UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESTILACIÓN DE MODELO EN REDES CONVOLUCIONALES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICO

JUAN PABLO RUIZ RODRIGUEZ

PROFESOR GUÍA:
FELIPE TOBAR HENRIQUEZ

MIEMBROS DE LA COMISIÓN:
JORGE SILVA SANCHEZ
JAVIER RUIZ DEL SOLAR SAN MARTIN

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICO
POR: JUAN PABLO RUIZ RODRIGUEZ
FECHA: 2020
PROF. GUÍA: FELIPE TOBAR HENRIQUEZ

DESTILACIÓN DE MODELO EN REDES CONVOLUCIONALES

Se estudia la destilación de conocimiento en clasificación de imágenes usando redes neuronales convolucionales. Se usa a modo de tutor una ResNet101 y a modo de estudiantes ResNet18 junto con MobileNet. Los experimentos se realizan sobre la base de datos Cifar10, sobre la cual se prueban distintas técnicas y configuraciones de destilación, tanto incluyendo información de features como solo usando la información en los logits. Sobre el dataset original, se realizan dos modificaciones para probar el funcionamiento bajo ruido y en datos artificiales. Los resultados son bastante prometedores, lográndose incluso entrenar satisfactoriamente sobre datos exclusivamente sintéticos generados por GAN.

A mi familia, amigos y gente querida.

A todos quienes ayudaron a formar lo que soy.

A quienes se dieron el tiempo de leer y guiar las siguientes paginas.

A quien quiera que sea que inventó la cerveza, la matemática y los computadores, ya que sin ellos nada de esto sería posible.

Tabla de Contenido

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

El campo de investigación y desarrollo conocido como aprendizaje de máquinas (*machine learning*) ha experimentado en esta última década, un crecimiento exponencial en cuanto al conocimiento acumulado. A la fecha, sus técnicas y procedimientos que la han ido posicionando fuertemente en ámbitos como la academia, la industria y el imaginario público, por nombrar algunos. Dentro de este reciente desarrollo, el protagonismo recae en las redes neuronales.

Cabe indicar que los pilares matemáticos sobre los que se erige el aprendizaje de máquinas, así como las redes neuronales no son en sí algo reciente, de hecho, la probabilidad tiene más de 300 años de desarrollo, la estadística más de 200, y el perceptrón, la primera red neuronal de una capa, fue creada en 1958, hace más de 60 años. Al parecer, son el esfuerzo conjunto de la industria y academia a nivel global, la otrora impensable cantidad de datos etiquetados gracias a internet y el perfeccionamiento del hardware y en particular el creciente aumento de las capacidades de las unidades de procesamiento gráfico (GPU) los que han posibilitado el avance en cuanto a las capacidades que tienen las redes neuronales profundas (DNN) modernas.

Desde que en el 2012 AlexNet logró ganar por amplio margen la competencia ImageNet, estas han sido usadas para imitar o superar la capacidad humana de reconocimiento de patrones en audio, procesamiento de imágenes de diverso tipo y las predicciones usando datos estructurados, entre otras tareas. Si bien la razón de estas capacidades sigue en discusión, es un consenso amplio el hecho que esta capacidad de representación de funciones complejas viene dada, por un lado, por la cantidad de neuronas dentro de una capa y por otro, por la cantidad de capas en cascada, una tras otra. Esto permite crear modelos capaces de resolver tareas tan complejas y diversas como el aprender a reconocer las mil clases en que se separa el millón de imágenes de Imagenet, permitiendo con ello generalizar para imágenes naturales no vistas anteriormente.

Un gran problema de estas redes es que la capacidad de aprendizaje que estas tienen, suele requerir de una composición interna del orden de miles de parámetros, los cuales no pueden ser ajustados de manera analítica exacta. La forma más común de realizar esto, al día de hoy, es

aproximarlos, usando descenso de gradiente sobre porciones pequeñas de base de datos (*batches*) compuestas de un centenar de miles de imágenes, cosa que a su vez requiere de una gran cantidad de energía y poder computacional, tanto al momento de entrenar como de operar la red neuronal. Por suerte, existe una cultura común en las grandes empresas y la academia en general, sobre la importancia que tiene compartir los modelos ya entrenados.

Al momento de usar un modelo en el mundo real, este suele ser sometido a distintas técnicas para adaptarlo aún más a la tarea requerida. Ejemplo de estas técnicas es transfer learning, mediante la cual se reutiliza el conocimiento de un modelo para una tarea distinta a la cual fue entrenada. Por lo tanto, en atención a esto último, esta investigación se enfocará en la destilación de conocimiento de un modelo mayor[1] en tanto capas y cantidad de parámetros para el entrenamiento de uno más pequeño, capaz de imitar de manera óptima el comportamiento del modelo original, a fin de obtener un modelo que resulte tanto o más preciso y, a la vez más eficiente.

1.2. Definición de Problema

Una de las razones por las cuales las DNN funcionan bien es que dado una cantidad suficiente de neuronas, estas son capaces de aprender cualquier mapeo de datos con su respectiva etiqueta. Para asegurar que la red sea capaz de generalizar sobre este mapeo aprendido se necesita una cantidad representativa de datos cosa que para problemas complejos, como la clasificación de imágenes, requiere a su vez de una cantidad importante de datos etiquetados y parámetros en el modelo. Por lo mismo, muchas de las redes que muestran mejor desempeño requieren de bastante computación para ser entrenadas o ejecutadas y muchas veces son incapaces de funcionar en un tiempo razonable fuera de un computador de alto rendimiento.

Las herramientas modernas con las que el problema de carga computacional ha sido tratado pueden en

muchos casos reducir notoriamente la cantidad de parámetros induciendo esparcidad en los pesos de las capas, cuantizandolos o realizando destilación de modelos desde modelos grandes a modelos más pequeños. Dentro de las variaciones de la ultima técnica una linea bastante permisora es la del uso de mapas de características [2] como una forma de aprovechar mas el conocimiento de la red original.

Si bien existe una cantidad no menor de estudios al respecto, en general estos no se encuentran suficientemente estandarizados. Al punto que muchos de estos estudios varían no solo en el modelo o el entrenamiento usado si no también en la base de datos. Además, de momento no se ha encontrado un estudio que recopile, sistematice y compare en las mismas condiciones todas estas técnicas.

[1] El término preciso usado por Geoffrey Hinton en [27] es *cumbersome*, traducido como pesado o incómodo. La traducción directa del término al español no refleja muy bien la idea que se intenta representar.

[2] Activaciones de alguna capa convolucional, mas conocidas en la literatura por su nombre en inglés *feature maps* o simplemente *features*.

1.3. Sobre el estado del arte

A estas alturas, es difícil dudar que el aprendizaje profundo funciona a la hora de realizar tareas que anteriormente solo podían hacer los humanos, ahora es tiempo de avanzar en otros aspectos sobre el mismo. Dentro de ellos dos puntos relevantes son por un lado el hacerse cargo del impacto energético que este puede tener y por otro del grado de certeza que se puede tener sobre el funcionamiento de un modelo. La destilación de modelo permite hacerse cargo del primer punto, al ser capaz de reproducir las salidas de una red usando menos recursos. Pero, abre toda una discusión sobre la certeza que se puede esperar de un modelo entrenado ya no directamente sobre datos ordenados, si no más bien sobre la interpretación que un modelo hace sobre esos datos.

