones presentan píxeles anotados categóricamente. En total se tienen 4 categorías, representadas mediante números enteros. Una categoría se corresponde con el corazón, otra con el pulmón derecho, otra con el izquierdo y finalmente la cuarta categoría se corresponde con todo lo que no es corazón ni pulmón (ver Figura 3.8).

Para todos los modelos que siguen, se utilizará 0.001 como *learning rate* inicial y un optimizador Adam con los parámetros estandard $\beta_1 = 0.9$ y $\beta_2 = 0.999$.

3.4.1. Segmentaciones Crudas

Para esta configuración, la entrada será la segmentación en formato categórico, normalizada con z-scores (ver Sección 3.8). Esta es la manera naive de utilizar las segmentaciones

ya que no son combinadas con las imágenes crudas para adicionar características, ni utilizadas para brindar atención a la red. Al utilizar las segmentaciones, las cuales mantienen unicamente la información referida a la anatomía de los órganos de la caja torácica, se espera que la red reconozca con mayor facilidad el corazón agrandado en muestras patológicas (i.e con cardiomegalia).

La arquitectura a utilizar será la misma del modelo base presentado en la sección anterior para la configuración **RAW-RAW**. El decoder/regularizador tendrá al final una función de activación sigmoide y la función de pérdida a utilizar será el error cuadrático medio (*mean square error*). A su vez, el decoder tomará como referencia de ground truth la segmentación que usa como entrada. Por esto, a esta configuración la llamaremos **SEG-SEG**.

3.4.2. Segmentaciones en formato One-Hot

Un problema del formato categórico es que genera que la red aprenda una relación de cercanía entre categorías, dada por el orden numérico de las mismas. Esto es incorrecto, y puede derivar en malos resultados de clasificación. Es por esto que se transformarán las segmentaciones a formato One-Hot. *One-Hot encoding* otorga una representación vectorial binaria B para cada píxel donde $B_c = 1$ y $B_i = 0, \forall i \in (0,\ldots,3) : c \neq i$, donde c es la categoría correspondiente al píxel en cuestión. Dicha transformación evitará además una desnormalización entre la entrada y el campo receptivo del primer kernel convolutivo de la red.

Esta configuración tendrá al final una función de activación *softmax* y la función de pérdida entropía cruzada ya que utiliza la segmentación One-Hot como ground truth para la regularización. A esta configuración la llamaremos **SEG1Hot-SEG1Hot**.

3.4.3. Concatenación

Esta configuración tomará cada segmentación en formato One-Hot y su muestra correspondiente para concatenarlas entre sí. La intención de concatenar la segmentación es **agregar información extra** (a modo de *features*) de cada uno de los píxeles: interesa ayudar a la red a reconocer los píxeles del corazón y así determinar el aumento de tamaño del mismo cuando corresponda.

Nuevamente, la reconstrucción se hará basándose en la imagen cruda, es por esto que la configuración se llamará **RAW+SEG1Hot-RAW**: se utilizará una función sigmoidea al final de la arquitectura y la función de pérdida *mean square error*.

3.4.4. Masking

Esta última configuración tomará como entrada la imagen cruda. La particularidad es que la regularización será hecha considerando como ground truth la imagen cruda a la cual se le aplicará *masking* utilizando la segmentación. Con esto, se buscará que la red considere únicamente el corazón y los pulmones a la hora de realizar la regularización, focalizando así en las entidades más relevantes de la radiografía durante el aprendizaje.

Para realizar *masking*, se obtiene una máscara binaria por cada segmentación. Cada máscara tendrá píxeles con valor 0 por cada píxel correspondiente al fondo de la radiografía. Caso contrario, el valor será 1 (píxeles de los pulmones y corazón). Realizando una multiplicación píxel a píxel entre la radiografía y la máscara, lograremos anular todos los

píxeles de la imagen original que no conforman el área abarcada por la segmentación, ver Figura 3.9.

La intención es brindar **atención** a la red al momento de realizar la regularización, filtrando información ajena a los pulmones y corazón. A esta configuración la llamaremos **RAW-MaskedRAW**.

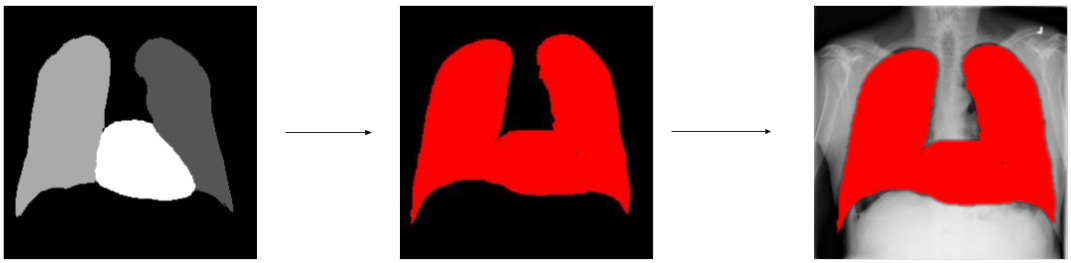

Fig. 3.9: MaskedRAW. Luego de aplicar la máscara, solamente prevalecerá la zona de la radiografía marcada con rojo para ser considerada al momento de la regularización.

3.5. Composición entre modelo denso y modelo de Cápsulas

Esta configuración tiene como objetivo aprovechar las cualidades de ambos tipos de redes neuronales: la extracción de features de las redes densas y el ruteo por acuerdo de las redes de cápsulas.

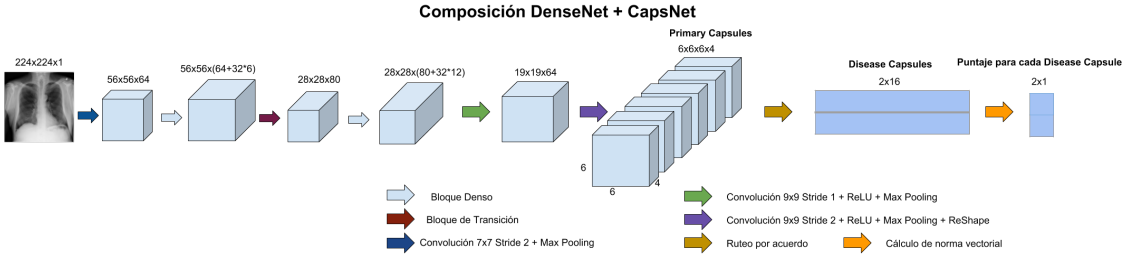

Fig. 3.10: Composición entre DenseNet y CapsNet. Growth rate $k = 32$, dos bloques densos cuyos θ son 6 y 16 respectivamente.