Como se ha visto en el marco teórico, la destilación ha sido usada en todo tipo de datos y con una cantidad bastante variada de técnicas, sin embargo no son muchos los intentos por sistematizar estas, ni tampoco existe algún tipo de *benchmark* común sobre el cual comparar estas. Dentro del marco específico de destilación de redes en imágenes usando features por ejemplo, es bastante común a muchas investigaciones no compartir un análisis sistemático sobre las pruebas realizadas, en tanto hiperparámetros, capas y arquitecturas usadas. Además, son bastante pocos los casos que se comparan en igualdad de condiciones con otras investigaciones. Una excepción a esto es [25], donde se propone el esquema presentado en la sección de destilación en features para modelos de clasificación. Fuera de este no se encontró mayor esfuerzo por integrar aprendizajes de otras técnicas en el desarrollo propio. Una tarea de bastante urgente es la de sistematizar estas técnicas, agrupándolas en categorías y proponiendo indicadores comunes mediante los cuales poder comparar investigaciones similares, y es precisamente a esto que se dedicará la presente memoria.

El estudio sobre el estado del arte en destilación se centró en dos líneas. Por una parte se revisó la literatura para ver el estado del arte en destilación, poniendo especial énfasis a técnicas que usan mapas de características dentro del procesamiento de imágenes. Por otra se buscó investigaciones sobre aspectos mas teóricos que pudiesen ayudar a explicar algunos resultados.

Sobre la linea del estado del arte, si bien se encontró cerca de un centenar de investigaciones en todo ámbito y en especial una cantidad importante de ellas en imágenes, se filtró la mayoría de estas y solo se profundizó en las que presentaban técnicas de destilación usando mapas de características y aquellas que fuesen gran importancia por la cantidad de citas o la revista en que fueron publicadas, cosa que se estudió en la sección 2.3. Desde eso se realizó una sistematización de tipos de destilación según objetivo mismo de la destilación y tipos de destilación de mapas de características según tarea objetivo y tipo de adaptación, lo cual puede revisarse en 2.3.3.

Sobre los aspectos teóricos se investigó sobre tres aspectos. El primero guarda relación con las mapas de características de las redes neuronales y en particular con la transferibilidad que estas tienen, para lo cual por suerte se encontró literatura, la cual se expone en 2.2.1. El segundo aspecto 2.2.2, se refiere a la relación entre cantidad de parámetros o capas en una red neuronal y su capacidad de aprendizaje, lo cual a su vez se centró en tres aspectos; la capacidad de aproximación sobre funciones continuas como por ejemplo la salida de una red neuronal mas grande, la capacidad de aproximación lineal por piezas y la capacidad de aprendizaje según cantidad de datos en la base de datos. En tercer lugar en 2.2.3, se profundizó sobre la capacidad de generalización, en particular

que aspectos del entrenamiento y la arquitectura afectan a la red. Al respecto de esto ultimó se intentó relacionar esto con la cantidad de piezas de aproximación lineal, pero al no encontrarse literatura al respecto no se prosiguió por esa linea.

1.4. Objetivos

1.4.1. Objetivo general

Se realizará un estudio sobre la destilación de conocimiento en redes neuronales convolucionales centrándose en la destilación usando transferencia de mapas de características . En particular, se estudiaran las ventajas y desventajas de esta frente al aprendizaje supervisado tradicional y se compararán las distintas técnicas encontradas en la literatura.

1.4.2. Objetivos Específicos

- Buscar resultados teóricos en la literatura que ayuden a explicar los resultados obtenidos en destilación y el funcionamiento de la técnica.
- Estudiar y sistematizar distintas técnicas de destilación usando mapas de características para facilitar su análisis.
- Estudiar la destilación sobre distintos modelos, técnicas de destilación y bases de datos, con el objetivo de cuantificar el efecto de la destilación sobre el nivel de aprendizaje, el nivel de sobreajuste a la base de datos y el tiempo de convergencia.
- Estudiar la destilación sobre bases de datos artificiales, con el objetivo de verificar la factibilidad de reducir la cantidad de datos necesarios para el entrenamiento usando esta técnica.
- Estudiar el efecto del ruido sobre el desempeño de la destilación, con el objetivo de verificar la factibilidad del uso de la destilación ante bases de datos con distintos niveles de degradación.

1.5. Estructura de la memoria

La estructura en que se organiza este documento es la siguiente:

- Capítulo Introducción: Se presentan el tema de estudio, el problema identificado y las interrogantes que se buscará resolver.
- Capítulo Marco Teórico: Se revisan conceptos base de las DNN , se muestran algunos resultados revisados respecto a su capacidad de aprendizaje y generalización y se formalizan

conceptos de *transfer learning*, compresión y destilación y destilación con features necesarios para adentrarse en los resultados. Al respecto de esto ultimo se expone una visión general de las distintas técnicas, las cuales son revisadas en mayor profundidad en la sección de anexos.

- Capítulo Marco Metodológico: Se expone la metodología de trabajo de la memoria, junto con las formas en que se diseñaron los experimentos realizados.
- Capítulo Resultados: Se presentan los resultados de los experimentos realizados junto con los análisis pertinentes.
- Capítulo Conclusiones y Trabajo Futuro: Se hace una discusión de los resultados, evaluando la idoneidad de la metodología propuesta para el estudio del estudio realizado y presentando posibles mejoras.

Capítulo 2

Marco Teórico

Se introducen a continuación, aquellos conceptos y premisas que son relevantes para el desarrollo de esta investigación. En detalle, se comienza describiendo el concepto de *transfer learning* y algunas tareas dentro del procesamiento digital de imágenes. Se sigue con un resumen sobre redes neuronales, redes neuronales convolucionales y otros elementos que hacen referencia a su capacidad de aprendizaje y generalización. Finaliza este capítulo con la introducción de destilación, con foco en la destilación de imágenes con la descripción de algunas técnicas de destilación y la destilación con uso de mapas de características.

2.1. Tareas, Dominios y Transfer Learning

Un buen punto de partida para definir el transfer learning en el contexto de aprendizaje profundo es el que definen en el *survey* [61]. Un dominio se puede definir como $\mathscr{D} = \{\chi, P(X)\}$. Es decir un par compuesto por χ, el soporte o espacio de características y $P(X)$, la distribución probabilidad que genera los $X = \{x_1, \ldots, x_n\} \in \chi$ objetos de la base de datos. Tomando como ejemplo la popular base de datos MNIST, χ podría tratarse de todos los arreglos $\mathscr{Z}_{255}{}^{28\times 28}$ donde $\mathscr{Z}_{255} = \{0 \ldots 255\}$ son todos los enteros representables en 8 bits, X las 70.000 imágenes del base de datos y $P(X)$ todos los dígitos manuscritos que puedan ser escritos en esa resolución.

Por otro lado una tarea se puede definir como $\mathscr{T} = \{y, f(x)\}$, es decir, un par compuesto por un espacio de valores continuos o clases y, y una función objetivo $f(x)$, la cual también puede ser descrita como $P(y|x)$. Siguiendo el ejemplo de MNIST, y correspondería a los dígitos del 0 al 9 y $f(x)$ a la predicción de un modelo sobre este.

Dadas estas definiciones, *transfer learning* corresponde al uso del conocimiento latente del par fuente, dada una tarea fuente $\mathscr{T}_s$ razonablemente resuelta sobre un dominio Ds $\mathscr{D}_s$ y para mejorar el desempeño del par objetivo $\mathscr{T}_t$ $\mathscr{D}_t$. Donde además, $\mathscr{D}_s \neq \mathscr{D}_t$ o $\mathscr{T}_s \neq \mathscr{T}_t$ o ambos son distintos.

El ejemplo más clásico de *transfer learning* es el *fine tunning*, que se suele realizar al

momento de usar una red previamente entrenada sobre una base de datos particular, donde sin necesidad de cambiar la función objetivo, se cambia el dominio y se aprovechan los mapas de características aprendidos en el entrenamiento más general de la red, para una tarea particular.

2.1.1. Nota sobre tareas dentro de procesamiento de imágenes

Cuando se habla de procesamiento de imágenes una base de datos de muestras de un dominio siempre consiste en X imágenes $\chi \in \Gamma^{w,h,c}$. Donde el soporte $\Gamma^{w,h,c}$ corresponde arreglos de un ancho w y un alto h relacionados a las dimensiones espaciales, y un numero de canales de color c, que normalmente puede ser 1 en escala de grises y 3 en casos con color. . Cabe señalar que la tarea puede variar según la salida que se quiera modelar, el objetivo a resolver y las etiquetas disponibles. A continuación se definen algunas tareas.

Clasificación.

Tarea de aprendizaje supervisado en que a cada imagen x_i en la entrada se le asigna un vector de $\hat{y}$ de dimensionalidad n; donde cada posición y_i dentro del vector, designa un grado de pertenencia a alguna clase i dentro de las n clases presentes en las etiquetas de la base de datos. Lógicamente, la clase mas probable de ser es aquella correspondiente a la posición de mayor valor dentro del vector.

Detección.

Tarea supervisada que, además de encontrar una o más clases dentro de la imagen, se debe dar una estimación de una ventana dentro de la imagen (*bounding box*) que indique la posición del objeto detectado. Originalmente, la detección se dividía en dos fases: una de propuesta de posiciones que retornaba ventanas sobre la imagen, y otra en que estas propuestas eran clasificadas individualmente. Gracias a que en el aprendizaje profundo es relativamente fácil incluir muchos objetivos dentro de una misma estructura, los modelos *fully convolutional* más modernos iniciados a partir de Yolo [58], permiten realizar ambas tareas al mismo tiempo. Para esto, se modifica la función objetivo de clasificación, mezclando la regresión sobre el *bounding box*, una predicción sobre que exista un objeto en el *bounding box* y otra sobre la clase que este objeto tendría, simultáneamente.

Segmentación semántica.

Tarea supervisada en la cual el objetivo un mapa de vectores, como los de clasificación, que asignan una clase a cada píxel de la imagen. Esto permite un mayor grado de detalle sobre posiciones de objetos dentro de la imagen que en detección , ya que no sólo hay una estimación de posición, si no también de forma. Es importante notar que en la segmentación semántica solamente se entrega un mapa sin interpretación sobre la imagen. Cuando, además de esto, los píxeles se

agrupan distinguiendo distintos objetos de una o más clases, se habla de segmentación de instancia, concepto que mezcla las características de detección y de segmentación dentro de una misma tarea .

Representation learning, reidentificación, *embedding* y generación.

Junto con las tareas presentadas, existen otras tareas que tienen como objetivo el aprendizaje de características que sean representativas de la imagen, las cuales luego son usadas para otros objetivos. Cuando el objetivo es crear una transformación de imagen a un vector representante, se habla de *embedding*. En aprendizaje profundo, una familia de modelos bastante utilizados para esto es la de los *autoencoders*, donde se usa la misma imagen como objetivo de aprendizaje o etiqueta cuando la tarea consiste en codificar una imagen para después decodificar una versión reconstruida de la misma imagen.

Cuando estos vectores se usan para identificar si una imagen corresponde a un objeto dentro de una base de datos, se habla de reidentificación. En esta tarea, una estructura bastante usada últimamente es la de las redes siamesas, las que buscan que una transformación tenga ciertas propiedades geométricas que hagan más cercanos los embeddings de imágenes de un mismo objeto y más distantes los de aquellos objetos diferentes . Por último, hay casos donde el objetivo es poder generar más muestras de $P(X)$, distintas de aquellas ya contenidas en la base de datos, en algún soporte. Para esto, es bastante común el uso de modelos bayesianos en redes neuronales Gal2015 y por sobre todo, los modelos generativos adversariales [19]. El nombre adversarial en este ultimo proviene de que el entrenamiento se realiza haciendo competir a ambos modelos, en detalle, un generador de imágenes sintéticas aprende a engañar a un discriminador de imágenes falsas, a la vez que el discriminador se vuelve cada vez mejor en distinguir imágenes falsas de imágenes reales.

2.2. Redes Neuronales

Una red neuronal es un modelo computacional paramétrico, vagamente inspirado en las redes que forman las neuronas dentro del cerebro biológico . Cada neurona tiene un peso y un bias característico y recibe una señal proveniente ya sea de una neurona vecina o de un estímulo externo. Según la ubicación que tengan las neuronas, éstas se van agrupando por capas. Una red neuronal de una capa oculta[1] que puede describirse de la siguiente forma:

- Datos $X = \{x_1, \ldots, x_n\}$
- Etiquetas $Y = \{y_1, \ldots, y_n\}$
- Pesos W_1 y *bias* b de la capa oculta y pesos W_2 de la capa de salida.
- Una función de activación compuesta por una operación no lineal σ por capa, como por ejemplo una sigmoidea o una o una función max(0,x), también conocida como rectificador lineal o ReLU [22].

Luego para una dato cualquiera x_i, la predicción $\hat{y}_i$ de una red neuronal de una capa queda descrita de la siguiente forma.

$$\hat{y}_i = \sigma(W_1 x_i + b)W_2$$

Una red compuesta de varias capas toma la siguiente forma, donde para cada capa oculta se tiene un matriz de pesos W_i, un bias b_i[2] y una función de activación σ_i.

$$\hat{y}_i = \sigma_n\left(W_n(\ldots) + b_n\right)W_{n+1}$$

Cuando se dispone de datos etiquetados sobre los que realizar aprendizaje, es decir, en contexto de aprendizaje supervisado hay dos tipos básicos de salidas desde las cuales se van creando salidas más complejas.

- **Regresión** Salida de la red $\hat{y}_i$ en $\mathbb{R}^n$.
- **Clasificación** Vector normalizado de puntuaciones $0 \leq \hat{p}_d \leq 1$, donde $\hat{p}_d$ corresponde a la función softmax sobre la salida de la red $\hat{y}_i$[3]. Este vector también toma el nombre de *logit*.

$$\hat{p}_d = \frac{\exp(\hat{y_d})}{\sum_{d'} \exp(\hat{y_{d'}})}$$

[1] Capa que no es entrada ni salida, o tal que $i < n+1$.

[2] Nótese que incorporar el *bias* dentro de la matriz de pesos W de una capa es tan fácil como agregar una entrada en cada capa de valor 1 e incorporar en cada matriz el valor del *bias*, de esta manera al realizarse la multiplicación este valor simplemente se suma al de los productos de las entradas con el peso. Durante las siguientes páginas se referirá indiscriminadamente a los parámetros de una capa tanto explicitando W y b por separado como solamente usando W.

[3] Para efectos prácticos muchas veces se usa solo la salida de la red, el i máximo después de softmax es el mismo que antes.

Para muchos casos, hoy en día no hay una forma analítica de obtener los pesos que mejor ajusten las predicciones de la red a la base de datos, por lo cual el proceso se hace mediante iteraciones pequeños ajustes sobre particiones mas pequeñas del dataset (*batches*), en vez de la base de datos completa. Para esto generalmente se hace uso de la técnica de *backpropagation*, la cual consiste en usar descenso de gradiente sobre el error en la capa de salida, propagando el error hacia atrás. Para un valor continuo y_i en el caso regresivo o una clase observada d_i en el caso de clasificación, se tienen las siguientes funciones para cuantificar el error:

- **Pérdida de regresión**

$$E^{W_1,W_2,b}(X,Y) = \frac{1}{2N}\sum_{i=1}^{N} || \, y_i - \hat{y_i} \, ||^2$$

- **Perdida de clasificación**

$$E^{W_1,W_2,b}(X,Y) = \frac{1}{N}\sum_{i=1}^{N} log(\hat{p}_{i,d_i})$$

Otra medida interesante de tener en cuenta al ser bastante usada en otras tareas fuera del aprendizaje supervisado más convencional, es la divergencia de Kulback-Leibler, la cual se usa para cuantificar la similitud entre una distribución Q y una P.