Se considera la primera parte de la arquitectura DenseNet-121: la convolución inicial y los primeros dos bloques densos. Entre dichos bloques se encuentra el correspondiente bloque de transición. Luego, se toma una CapsNet con 64 kernels convolutivos iniciales, seguido a esto se generarán 6 tipos de cápsulas primarias donde cada una de estas se compondrá de 4 neuronas. La cantidad de unidades por Disease Capsule se mantiene en 16 como en todos los modelos de cápsulas presentados.

Finalmente, el decoder/regularizador tomará como referencia ground truth la radiografía que toma como entrada. Se utilizará la función de error mean square error para estimar el error de la reconstrucción (ver Figura 3.7).

3.6. Funciones de error

3.6.1. Entropía cruzada

La entropía cruzada (cross entropy) computa un coeficiente de penalidad según la diferencia entre una distribución de probabilidad estimada q y la correspondiente distribución deseada p, se define como:

$$H_{p,q}(y) = -\frac{1}{N}\sum_{y} p(y)log(q(y)), \tag{3.1}$$

con N, el número de observaciones.

3.6.2. Entropía cruzada binaria

Esta función de costo es la más habitual en problemas de clasificación binaria, es decir, donde se escoge entre dos etiquetas para asignar a una muestra, es un caso particular de la entropía cruzada:

$$H_p(Y) = -\frac{1}{N}\sum_{y}^{Y} ylog(p(y)) + (1-y)log(1-p(y))), \tag{3.2}$$

con N, el número de observaciones.

Si y es la etiqueta (en este caso 1 será patológico y 0, *no* patológico) a asignar a la entrada x, entonces $p(y)$ representa la probabilidad de que la muestra x sea patológica.

Entonces, si por ejemplo y debe ser etiquetada con 1 pero $p(y)$ es un número muy cercano a 0 entonces $log(p(y))$ será un número negativo cuyo valor absoluto es muy alto. Luego, si le cambiamos el signo obtendremos un número positivo elevado el cual representa justamente la alta penalidad que buscamos. Análogamente, ocurre lo mismo si la etiqueta debió ser 0.

3.6.3. Error cuadrádatico medio

Esta función de error mide el promedio de los errores al cuadrado, es decir la diferencia entre lo que se estima y el estimador. Sea $\hat{Y} = (\hat{y_1} \dots \hat{y_N})$, un vector de predicciones e $Y = (y_1 \dots y_N)$ son los valores esperados correspondientes entonces el error cuadrático medio para el predictor es el siguiente:

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2. \tag{3.3}$$

3.7. Metodología de entrenamiento

En este contexto de entrenamiento supervisado (donde se aprovecha el etiquetado de las radiografías para ajustar los parámetros entrenables de las redes) la metodología adoptada fue la siguiente:

- Primero se particiona el *dataset* en tres conjuntos: *training, validation y testing* los cuales no deben solaparse entre sí.

- Se entrena el modelo utilizando el *training* set. Durante este proceso, cada vez que se itera por completo dicho conjunto se concreta una *época*. La cantidad de épocas a ejecutar depende de la velocidad de aprendizaje del modelo y el *criterio de corte* a utilizar. Para esto, se considera el *validation* set para poner a prueba el modelo tras finalizar cada época: se obtiene una métrica de eficiencia del modelo para determinar cuándo detener el entrenamiento y evitar así el sobreajuste o *overfitting* [1] sobre el set de entrenamiento, quedándonos con el mejor modelo hasta el momento según dicha métrica.
- Una vez finalizado el entrenamiento, se toma el conjunto de *testing* para reportar la eficacia del modelo mediante una métrica. Como hemos mencionado en la Sección 2.7, en nuestro caso será *accuracy*.

3.8. Preprocesamiento

Estrategias de muestreo

La forma en la cual se eligen las imágenes del conjunto de entrenamiento puede impactar en la velocidad de convergencia del aprendizaje de la red. Para ejemplificar esto, consideremos las siguientes situaciones extremas:

- Si tomaramos aleatoreamente, de a una imagen a la vez, al realizar el cálculo del gradiente de la función de costo respecto a los parámetros de la red, el mismo indicaría direcciones muy cambiantes, no representativas, para alcanzar un mínimo.
- Si utilizaramos el dataset de entrenamiento completo para calcular el gradiente y actualizar los pesos de la red, el proceso sería sumamente lento y costoso.

Una opción intermedia es la utilización de *mini batches* de imágenes. Usaremos *mini batches* de 10 muestras en este trabajo. Como hemos mencionado, la métrica de evaluación *accuracy* exige tener conjuntos de *training, validation y testing* balanceados. Es por esto que el particionado del dataset se hará considerando las especificaciones que siguen:

- Los conjuntos de *training, validation y testing* estarán balanceados en términos de muestras por clase.
- No hay solapamiento de pacientes entre dichos conjuntos (recordar que las radiografías están asociadas a pacientes, cada uno de ellos presenta más de una radiografía por ser parte de la correspondiente historia clínica).
- Se utilizan todas las muestras con cardiomegalia del dataset para conformar los tres conjuntos del particionado.
- Se toma la misma cantidad de imágenes sin cardiomegalia para el particionado, elegidas al azar, tales que no rompan las condiciones mencionadas en los items anteriores.
- Los *sets* de, *training*, *validation* y *testing* iniciales tendrán: 3900, 530 y 1162 muestras respectivamente. Los tres conjuntos están balanceados por definición.

De esta manera, al garantizar el balance de los datos, puede utilizarse sin problemas la métrica *accuracy*.

[1] *Overfitting*es una situación que ocurre cuando se sobre ajusta un modelo a los datos de entrenamiento, perdiendo su poder de generalización en datos no vistos durante el proceso de aprendizaje.

Aumentación de datos

Debido a que en el set de entrenamiento sólo se cuenta con 1950 muestras con cardiomegalia, realizaremos un aumento de los datos buscando entrenar modelos más robustos. En general, las redes profundas detectan y aprenden patrones con mayor facilidad si se les provee una gran cantidad de muestras durante el entrenamiento. Además, esta técnica ayuda a evitar el *overfitting*. La manera en la cual se realiza la aumentación de datos es la siguiente:

- Se se adoptó la misma técnica que en otros trabajos que presentan resultados sobre imagenes similares que consiste en realizar un *flip horizontal* de las muestras que presentan cardiomegalia [45]. Obteniendo así un total de 3900 muestras patológicas para el set de *training*.
- Se toman 1950 muestras nuevas sin cardiomegalia.
- El conjunto de *training* quedará balanceado y contará finalmente con 7800 muestras.

Resampleo

Durante la experimentación se ha realizado un *downsampling* a 224x224 de las imágenes, cuyo tamaño original era de 1024x1024 píxeles. La intención es reducir la cantidad de parámetros a entrenar, implicando una reducción del tiempo de entrenamiento y la memoria del GPU requerida.

Normalización

Una práctica usual es realizar una normalización z-scores de los píxeles de las imágenes a utilizar. La misma consiste en sustraer la media μ de la intensidad de los píxeles y dividir por el desvío estandard σ:

$$z_i = \frac{x_i - \mu}{\sigma}. \tag{3.4}$$

Early Stopping

Como ya hemos mencionado, el entrenamiento supervisado de cada modelo consiste en suministrar *mini batches* de muestras a la red, de esta manera se irán corrigiendo los parámetros entrenables de la misma. Cada vez que se termina de recorrer todo el set de entrenamiento, se concreta una *época*. Si bien se elige un número máximo de *épocas* de entrenamiento, se suele determinar un criterio de corte para finalizar antes y así asegurarse de que el modelo que se toma como resultante es el mejor: entrenar durante más épocas no implica necesariamente mejores resultados, dado que puede derivar en un *overfitting* sobre el conjunto de entrenamiento acompañado de malos resultados de clasificación en el conjunto de prueba. Utilizaremos la técnica *early stopping* cuyo criterio de parada será calcular el *accuracy* sobre el set de validación y, si pasan 15 épocas sin obtenerse una mejora, entonces detendremos el entrenamiento y nos quedaremos con el modelo que presentó el mejor desempeño hasta entonces.

4. INFRAESTRUCTURA DE ENTRENAMIENTO Y EVALUACIÓN

A lo largo del trabajo se utilizaron dos servidores, ambos cuentan con GPUs Nvidia TITAN XP. Dichos servidores fueron fundamentales debido al poder de cómputo para tratar múltiples imágenes. Realizar la experimentación utilizando CPUs no era viable. Uno de estos servidores se encuentra localizado en el Departamento de Computación de la UBA y el otro se encuentra en la Universidad Nacional del Litoral.

El código del modelo base de redes neuronales basadas en cápsulas fue implementado en Python. Para el caso de las redes densas en conexiones, se ha utilizado una implementación provista por Torchvision.

A continuación se muestran las bibliotecas/tecnologías utilizadas.

Numpy

Numpy [1] es una biblioteca de Python, que agrega soporte para el manejo de arreglos multidimensionales y matrices, ofreciendo una colección de operaciones matematicas de alto nivel para operar sobre estos.

Se utilizó principalmente para el manejo de los resultados y demás información que no requería ser procesada por la GPU. Para manejar las imágenes del set de datos, se han utilizado Tensores y Variables propios de la biblioteca Pytorch/Torch/Torchvision.

Pytorch - Torch - Torchvision

Pytorch[3] es una biblioteca basada en Torch, diseñada e implementada por Facebook que ofrece dos abstracciones sumamente importantes:

- Tensores, los cuales permiten manipular matrices y arrays multidimensionales (como Numpy) admitiendo aceleración por parte del GPU. Algo a notar es que las GPU deben admitir CUDA.
- Redes neuronales profundas ya implementadas: en nuestro caso utilizamos una implementación ya existente de la red DenseNet-121, que se puede hallar en el módulo Torchvision de Pytorch.

Torch[2] es una biblioteca de machine learning open-source, la cual funciona como framework para computación científica. Provee una serie de algoritmos para aprendizaje profundo. A bajo nivel, está implementado en C.

El módulo central de Pytorch se llama torch. Posee colección de operaciones para el manejo de arreglos multidimensionales o Tensores: soporta operaciones tales como indexado, transposición, *resizing*, *slicing*, casteo de tipos. Abstrae, a su vez, el manejo de la memoria, el *sharing* de la misma, como así también el clonado.

Un módulo muy utilizado durante la implementación de nuestra aplicación fue nn: se usó para construir la red neuronal basada en cápsulas. nn provee implementadas, por ejemplo, las siguientes clases: nn.Conv2D, nn.Linear, nn.Sequential, nn.ReLu, nn.Sigmoid,

[1] www.numpy.org
[2] https://pytorch.org/docs/stable/torch.html
[3] https://pytorch.org/

nn.Softmax, entre otras. Todas estas clases extienden nn.Module, que exige implementar, por ejemplo, el método *forward*(). Implementa también el método *backward*(). También provee métodos que permiten guardar redes neuronales en archivos planos (es decir, se puede guardar la instanciación de los parámetros de la misma) para poder instanciarlas nuevamente a partir de estos archivos.

Luego, para armar las redes es necesario combinar diferentes Módulos mediante compositores. El más utilizado en este trabajo fue nn.Sequential (permite ejecutar en secuencia diferentes módulos, uno tras otro).

Finalmente, Torchvision[4] es un paquete de Pytorch que incluye arquitecturas de redes neuronales ya implementadas en Pytorch, así como también data sets populares para la comunidad (como por ejemplo MNIST, CIFAR) y transformaciones sobre imágenes usuales en el campo de visión por computadora.

Compute Unified Device Architecture (CUDA)

CUDA es [5] una plataforma de cálculo computacional en paralelo, desarrollada por NVIDIA. Explota las ventajas de las GPU frente a las CPU de propósito general ya que utilizan el paralelismo que ofrecen sus múltiples núcleos Esto es fundamental para el procesamiento de operaciones sobre múltiples imágenes en paralelo.

Pandas

Pandas [6] es otra biblioteca open source que maneja de manera sencilla y con una alta performance estructuras de datos. A su vez provee herramientas de analisis de datos para Python. Se ha utilizado particularmente para leer y manipular los archivos .csv que almacenan las etiquetas de cada imágen de radiografía torácica.

Python Image Library (PIL)

Agrega soporte para abrir, manipular y guardar imágenes con diferentes formatos [7]. La misma fue fundamental para implementar la clase correspondiente al Dataset (ChestX-RayDataset.class), que se encarga de abrir las diferentes imágenes cuando sean requeridas y aplica, a su vez, las transformaciones pertinentes a las mismas.

Scikit-learn

Esta biblioteca de Python [8] presenta herraminetas simples y eficientes para data-mining y analisis de datos. Fue construida utilizando NumPy, SciPy y matplotlib. Las herramientas utilizadas de esta biblioteca, fueron las que preparan las diferentes combinaciones de parametros (ParameterGrid) para ejecutar diferentes instancias de entrenamiento.