- **Divergencia KL**

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathscr{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

2.2.1. Redes neuronales convolucionales y aprendizaje profundo

Tradicionalmente, el machine learning ha consistido en una serie de pasos más o menos estandarizados, utilizados para el procesamiento de los datos ya recolectados y etiquetados. Se comienza por el preprocesamiento de los datos, para eliminar datos incompletos o bien para transformarlos a un formato determinado. Sobre estos datos se extraen características o descriptores usando algún tipo de algoritmo, los cuales son luego usados por un clasificador o regresor para entrenarse y realizar predicciones. Todo este proceso requiere probar varias combinaciones de descriptores y modelos, usando tiempo y conocimiento de un experto, ocasionando un desarrollo por lo general largo, caro y complejo.

El gran avance que permite el uso de aprendizaje profundo , por sobre los demás sistemas tradicionales, consiste justamente en facilitar ese proceso, reemplazando gran parte del diseño y prueba, por el aprendizaje automático de estas características.

Dependiendo del tipo de dato usado en el problema, existen diversas herramientas que se pueden usar, particularmente para el procesamiento de imágenes. La arquitectura que mejor ha funcionado, de momento , es el uso de capas de filtros convolucionales cuyos pesos son neuronas. Estas capas de filtros van formando una serie de módulos que no solo aprovechan la información

que tiene cada píxel, como en una red tradicional, si no que también aprovechan la información espacial de la imagen. La activación convolucional de una capa l es un mapa de características F_l , con información sobre las imágenes de la base de datos, aprendido de manera automática.

Tradicionalmente el machine learning ha consistido en un recetario de pasos más o menos estandarizados para el procesamiento de los datos ya recolectados y etiquetados. Se comienza por el preprocesamiento de los datos para eliminar datos incompletos o bien para transformarlos a un formato determinado, sobre estos se extraen características usando algún tipo de algoritmo o descriptor, las cuales son luego usadas por un clasificador o regresor para entrenarse y realizar predicciones. Todo este proceso requiere de probar varias combinaciones de descriptores y modelos usando tiempo y conocimiento de un experto, cosa que suele ser larga, cara y compleja.

El gran avance que permite el uso de aprendizaje profundo [4], por sobre los sistemas tradicionales consiste justamente en facilitar de ese proceso, reemplazando gran parte del diseño y prueba por el aprendizaje automático de estas características.

Dependiendo del tipo de dato usado en el problema existen diversas herramientas que se pueden usar y en particular para el procesamiento de imágenes la arquitectura que mejor ha funcionado de momento[5] es el uso de capas de filtros convolucionales cuyos pesos son neuronas. Estos van formando una serie de módulos que no solo aprovechan la información que tiene cada píxel como en una red tradicional, si no que también se aprovecha la información espacial de la imagen. La activación convolucional de capa l es un mapa de características F_l con información sobre las imágenes de la base de datos, aprendido de manera automática.

Los filtros son aplicados sobre la entrada realizando una convolución[6] en las dos dimensiones espaciales de la imagen, creando un tensor a la salida de dimensiones $F_l \in \mathrm{R}^{w',h',c_l}$, donde w', h' dependen de el tamaño en la dimensión espacial de los filtros y la forma en que se apliquen, mientras que c_l es la cantidad de filtros aplicados. Consecuencia de esto, al ver una posición i, j en la salida de la capa convolucional $F^l_{i,j}$, el valor del tensor corresponde a la activación de cada filtro frente a patrones detectados en una ventana con las dimensiones w, h del filtro de píxeles dentro de la imagen, tal y como puede notarse en la figura 2.1. Repitiendo esto, la posición en $F^{l+1}_{i,j}$ corresponde patrones detectados dentro de una ventana dentro de F, lo cual a su vez corresponde a una ventana mas grande dentro de la imagen. Además de lo expuesto existen bastantes variantes como por ejemplo las capas de deconvolución, la convolución separable en profundidad, el uso de *stride*, *padding* o dilatación, entre otras[7].

[4]Durante las ultimas semanas ha habido una interesante discusión al respecto de una definición precisa sobre que es aprendizaje profundo y sobre si este debe incluir distintos de las redes neuronales artificiales. Si bien se suscribe a la definición inclusiva dada por Yoshua Bengio en su facebook personal "Deep learning is inspired by neural networks of the brain to build learning machines which discover rich and useful internal representations, computed as a composition of learned features and functions.", cuando se hable en esta investigación de aprendizaje profundo se referirá a las investigaciones hechas con redes neuronales.

[5]Hay ciertos ejemplos interesantes de otros modelos como por ejemplo *Deep Forest* [76]. De momento ninguno funciona al nivel de las redes neuronales.

[6]En la practica la operación que se implementa en software es la de correlación cruzada, ya que convolución implica girar el kernel, cosa que en este caso no se hace. De todas maneras, como los filtros son aprendidos, probablemente en caso de usarse convolución propiamente tal el filtro aprendido seria el mismo solo que girado.

[7]Un review bastante completo al respecto se encuentra en [13].

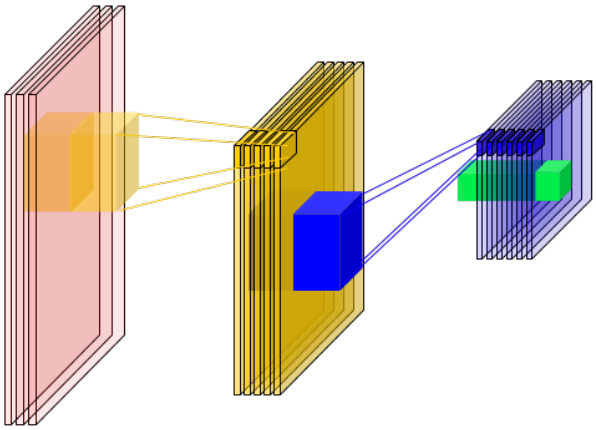

Figura 2.1: Composicion de capas Convolucionales.

Es importante tener en cuenta que, si bien en esencia la aritmética de fondo en las convoluciones es similar a la de las redes neuronales clásicas, el hecho de que se restrinja la cantidad de conexiones de cada neurona en la capa, favorece el uso de tensores y reduce la cantidad de parámetros, cosa que a su vez causa que la computación de estas capas sea mucho más eficiente en las modernas unidades de procesamiento gráfico. Además, el uso de convoluciones tiene un efecto positivo en la generalización del modelo, al dar cierto nivel de invariabilidad frente a las traslaciones de objetos dentro de la imagen y al reducir la cantidad de parámetros necesarios para aproximar un modelo, a lo cual se le debe sumar la regularización, producto de la profundidad del procesamiento.

mapas de características como representaciones de patrones en la imagen

Una particularidad interesante de los mapas de características , es que estos representan información sobre qué patrones son los que ve la red neuronal en la imagen. Como puede verse en la imagen 2.2 las primeras capas convolucionales se activan con información de colores y texturas de una manera similar a como funcionan los descriptores anteriormente usados en *machine learning*. A medida que las capas van procesando estos patrones de activación, las activaciones de las neuronas pueden proyectarse sobre zonas de la imagen con mayor sentido semántico, ya sea como zonas de un objeto o objetos determinados.