[4] https://pytorch.org/docs/stable/torchvision/index.html
[5] https://www.nvidia.es/object/cuda-parallel-computing-es.html
[6] https://pandas.pydata.org
[7] http://www.pythonware.com/products/pil/
[8] scikit-learn.org/stable/index.html

5. ANÁLISIS COMPARATIVO DE LOS MODELOS PROPUESTOS

En este trabajo se utilizó un conjunto de datos organizado en tres particiones, tal y como se mencionó en la Sección 3.8. El dataset fue normalizado mediante la técnica z-scores (ver Sección 3.8). Cada muestra fue etiquetada a partir de la utilización de técnicas de procesamiento de lenguaje natural sobre los informes/reportes médicos asociados a cada una de las muestras del dataset en cuestión (ver Sección 3.1).

5.1. Selección del mejor modelo de cápsulas

Dado que el modelo denso que utilizaremos es usual en contextos de clasificación de imágenes (DenseNet-121), nos detendremos para explorar y encontrar el mejor modelo de cápsulas para analizarlo y luego realizar las comparaciones entre modelos.

Independientemente de las técnicas de procesamiento que se utilicen sobre la entrada (mencionadas en la Seccion 3.4), existen diferentes parámetros a ajustar, propios de la arquitectura de cápsulas, y son los siguientes:

- Cantidad de *kernels convolutivos* de la primer capa convolucional: el tamaño del kernel siempre será fijo, de 9x9.
- Cantidad de cápsulas primarias: este parámetro define la cantidad de entidades distintas que la red de cápsulas intentará reconocer y capturar para luego conformar un todo uniéndolas.
- Cantidad de neuronas por cápsula primaria.
- Disease Capsules: se tienen 2 cápsulas de clases de 16 unidades cada una. No se variará la cantidad de unidades.
- Decoder/Regularizador: el mismo será fijo, será un perceptrón multicapa de tres capas totalmente conectadas que tienen 512, 1024 y 224x224 neuronas, respectivamente. El número de neuronas de la última capa es la cantidad de píxeles de la imágen de entrada a reconstruir.

Estos párametros fueron variados siguiendo un esquema *greedy*. En el presente trabajo se mostrará el subconjunto de resultados más relevantes. Como ya hemos mencionado en la Sección 2.7, utilizaremos la métrica *accuracy* sobre el conjunto de test para reportar la eficiencia de los modelos.

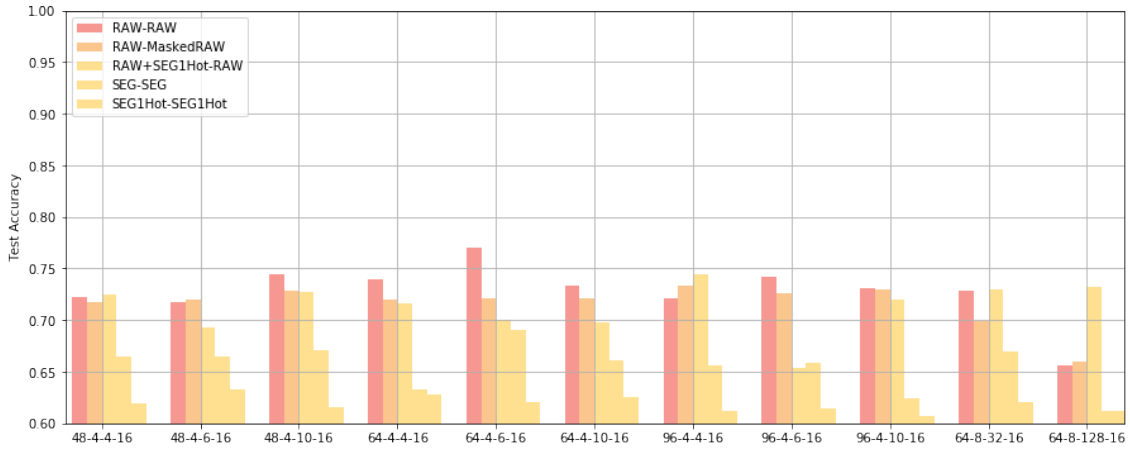

Fig. 5.1: Resultados para cada configuración de cápsulas. Cada tupla del eje de abscisas se mapea a los parámetros: **# kernels convolutivos iniciales - # neuronas por cápsula primaria - # cápsulas primarias - # neuronas por Disease Capsules**

Como se puede ver en la Figura 5.1 la configuración 64-4-6-16 (64 kernels convolutivos inciales, 4 neuronas por cápsula primaria, 6 cápsulas primarias y 16 unidades cada una de las dos Disease Capsules) fue la que mejores resultados brindó, 77 % accuracy en la fase de testeo. Algo a notar es que utiliza como entrada y como ground truth la imagen cruda (**RAW-RAW**).

El desempeño de la red decrementó alrededor del 10 % al utilizar unicamente las segmentaciones (configuración (**SEG-SEG**). Si bien una segmentación es más simple que su correspondiente radiografía, la red de cápsulas no resultó eficiente en la tarea de clasificación al utilizarlas. Posiblemente esto se deba a la calidad de las segmentaciones, las cuales fueron obtenidas por un método automático. Bajo la hipótesis de que dichas segmentaciones contienen la información discriminativa necesaria (en el caso de la cardiomegalia, asociada a la estructura del corazón), se supone que si el poder de una cierta red de cápsulas es suficiente para clasificar radiografías completas con cierto *accuracy*, se esperaba que al hacerlo sobre segmentaciones se obtenga el mismo resultado o mejor porque hay menos cantidad de posibles features a ser capturadas por las cápsulas: la entrada se reduce al corazón y los pulmones. Por esta razón, creemos que la calidad de las segmentaciones no fue suficiente.

En 3 configuraciones (48-4-4-16, 96-4-4-16 y 64-8-128-16) agregar información concatenando las segmentaciones en formato one hot (**RAW+SEG1Hot-RAW**), superó al resto de las configuraciones, aunque dichas diferencias no fueron realmente significativas ni sistemáticas para todas las configuraciones. Para el resto se obtuvo un rendimiento similar o peor. Podemos decir entonces que agregar información asociada a los órganos no mejoró el rendimiento de las cápsulas y en algunos casos hasta añadió ruido.