El uso de mapas de características puede resultar de bastante utilidad a la hora de adaptar un modelo entrenado a una base de datos o una tarea distinta. Una técnica bastante común a la hora de realizar detección, por poner un ejemplo, es la de entrenar una red para clasificar y reutilizar las capas convolucionales, cambiando la parte final de la red para detectar. De esta manera, puede aprenderse los mapas de características usando las etiquetas de clasificación, que son más fáciles de obtener que las de detección, disminuyendo de este modo el costo total del entrenamiento.

Al respecto de esto, en [73] se investigó el grado de transferibilidad de mapas de

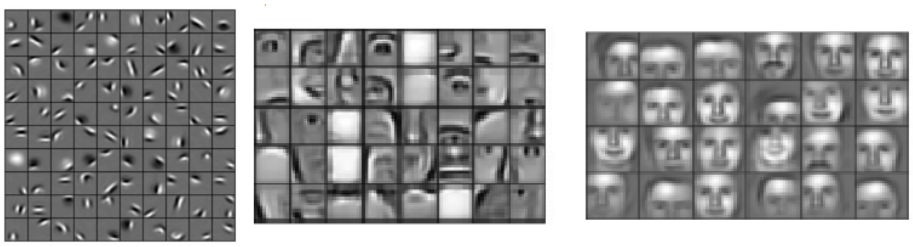

Figura 2.2: Zona de la imagen a la que apunta una activación según profundidad de la red.

características al cambiar de base de datos sin cambiar de tarea. Se usó la arquitectura *ResNet* entrenada sobre alguna base de datos y congelando los las capas convolucionales a distintas profundidades para entrenar sobre otra base de datos. Se encontraron 3 resultados importantes.

- A mayor profundidad, los mapas de características son más específicos a el dominio y tarea sobre los que fueron entrenados. En las capas finales, esto puede corregirse un poco, gracias a la abstracción de la misma, pero sin lograr la versatilidad de las primeras capas.
- Mientras más distintos sean los dominios, menos compatibles serán los mapas de características de las redes. Por ejemplo, para una red entrenada con imágenes de la naturaleza, el accuracy final va a ser mayor si se ajusta a otros objetos naturales que si se ajusta a objetos hechos por el hombre.
- El reentrenamiento de la red completa suele ser suficiente para mejorar cualquier problema de adaptación.

2.2.2. Sobre la capacidad de aprendizaje de las redes neuronales

La tarea de aprendizaje de un modelo puede entenderse como la acción de encontrar una función hipotética que describa la etiqueta para todos los ejemplos posibles de un objeto sobre un soporte. Por ejemplo, todas las imágenes de perros en una resolución y profundidad de color determinados. Lógicamente, nadie tiene acceso a tal función, por la sencilla razón de que nadie puede tener todas las fotos de perros del mundo, y aun si las tuviera, la cantidad sería demasiado grande y probablemente improcesable por los computadores de los que disponemos actualmente.

Por esta razón, en machine learning, esta $f(x)$ se aproxima usando las muestras de P(X) disponibles en la base de datos. Existen bastantes estudios al respecto que permiten dar cotas analíticas al error, según la cantidad de capas o neuronas en una red. Se describe a continuación lo encontrado en la literatura con respecto al caso no convolucional, tanto profundo como no, a través de tres distintas formas de enfrentarse al problema.

Capacidad de aproximación de funciones continuas

Con respecto a esta propiedad, un resultado fundamental de las redes neuronales es el de [11] que muestra en 1989 que una red de una capa oculta sin límite de neuronas de activación sigmoideas aproxima cualquier función continua con un error arbitrario, lo cual se demuestra luego para el caso multicapa con activación no constante en [28]. En [55], [54], [46] entre otros, se extiende el resultado para las activaciones ReLU, actualmente más usadas sobre todo en las capas convolucionales.

Saltando al año 2017, en [46] se logra dar un límite a la cantidad de neuronas necesarias por capa para una red de activación ReLU profundidad arbitraria. En detalle, cualquier función Lebesgue integrable en $\mathbb{R}^d \to \mathbb{R}$ puede ser aproximada por una red de ancho de al menos $d+4$ con un error arbitrario con respecto a la distancia L_1. Por otro lado, ninguna función Lesbegue integrable en en $\mathbb{R}^d \to \mathbb{R}$ de medida no 0 es aproximable con una red de ancho de capa menor a d.

Con respecto a la profundidad necesaria, si bien no se obtiene una cota cerrada, si se logra establecer que el aumento de la profundidad provoca un incremento logarítmico de la eficiencia de la red. Para un entero k, existe una red ReLU de ancho del orden $O(k^2)$ y profundidad 2 que no puede ser aproximada por ninguna red de ancho de orden $O(k^{1.5})$ y profundidad k. Cosa que se refuerza con lo estudiado en [63] donde se llega a que una red de $\Theta(k^3)$ capas de $\Theta(1)$ unidades representa mas que una de profundidad $\Theta(K)$ y menos $\Theta(2^k)$ unidades por capa.

Similarmente, en [23] se ve que sin cota máxima de profundidad, cualquier función en el cubo $[0,1]^d$ es aproximable con un error arbitrario si el ancho de capa es al menos $w_{\min}(d) \leq d+2$. En el caso de convexas positivas bastan $d+1$ neuronas y si la función es afín por parte, puede ser representada exactamente con $d+3$ neuronas de ancho. Sobre la profundidad necesaria en estos casos, fijando un ancho cercano al $w_{\min}(d)$ especificado, cualquier función que es afín por partes, con N partes, puede ser representada exactamente por una ReLU de ancho $d+1$ y profundidad N

Capacidad según cantidad de piezas de aproximación lineal

El modelo que crea ReLU es una aproximación lineal por piezas de la función que se intenta imitar. De esta manera, una forma interesante de medir la capacidad de la red es contando la cantidad de piezas con las que se puede aproximar.

En una red de una capa, cada neurona separa el espacio de entrada en 2. En el caso multicapa, cada capa realiza una especie de operación OR que se propaga hacia adelante. El teorema de Zalavsky dice que para una cantidad finita de m hiperplanos en un espacio n_o-dimensional común (arreglo $\mathscr{A}$), la cantidad total de regiones es $r(\mathscr{A}) = \sum_{s=0}^{n_0} \binom{m}{s}$ y la cantidad de regiones delimitadas es $b(\mathscr{A}) = \binom{m-1}{n_0}$. Para una red de una sola capa oculta, con n_0 entradas y n_1 unidades en la capa oculta, la cantidad de piezas de aproximación lineal está dado por el arreglo de n_1 hiperplanos en el espacio n_0 dimensional $\mathscr{R}(n_0, n_1, n_y) = \sum_{j=0}^{n_0} \binom{n_1}{j}$. Considerando n_0 $O(1)$ entradas y kn unidades en una sola capa, el valor es del orden.

$$O(k^{n_0-1} n^{n_o-1})$$

Mientras que, para un modelo de k capas ocultas de ancho n, el numero máximo de regiones de aproximación tiene la siguiente cota.

$$\Omega\left(\left\lfloor\frac{n}{n_0}\right\rfloor^{k-1}\frac{n^{n_o-2}}{k}\right)$$

Un aspecto importante a considerar, es que siempre que hayan más neuronas que unidades en la entrada, la cantidad de regiones va a crecer exponencialmente con la cantidad de capas, aumentando a su vez el detalle con el que se va a aproximar la función buscada. Además, debe siempre tomarse en cuenta que estas regiones de activación no son totalmente independientes unas de otras, dentro de una misma capa, lo cual puede ser un impedimento para la aproximación exacta de una función. Sin embargo, es necesario recordar que no disponemos de una función exacta que represente el objetivo, sino de muestras de ésta, por lo que esta deficiencia es bastante útil a la hora de asegurar la generalización de la red.