Algo similar ocurre con las configuraciones que usan atención, mediante *masking*, en el regularizador (**RAW-MaskedRAW**). Dicho modelo sólo prevalece para una única configuración (48-4-6-16). En general sus resultados están siempre cerca, aunque por debajo, de los obtenidos a partir de los modelos **RAW-RAW**. Esto da la pauta de que agregar atención no ayudó a dar un salto significativo en la eficiencia de los modelos de cápsulas.

Tal como puede observarse en las figuras 5.2 y 5.3, la función de pérdida en la partición

de validación tiende a estabilizarse, mientras que la misma en el dataset de prueba continúa disminuyendo. Esto es utilizado como criterio para detener el entrenamiento y evitar así el sobreajuste. Para seleccionar el mejor modelo, se utilizó siempre la estrategia mencionada en 3.8

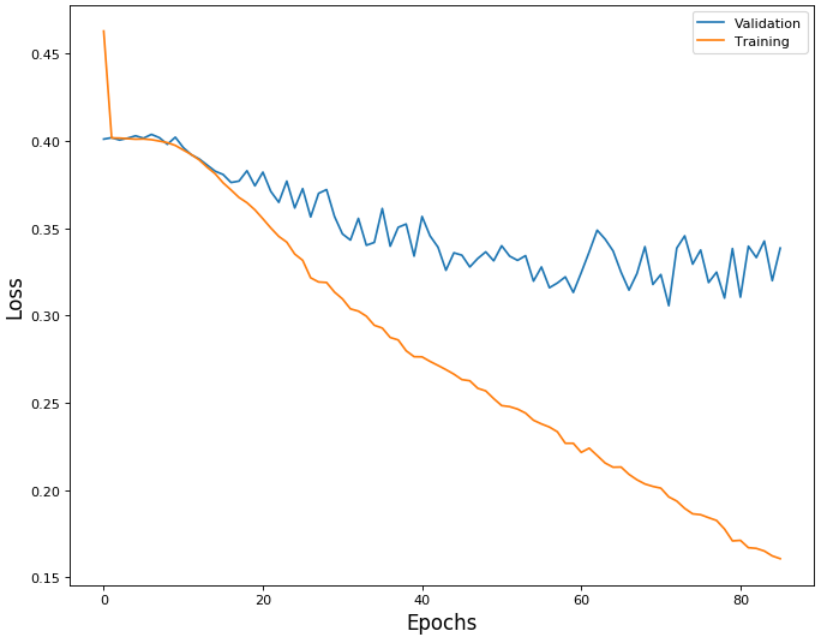

Fig. 5.2: Función de pérdida de entrenamiento y validación para la configuración que mejores resultados dio en la tarea de clasificación de radiografías.

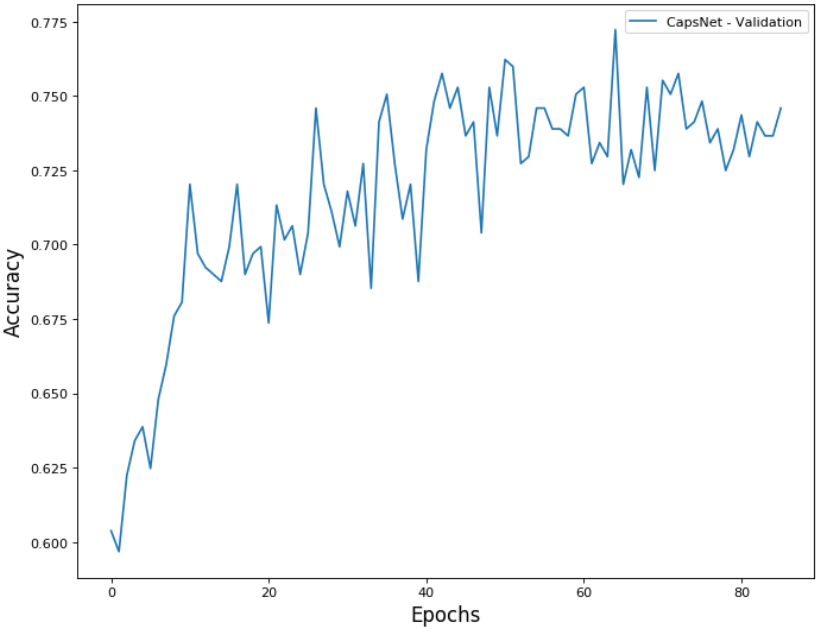

Fig. 5.3: Accuracy sobre el set de validación, época tras época

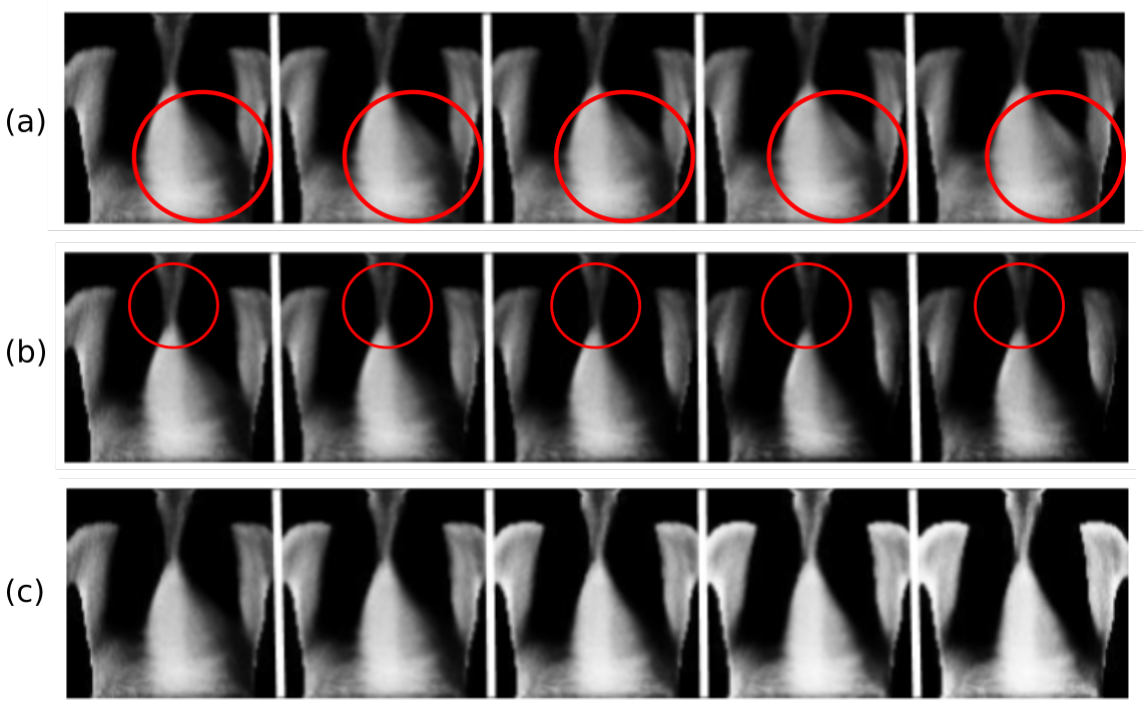

Fig. 5.4: . Variaciones producidas en la reconstrucción de la imagen de entrada a partir de los códigos de la Disease Capsule asociada a la clase patológica, al aplicar pequeños incrementos en 3 códigos diferentes. La primer columna corresponde a la imagen original, y las siguientes a variaciones incrementales de distintos códigos: (a) código asociado al tamaño del corazón, (b) código asociado a la separación entre los pulmones, (c) código asociado al brillo general de la imagen.