Muestreo discreto

En general la expresividad de un modelo se mide en un espacio continuo, normalmente para una familia determinada de funciones. En [75] proponen una forma de medir expresividad mas aplicable a tareas sobre una base de datos. Resumiendo, para cualquier base de datos de n muestras en d dimensiones una red ReLU de 2 capas con $2n+d$ unidades puede aprender cualquier etiquetado, hasta etiquetas aleatorias. Por otro lado una red de profundidad k necesitará $O(n/k)$ parámetros para realizar la misma tarea.

En resumen

1. Las redes neuronales en general son capaces de aproximar cualquier función o mapeo.
2. Con una red de 2 capas es posible memorizar cualquier base de datos de n muestras con un ancho de $2n+d$, para aproximar cualquier función continua en $(0,1)^n$ y afín por K partes en $\rightarrow [0,1]^d$ basta una red ReLU de ancho $d+1$ y una profundidad K.
3. La cantidad de regiones de aproximación lineal aumenta de forma exponencial según aumenta la cantidad de parámetros (y la profundidad de la red).

2.2.3. Generalización en redes neuronales

Dado que el aprendizaje en *machine learning* se realiza sobre una cantidad acotada de muestras de un proceso al cual no siempre se tiene acceso total, una pregunta siempre presente es si acaso el modelo aprendido va a ser capaz de extender sus predicciones a ejemplos fuera de la base de datos, característica conocida como generalización. Dependiendo del enfoque que el investigador tome y el modelo utilizado hay varias herramientas que permiten mejorar el

desempeño de un modelo, las cuales se enfocan en general en la regularización del $f(x)$ aprendido, cosa de evitar que un sobreajuste a los puntos de la base de datos de entrenamiento, a costa empeorar la predicción en el resto del espacio de entrada.

Salvo algunos estudios aislados como [32] no existen muchas herramientas teóricas que permitan asegurar límites robustos sobre la capacidad de generalización en aprendizaje profundo . Como se expone en el mencionado estudio, ciertos cuantificadores del machine learning clásicos, complejidad Rademacher o dimensión V-C tienden a relacionar una mayor cantidad de parámetros y capacidad de aprendizaje con una menor capacidad de generalización. Al entrar al terreno de aprendizaje profundo , estos resultados son aparentemente contradictorios ya que las DNN son capaces de generalizar relativamente bien a pesar de componerse de cantidades desde miles a millones de parámetros. Según los resultados empíricos de la investigación, influye mas la arquitectura y la cantidad de datos en la base de datos que la cantidad de parámetros misma. Aparentemente el hecho mismo de que estos parámetros se distribuyan en muchas capas permite suavizar el $f(x)$ aprendido.

Desde el lado mas empírico del estudio de este fenómeno en [75], se explica la causas de la regularización puede provenir de dos categorías generales. . Por un lado están aquellas causas implícitas en el uso de convoluciones, el procesamiento por capas y particularidades de la arquitectura de la red neuronal, junto con aquellas causas que surgen naturalmente del procesamiento por batches, al entrenar mediante descenso de gradiente estocástico. Por otro lado están aquellas explicitas, es decir, aquellas que no son parte inherente del sistema y son agregadas aparte para mejorar la regularización, tales como las que van directo sobre los valores de los pesos como la regularización l_2[8], las que inducen ruido estocástico dentro de los pesos o la entrada, como el *dropout* y los métodos diseñados para aprovechar mejor la base de datos, como son las múltiples variantes de *data augmentation* que inducen modificaciones sobre los datos de entrada según el diseño del programador.

En miras de explicar empíricamente como cada uno de estos afectan a la generalización, entrenaron de manera efectiva una DNN sobre bases de datos con distintos niveles de aleatorización y distintos tipos de regularizadores. Si bien en todos los casos es posible para una red aprender perfectamente la base de datos, la capacidad de generalización depende por sobre todo de la cantidad de datos no corruptos presentes. Con respecto a la regularización explicita, esta si bien puede ayudar a mejorar la generalización, no suele ser suficiente para explicar la generalización de la red ni para controlarla, por el contrario, los cambios en la arquitectura pueden tener un impacto mucho mayor.

2.3. Compresión de Modelo y destilación

Como se vio en la sección anterior, detrás de la alquimia[9] del aprendizaje profundo , las sustancias elementales son los datos, los modelos basados en múltiples capas de muchos

[8] Se agrega a la pérdida un regularizador L_2 sobre los distintos parámetros para favorecer un comportamiento más suave de la red y una mejor generalización. $\hat{\mathscr{L}} = E^{W_1,W_2,b}(X,Y) + || W_1 ||^2 + || W_2 ||^2 + || b ||^2$

[9] https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

parámetros y el poder computacional para poder entrenarlos. Si bien el funcionamiento del hardware moderno es mucho más eficiente que hace algunos años y mantiene una tendencia a mejorar, estas arquitecturas pueden seguir siendo en muchos casos muy pesadas para aplicaciones masivas reales. Además de esto, el poder computacional usado en entrenamiento y ejecución no es inocuo, llegando a tener en algunos casos tanto impacto en el medio ambiente como el que pueden tener 5 autos durante toda su vida útil.[10]. Es deseable encontrar una forma de obtener modelos con la capacidad de esas arquitecturas grandes en modelos más pequeños.

La literatura al respecto es bastante amplia y existen bastantes *reviews* y librerías casi *plug and play* como Tensorflow-lite o pyTorch Mobile que facilitan esta tarea. En [10] las técnicas existentes actualmente se dividen 4 clases; *low-rank factorization* correspondiente a la descomposición de la matriz para estimar parámetros informativos, *parameter pruning and sharing* centrados en la reducción de parámetros redundantes y no informativos, *transferred and compact convolutional filters* consistente en el uso de estructuras convolucionales diseñadas para ahorrar el número de parámetros, y *knowledge distillation* consistente en el entrenamiento de una arquitectura más pequeña usando lo aprendido por una arquitectura más grande.

Esta ultima técnica se desarrolla por primera vez en el trabajo de Caruana et al. [3] centrado en la obtención de un modelo único desde un ensamble de modelos distintos y continua en [2] del mismo autor, donde se reproduce la salida de una red profunda usando una red de pocas capas usando distancia L_2. Sin embargo, la técnica no se hace popular hasta *Knowledge Distillation* de Hinton et al. en [27], donde lo que se realiza es la transferencia de una red tutora T de un mayor tamaño a una red estudiante mas pequeña S [11]

2.3.1. Destilación de conocimiento en redes neuronales usando logits

El caso presentado en [27] KD en adelante, se centra exclusivamente en la imitación de los logits en la capa de salida de una red neuronal, aprovechando que típicamente en el caso de clasificación la última capa usa una activación *Softmax* que produce puntuaciones sobre las clases normalizados en 1. De esta forma la red estudiante se entrena sobre la predicción de la red tutora a la vez que sobre las etiquetas originales, usando una versión de softmax que incluye un valor T de temperatura.

$$\hat{q}_d = \frac{\exp(\hat{z_d}/T)}{\sum_{d'} \exp(\hat{z_{d'}}/T)}$$

Denotando Z_S a la salida de la red estudiante antes de los logits y Z_T a la de la red tutora, el valor final de la perdida es[12].

$$\mathscr{L}_{KD} = D_{KL}[q(Z_S), q(Z_T)]$$

[10] https://www.technologyreview.com/s/613630/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

[11] Interesantemente, esta formulación es bastante similar al *privileged information* de Vapnik aplicado a *support vector machines* [**?**]

[12] Sobre $q(Z_S)$ suele usarse un log para hacer el entrenamiento mas fácil, sin afectar en mayor manera la convergencia.