5.2. Interpretabilidad del modelo por medio de reconstrucciones

Una particularidad de las redes de cápsulas, es que los vectores de las Disease Capsules pueden ser utilizados para reconstruir las imágenes y tratar de interpretar lo aprendido por el modelo. Bajo la hipótesis de que los diferentes códigos aprendidos por las Disease Capsules se corresponden con distintas características de la clase asociada, veremos cómo se comporta el decoder cuando variamos una a una, de manera independiente, cada activación de la Disease Capsule patológica. Dado que el decoder parte de una representación de baja dimensionalidad de la imagen, siguiendo el principio de funcionamiento de los autocodificadores, sabemos que dichos códigos capturarán información relacionada a las variaciones más significativas entre las imágenes de entrada. Se espera ver que alguna de sus componentes haya capturado características asociadas a la estructura del corazón, dado que es una de las principales herramientas utilizadas por los médicos para realizar el diagnóstico de cardiomegalia.

Dada una Disease Capsule de 16 activaciones $d_1 \dots d_{16}$ correspondiente a una imagen de entrada arbitraria, la Figura 5.4 muestra lo que sucede al explorar el espacio de posibles reconstrucciones variando tres de estas activaciones. Para ello, se realizó una exploración manual de dichas componentes, procediendo de la siguiente forma: tomaremos la activación correspondiente a la imagen de entrada, y a cada d_i le sumaremos un valor $\theta \times p$ con

$\theta = 0{,}01$ y $p \in (0, 4, 8, 12, 16)$. d_j con $i \neq j$ quedarán sin ser modificadas. La Figura 5.4 muestra el resultado para tres componentes diferentes, asociadas al tamaño del corazón (a), la separación entre los pulmones (b) y el brillo general de la imagen (c).

5.3. Estudio comparativo

Habiendo seleccionado el mejor modelo de cápsulas en la Sección 5.1, realizaremos a continuación una comparación entre las diferentes arquitecturas propuestas.

5.3.1. Aumento incremental del dataset

Para realizar la comparación se utilizarán diferentes *training subsets*, generados a partir del *set* de entrenamiento definido en la Sección 3.8, manteniendo el balanceo de clases. La idea de este experimento es evaluar el rendimiento de los diferentes modelos al ser entrenado con distinta cantidad de imágenes, con el fin de observar si en algún régimen específico (con pequeños o grandes datasets) los modelos presentan distintos comportamientos.

- Los *sets* de *validation* y *testing* quedarán igual: es decir, con 530 y 1162 muestras respectivamente, ambos balanceados por definición.
- Cada nuevo *subset* de *training* tendrá: 25, 50, 100, 200, 500, 1000, 1950, 3900 muestras por clase. El *subset* de 3900 radiografías por clase incluye el aumento de datos mencionado en la Sección 3.8.

5.3.2. Modelos comparados y metodología

Para realizar la comparación final entre los modelos propuestos, consideraremos el mejor modelo de cápsulas que puede apreciarse en la Figura 5.1. El segundo modelo a considerar, denso en conexiones, es la red DenseNet-121. Finalmente, el tercer modelo para la comparación será el definido en la Sección 3.5, el cual combina aspectos de las redes densas en conexiones y las redes de cápsulas en una única arquitectura. Para cada modelo, ejecutaremos el entrenamiento con los diferentes *subsets* definidos en la sección anterior y obtendremos el *accuracy* correspondiente sobre el set de test.