Al comparar las diferencias de este caso con *cross entropy*, es interesante notar que en una etiqueta normal de clasificación toda la información disponible es *crisp*, es decir solo se sabe cual es la categoría a la que pertenece el dato. Por el contrario, la predicción que se obtiene de la red tutora es *soft*, en el sentido que entrega información no solo de cual es el valor real si no de a que otra categoría se parece. Luego, asumiendo que la red tutora es una aproximación lo suficientemente fiable de un clasificador $f(x)$ perfecto[13], la tarea de entrenar un clasificador pasa de ser la de aprender datos crudos desde un dataset de K datos, a la de aprender la función mapeo del espacio de entrada en $\mathbb{R}^d$ al espacio de etiquetas en $[0,1]^n$, $f' : \mathbb{R}^d \to [0,1]^n$.

Volviendo un poco a 2.2.2, es sabido que una red con ReLU aproxima creando una función continua, convexa y afín por partes. Luego, la salida de la red tutora puede ser aproximada usando una red estudiante ReLU de ancho $d+1$ y alguna profundidad k, necesitando del orden de $O(dk_S)$ parámetros para aproximar esta . La red tutora tuvo que usar del orden $O\left(\frac{K}{k_T}\right)$ parámetros para aprender los K datos usando k_T capas. Tanto la dimensionalidad de entrada d como el numero de capas k_S y k_T son valores constantes y acotados, mientras que la cantidad de datos K es un valor que, en aprendizaje profundo suele ser varios ordenes de magnitud mayor[14].

2.3.2. Algunos ejemplos de destilación

Luego de publicado [27] el 2014, comenzaron a surgir una serie de investigaciones usando el concepto, llegando incluso en momentos a hacer difusa la línea entre lo que es destilación y transfer learning. Dentro de lo revisado en literatura hay dos grandes categorías que sin ser disjuntas entre ellas y posiblemente sin englobar el total de la literatura disponible, pueden describir las tareas en las que han ido decantando las investigaciones al respecto. La primera se refiere a la tarea ya descrita de comprimir un modelo aprendiendo desde la función que genera una red tutora. La segunda, centra sus esfuerzos en aprender realizar la destilación sin necesidad de usar una base de datos, solo aprovechando el conocimiento implícito en los pesos y arquitectura de la red tutora. Se describen algunos ejemplos interesantes encontrados en literatura.

Dentro de aquellas investigaciones orientadas a la eficiencia, la destilación ha sido usada en imágenes para detección de caras [21], *visual question answering* [51] y detección [57], [60], este ultimo usando una mezcla de mapas de atención y el jacobiano de la red. En vídeo, se ha usado para detección activa [14], anticipación de acción humana [64], destilación de atención [41] y destilación de acción en grafo [47]. Cambiando de tipo de dato, [30] y [45] ha sido usada para destilación en secuencias, mientras que [1] ha sido usada en adaptación de modelos en audio.

Dentro de destilación orientada a los datos, se pueden distinguir dos niveles de ausencia de estos dentro del proceso. [40], [34], logran destilar reduciendo notoriamente las muestras necesarias. Por otro lado, [52], [72], [44], [7] y [49] centran su esfuerzo en destilar sin usar ningún dato real. Por otra parte [67] realiza destilación no sobre un modelo si no de la base de datos misma

[13] O en otras palabras, si el modelo tutor funciona bien.

[14] Es difícil saber a ciencia cierta cuanto debe ser el valor de K. Lo importante en realidad es tener una colección lo suficientemente representativa del problema, cosa que depende de la complejidad que tiene el f_x que se intenta aproximar. En casos simples como algunas regresiones lineales pueden bastar algunas decenas, mientras que en casos complejos pueden llegar a requerirse centenas de miles.

directamente desde un modelo entrenado.

2.3.3. Destilación usando mapas de características

Como se vio en la sección 2.2, los mapas de características representan patrones que la red neuronal aprende sobre el dominio en que se entrenó y como tales pueden ser de bastante utilidad a la hora de realizar transfer learning. En destilación de modelo, la primera investigación al respecto es [59], la cual aplica el mismo método de [27] pero lo complementa con *hints* de los mapas de características de la red tutora F_T para pre-entrenar la red estudiante.

Destilación de mapas de características en clasificación

Comenzando por aquellas investigaciones referidas a la tarea de clasificación, la mayoría de las pérdidas de las técnicas revisadas en literatura comparte un esquema propuesto en [25] y consistente en tres elementos:

- Una transformación T_T sobre F_T.
- Una transformación T_S sobre F_S.
- Una medida de distancia o disimilaridad d entre $T_T(F_T)$ y $T_S(F_S)$

e esta manera, el entrenamiento puede realizarse conjuntamente, por fases separadas o de manera *greedy*, es decir, entrenando capa a capa y usando una fórmula que toma la forma.

$$\mathscr{L}_{\text{feature}} = d\left(T_T(F_T), T_S(F_S)\right)$$

Para profundizar en el rol de cada uno de estos módulos es bueno recordar que cada capa convolucional produce un tensor mapa de características en $\mathrm{R}^{c,w,h}$ donde c es el numero de filtros y w, h son el ancho y alto resultante de la convolución, considerando el tamaño del filtro y *padding, stride* y otras especificidades de la convolución. En general la elección de capa desde la cual extraer los mapas de características viene de dos condiciones, ambas capas corresponden a un bloque que comparte la misma dimensionalidad en w,h y estas corresponden a la última capa de ese bloque. Luego, hay que considerar que el numero de filtros necesarios para las redes tutora y estudiante es bastante posible que sea distinto al momento de entrenarse sobre la misma tarea y seguramente va a ser distinto si se cambia la base de datos sobre el que se va a entrenar. Por esto mismo se hace necesario el uso de las transformaciones T_T y T_S para adaptar las dimensionalidades de ambas redes. Estas varían según investigación, agrupándose en general en 3 categorías; mediante red neuronal, mediante reducción matemática y mediante transferencia de activación o borde[15].

[15]Dentro de lo leído en literatura dos investigaciones que escapan a este esquema son [6] que usa reducción a nivel de batch y [8] que usa una estructura tipo GAN.

Tabla 2.1: Resumen de destilación usando mapas de características en clasificación, detalle de cada investigación en anexos.

Key	año	Técnica	Tipo	T_T	T_S	Distancia
[26]	2019	Match de bordes de activación	Activacion	Binarizacion	conv 1x1	Hinge
[25]	2019	Match de activacion positiva	Activacion	Relu Marginal	conv 1x1	Partial L2
[59]	2015	Hints	Red neuronal	-	conv 1x1	L2
[33]	2018	Match de factores	Red neuronal	Auto-encoder	Auto-encoder	L1
[37]	2019	Auxiliary learning structre	Red neuronal	ALS	ALS	L2
[71]	2017	Flow of procedure	Reducción	Correlacion	Correlacion	L2
[74]	2017	Mapas de atencion	Reducción	Mapa de atencion	Mapa de atencion	L2
[56]	2018	Match de probabilidad (PKT)	Reducción	KDE	KDE	KL
[31]	2019	MMD sobre mapas de características (NST)	Reducción	Kernel (MMD)	Kernel (MMD)	L2
[6]	2018	Match de LFS	Reducción*	-	-	-
[8]	2018	GAN sobre feature maps	Otros	-	-	-
[60]	2018	Match de jacobianos	Red neuronal	Jacobiano	Jacobiano	L2

En las transformaciones vía red neuronal la medida de distancia entre los mapas de características de cada red suele ser directamente una distancia tipo L_p tomándose por lo general $p = 2$ o 1. La transformación entonces cumple el doble propósito de adaptar el tensor F_S en $\mathrm{R}^{c_S,w,h}$ a $\mathrm{R}^{c_T,w,h}$ para ser comparable con F_T y de realizar una transferencia del gradiente del error. Las transformaciones suelen ir en la red tutora y consisten en cualquier estructura convolucional entrenable en simultáneo o separadamente al entrenamiento de la capa final. En el caso de [59] por ejemplo, se usa un regresor de $1x1$ y c_T canales, lo cual tiene la particularidad de que al mismo tiempo que hay una transferencia de la magnitud del error, el cuello de botella que se crea al reducir la cantidad de filtros desde F_T a F_S provoca que el regresor tenga un efecto similar al que tiene PCA,comprimiendo la información a lo mas relevante de F_T.