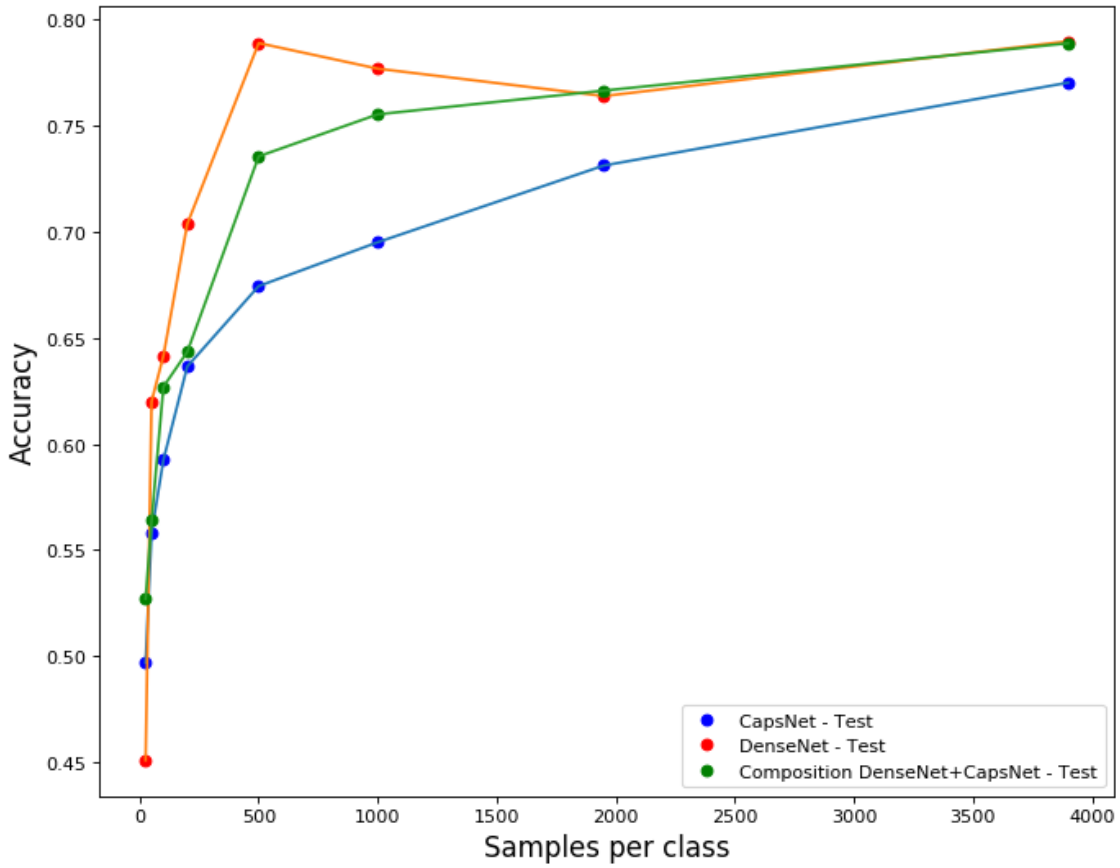

Fig. 5.5: Resultados variando la cantidad de muestras por clase del conjunto de entrenamiento

La Figura 3.10 muestra que el modelo DenseNet-121 supera a los modelos base de redes basadas en cápsulas en el contexto de clasificación de radiografías torácicas. Sin embargo, la composición entre la DenseNet y la CapsNet tuvo un desempeño equiparable con el de la DenseNet-121. Esta composición supera a la red de cápsulas en la tarea de clasificación, y nos da la pauta para afirmar que la extracción de características inicial es una etapa clave en el proceso de clasificación.

El análisis respecto al entrenamiento con datasets de diferentes tamaños revela que la tendencia se mantiene independientemente del tamaño del dataset de entrenamiento. En otras palabras, los modelos DenseNet-121 y compuesto siempre presentan un mejor desempeño que el modelo basado en cápsulas.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Durante este trabajo se describieron teórica y conceptualmente dos modelos de redes neuronales actuales utilizados para clasificación de imágenes. Se introdujo el problema de la clasificación de radiografías torácicas y se plantearon diversos modelos de redes neuronales para resolverlo.

Las redes de cápsulas descriptas en la sección 2.6 de esta tesis fueron introducidas recientemente (en el año 2017), y por lo tanto no abundan en la literatura trabajos que las utilicen en el contexto de clasificación de imágenes médicas y diagnóstico asistido por computadora. En ese sentido, esta tesis de licenciatura constituye un aporte relevante al presentar un estudio minucioso de diferentes modelos basados en redes de cápsulas, evaluados en el contexto de la clasificación de cardiomegalia a partir de imágenes de rayos X y considerando además la inclusión de segmentaciones anatómicas.

El estudio comparativo presentado en la sección 5.3 arroja varias conclusiones interesantes. En primer lugar, se observa que para este problema en particular los modelos base de cápsulas no son más poderosos que las redes densas, independientemente de la cantidad de imágenes utilizadas durante el entrenamiento. Sin embargo, extendiendo el procesamiento inicial de la red de cápsulas para extraer características de la entrada mediante una arquitectura densa, se obtiene un desempeño equiparable al de dichas redes, las cuales son mayormente utilizadas en contextos de visión por computadora. Esto da lugar al planteo de nuevos trabajos a futuro que incluyen: considerar particularmente cada una de las enfermedades pulmonares y verificar si este tipo de composiciones de redes son mejores que las redes densas en conexiones en otro tipo de patologías.

El estudio y visualización de las características aprendidas por los modelos resulta imporante para dar interpretabilidad a los resultados obtenidos. En ese sentido, las redes de cápsulas permiten visualizar características distintivas utilizadas para en el proceso de clasificación. En la Sección 5.1 se incluyeron visualizaciones de dichas características asociadas a distintos factores, como el tamaño del corazón o la separación entre los pulmones. Esto da cuenta de que las características utilizadas finalmente en la clasificación están asociadas a cuestiones anatómicas relevantes.

Existe un gran número de *datasets* médicos (y de diferente naturaleza) que son interesantes para visitar nuevamente utilizando la arquitectura compuesta. Es necesario corroborar si esta arquitectura se equipara en desempeño con las redes DenseNet en los diferentes contextos, o incluso ver si dicha red compuesta puede lograr mejores resultados, tomando en cuenta, además, el valor que agrega la posibilidad de visualizar las características aprendidas por el modelo.

Bibliografía

[1] [Krizhevsky, Hinton, 2012]A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deep convolutional neural networks.* In Advances in neural information processing systems, pages 1097–1105, 2012

[2] [Marvin Minsky, 2017] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[3] [Russakovsky, 2015] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. *Imagenet large scale visual recognition challenge.* International Journal of Computer Vision, pages 115(3): 211–252, 2015.

[4] [Everingham] M. Everingham, S. M. A. Eslami, L. J. Van Gool, C. Williams, J. Winn, and A. Zisserman. *The pascal visual object classes challenge: A retrospective.* International Journal of Computer Vision, pages 111(1): 98–136, 2015.

[5] [Lin, 2014] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and L. Zitnick. *Microsoft coco: Common objects in context.* ECCV, pages (5): 740–755, 2014. 1

[6] [Roth, 2014] H. R. Roth, L. Lu, A. Seff, K. M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, and R. M. Summers. *A new 2.5D representation for lymph node detection using random sets of deep convolutional neural network observations.* In MICCAI, pages 520–527. Springer, 2014.

[7] [Shin, 2016] H. Shin, H. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. Summers. *Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learnings.* IEEE Trans. Medical Imaging, 35(5):1285–1298, 2016.

[8] . [Roth, 2015], Roth L. Lu, A. Farag, H.-C. Shin, J. Liu, E. B. Turkbey, and R. M. Summers. *Deeporgan: Multi-level deep convolutional networks for automated pancreas segmentation.* In MICCAI, pages 556–564. Springer, 2015.

[9] [Ronneberger, 2015] O. Ronneberger, P. Fischer, and T. Brox. *U-net: Convolutional networks for biomedical image segmentation.* In MICCAI, pages 234–241. Springer, 2015

[10] [Jamaludin, 2016] A. Jamaludin, T. Kadir, and A. Zisserman. *Spinenet: Automatically pinpointing classification evidence in spinal mris.* In MICCAI. Springer, 2016. 2

[11] P. Moeskops, J. Wolterink, B. van der Velden, K. Gilhuijs, T. Leiner, M. Viergever, and I. Isgum. *Deep learning for multi-task medical image segmentation in multiple modalities.* In MICCAI. Springer, 2016

[12] [Havaei, 2016] M. Havaei, N. Guizard, N. Chapados, and Y. Bengio. *Hemis:Heteromodal image segmentation.* In MICCAI, pages (2): 469–477. Springer, 2016.

[13] [Deng, 2009] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). *Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009.* CVPR 2009. IEEE Conference on (pp. 248-255). IEEE.

[14] [Mehrotra, 2009] Mehrotra, P., Bosemani, V., & Cox, J. (2009). *Do radiologists still need to report chest x rays.* Postgraduate medical journal, 85(1005), 339-341.

[15] [Bhuvaneswari, 2014] Bhuvaneswari, C., Aruna, P., & Loganathan, D. (2014). *Classification of Lung Diseases by Image Processing Techniques Using Computed Tomography Images.* International Journal of Advanced Computer Research, 4(1), 87.

[16] Q. Dou, H. Chen, L. Yu, L. Zhao, J. Qin, D. Wang, V. Mok, L. Shi, and P. Heng. *Automatic detection of cerebral microbleeds from mr images via 3d convolutional neural networks.* IEEE Trans. Medical Imaging, 35(5):1182–1195, 2016.

[17] [Shin, 2016] H. Shin, L. Lu, L. Kim, A. Seff, J. Yao, and R. Summers. *Interleaved text/image deep mining on a large-scale radiology database for automated image interpretation.* Journal of Machine Learning Research, 17:1–31, 2016. 2

[18] [Pattar, 2015]Pavithra, R., & Pattar, S. Y. (2015). *Detection and classification of lung disease-Pneumonia and lung cancer in Chest Radiology using Artificial Neural Network..* International Journal of Scientific and Research Publications, 5(10).

[19] [Pattar, 2015]LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning.* Nature, 521(7553), 436.

[20] [He, 2016] He., K., Zhang, X., Ren, S., & Sun, J. (2016) *Deep residual learning for image recognition..* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[21] [Huang, 2017] Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2017, July). *Densely connected convolutional networks..* In Proceedings of the IEEE conference on computer vision and pattern recognition (Vol. 1, No. 2, p. 3).

[22] [Zhang, 2017] Zhang, J., Shen, X., Zhuo, T., & Zhou, H. (2017). *Brain Tumor Segmentation Based on Refined Fully Convolutional Neural Networks with A Hierarchical Dice Loss.* arXiv preprint arXiv:1712.09093.

[23] [Wang, 2017] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017, July). *Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases..* In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 3462-3471). IEEE.

[24] [Afshar, 2018] Afshar, P., Mohammadi, A., & Plataniotis, K. N. (2018). *Brain Tumor Type Classification via Capsule Networks.* arXiv preprint arXiv:1802.10200.

[25] [Sabour, Hinton, 2017] Sabour, S., Frosst, N., & Hinton, G. E. (2017) *Dynamic routing between capsules.*. In Advances in Neural Information Processing Systems (pp. 3859-3869).

[26] [Bradley, 1997] Andrew E. Bradley (1997) *The use of the area under the ROC curve in the evaluation of machine learning algorithms.* Cooperative Research Centre for Sensor Signal and Information Processing, Department of Electrical and Computer Engineering, The University of Queensland, QLD 4072, Australia

[27] [Litjens, 2017] Litjens (2017) *A Survey on Deep Learning in Medical Image Analysis.* arXiv preprint arXiv:1702.05747.

[28] [Ko, 2011] Ko (2011) *X-ray Image Classification Using Random Forests with Local Wavelet-Based CS-Local Binary Patterns.* Journal of Digital Imaging.

[29] [Quinlan, 1986] Quinlan (1986) *Induction of decision trees.* Journal of Machine Learning 1. 81-106

[30] [Quinlan, 1992] Quinlan (1992) *c4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc.

[31] [Breiman, 1984] Breiman (1984) *Classification and Regression Trees.* Wadsworth, Mezzovico, Switzerland.

[32] [Gomathi, 2010] Gomathi (2010) *A Computer Aided Diagnosis System for Lung Cancer Detection using Support Vector Machine* . American Journal of Applied Sciences 7 (12): 1532-1538, 2010.

[33] [Ghumbre, 2011] Ghumbre (2011) *Heart Disease Diagnosis using Support Vector Machine.* International Conference on Computer Science and Information Technology.

[34] [Akay, 2009] Akay (2009) *Support vector machines combined with feature selection for breast cancer diagnosis.* Expert Systems with Applications 36 (2009) 3240–3247. Department of Electrical and Electronics Engineering, Cukurova University

[35] [Bhatia, 2008] Bhatia (2008) *SVM based Decision Support System for Heart Disease Classification with Integer-coded Genetic Algorithm to select critical features.* Proceedings of the World Congress on Engineering and Computer Science, San Francisco, USA, pp.34-38, 2008.

[36] [Chen, 2005] Chen (2005) *Combining SVMs with various feature selection strategies.* Department of Computer Science, National Taiwan University, Taipei 106, Taiwan.

[37] [Kadah, 1996] Kadah (1996) *Classification Algorithms for Quantitative Tissue Characterization of Diffuse Liver Disease from Ultrasound Images.* IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 15, NO. 4, AUGUST 199

[38] [Sharawi, 1990] Sharawi (1990) *Quantitative tissue characterization parameters for liver diseases.* Ph.D. thesis, Systems and Biomed. Eng. Dept, Faculty of Eng., Cairo Univ., 1990.

[39] [Breiman, 2001] Breiman (2001) *Random Forests. Machine Learning..* 45:5–32. doi: 10.1023/A:1010933404324.

[40] [Goodfellow, 2016] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaaron (2016) *Deep Learning.* MIT Press. p. 196. ISBN 9780262035613

[41] [Hughes G., 1968] Hughes G. (1968) *On the mean accuracy of statistical pattern recognizers.* IEEE Transactions on Information Theory 14: 55-63.

[42] [Mansilla L., 2018] Mansilla Lucas; Ferrante Enzo (2018) *Segmentación multi-atlas de imágenes médicas con selección de atlas inteligente y control de calidad automático.* CACIC 2018, Tandil, Argentina

[43] [Kingma, 2015] Kingma G. (2015) *Adam: A Method for Stochastic Optimization.* 3rd International Conference for Learning Representations, San Diego, 2015

[44] [Lecun, 1998] Lecun G. (1998) *Gradient-based learning applied to document recognition.* Proceedings of the IEEE (Volume: 86 , Issue: 11 , Nov 1998)

[45] [Andrew Ng, 2017] Andrew Ng. (2017) *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning.* arXiv:1711.05225v3 [cs.CV]