Cuando hay reducción matemática de los mapas de características , T_T corresponde a alguna función $\mathrm{R}^{c_T} \rightarrow \mathrm{R}^{c}$ y T_S a alguna $\mathrm{R}^{c_S} \rightarrow \mathrm{R}^{c}$ que aplicadas en cada canal a lo largo y ancho de estos permiten reducirlos a una dimensionalidad en común c. La distancia en este caso suele ser tipo L_p, salvo algunos métodos cuyas transformaciones producen distribuciones de probabilidad, en cuyo caso se usa la divergencia d_{KL}. Las transformaciones revisadas varían desde el uso de un kernel MMD [31], la correlación entre canales [71], mapas de atención [74] y [60] a un kernel de estimación de densidad [56]. En este caso el efecto que tiene la transformación escogida varía de caso a caso, permitiendo un diseño mas explicito sobre la información que quiere escogerse de F_T que en el categoría anteriormente presentada.

La ultima categoría, transferencia de activación o borde, se diferencia con respecto a los casos anteriores en que se busca imitar la región donde hay respuesta en el espacio de entrada, para lo cual se crean modificaciones a nivel de la activación. Para la reducción se usa o bien una transformación tipo regresor de 1×1 o ningún tipo de adaptación. Dentro de la literatura se encontraron dos ejemplos; transferencia de borde [26] y activación positiva modificada [25]. La primera usa un regresor sobre F_T, binarizando ReLU para tomar valor 1 en caso de activación y 0 en caso contrario, usando a modo de distancia *Hinge Loss*[16] como en SVM. Siguiendo esa misma línea, la segunda usa un regresor con activación ReLU de margen negativo[17], junto con lo cual se

[16] Detalles sobre esto en anexos.

[17] $\sigma_{\mathrm{MReLU}} = max(x,m)$ en vez del $max(x,0)$ del ReLU original. m se calcula por batch o época como la media de las respuestas negativas de la iteración previa.

usa una distancia L_2 modificada para retornar 0 en todos aquellos casos que $F_{S_i} \leq F_{Ti} \leq 0$. Vale la pena mencionar en este grupo también [74], que dentro del estudio de destilación con mapas de atención usa una variante donde obtiene un mapa de atención, no de las activaciones, si no del gradiente de estas.

Destilación de mapas de características fuera de clasificación

Fuera de clasificación, es difícil organizar estos métodos ya que además de la información general de los mapas de características de la imagen. Cada una de las tareas tiene sus especificidades que deben ser tratadas por separado según tareas y en ocasiones incluso según arquitectura, razón por la cual no es tan fácil encontrar un esquema común de destilación. En cuanto a las tareas objetivos en destilación usando mapas de características se encontró investigaciones en detección, segmentación semántica, reidentificación y una bastante particular cuyo objetivo era la imitación directa de los mapas de características de la parte convolucional de la red tutora. Se omiten algunos casos encontrados donde el rol de la destilación era poco claro o aquellos referidos a destilación en vídeo.

Se comenzará con el caso de la segmentación semántica, por ser el único que no guarda ninguna relación con otras investigaciones. En [43] se mezclan 4 perdidas distintas para aprovechar tanto las etiquetas, como las predicciones de una red ya entrenada. A nivel de cada píxel de la imagen se usa el *cross entropy* entre la predicción de la red y la etiqueta real, a lo que se le suma a nivel de píxel la perdida KD de [27]. Sumado a esto esto se usa la distancia entre los mapas de activaciones de cada píxel y un discriminador que actúa a modo de embedding, usando la red de segmentación como un generador. Sobre la salida de este ultimo se usa distancia wasserstein.

Con respecto a la destilación de mapas de características en arquitecturas de detección, es importante notar que dependiendo del tipo de objetivo con que se entrene la red, la salida va a variar en la forma en que codifica la predicción de los objetos, sus *bounding boxes* y otros valores como por ejemplo *objectness* o saliencia. Hay dos formas de transmitir la información desde la salida de la red. La primera es imitar la salida de la red directamente desde el tensor en crudo, modo que puede ser mas fácil de implementar, al costo de requerir que la salida de la red estudiante sea idéntica a la de la red tutora. Dentro de esta linea se encuentra por una parte [48] que crea un modelo estudiante desde YOLO y por otra [38] junto a [9][18] que destilan desde la familia de R-CNN. La otra forma de transmitir esta información consiste en usar la predicción de las redes de una manera procesada, es decir, usar las predicciones ya transformadas desde el tensor de salida a sus respectivas *bounding box* y clases para entrenar una red. Este es el caso de [66], donde además de realizarse destilación usando hints, se aprovechan las ventanas detectadas para crear una mascara y destilar solo las partes importantes.

Un detalle importante en este caso, es que en todos estos casos el desbalance entre la cantidad de píxeles que corresponden a fondo y los que corresponden a objeto, puede ocasionar que la

[18] Siendo un poco mas riguroso con esto, ni [48] ni [38] realizan destilación de mapas de características como objetivo si no mas bien como consecuencias de imitar ciertas partes que usan las redes respectivas al predecir *bounding boxes* y objectness (RPN en R-CNN y el header *fully convolutional* en YOLO). [9] por el contrario integra directamente *hints* como los de [59] dentro de a destilación.

destilación no funcione bien, al obligar a la red a reproducir mapas de características sobre partes de la imagen que no es necesario saber, cosa que se enfrenta de manera directa en [9] y [66]. Por otra parte, en [48] el modelo YOLO incorpora dentro de su predicción el *objectness* de las ventanas, cosa que se aprovecha para balancear el peso de las predicciones y clases innecesarias dentro de la pérdida.

Todas las restantes técnicas revisadas comparten el objetivo de buscar reproducir el mapa de característica completo de la red a nivel de alguna capa. En [17], [42] y [8] el mapa de característica se reproduce en la ultima capa convolucional, mientras que en [16] se reproduce en la red completa. Ge2018 y Liu2018 concuerdan en usar GAN para imitar los mapas de características la red tutora y mejorar la predicción del header de la red, además de tener una tarea distinta de la imitación directa de los mapas de características . Por otra parte [8] usa una estructura generativa para entrenar un clasificador a la vez que un discriminador, [42] prueba para un detector y un clasificador mientras que [17] usa la estructura generativa para generar muestras que se suman a toda una estructura diseñada para desacoplar la pose del *embedding* de personas en la tarea de reidentificación. Finalmente, en [16] entrenan la red reproduciendo bloque por bloque los mapas de características de la red tutora. Para esto, usan la distancia L_2 en el primer bloque de la red tutora y la red estudiante, luego congelan los pesos y entrenan el nivel siguiente y así sucesivamente, hasta llegar al header donde simplemente entrenan usando *ground truth* y la tarea original del modelo.

2.3.4. Uso de GAN y otras técnicas probabilísticas

Una red neuronal bayesiana es una en que los pesos de sus neuronas θ en vez de ser números puntuales, están determinados por alguna distribución de probabilidad. En [15] se demostró[19] que el uso de las técnicas de regularización estocástica tales como dropout o la inclusión de ruido gaussiano aditivo, además de mejorar la generalización en el entrenamiento permiten usar un modelo de manera probabilística en inferencia. Esta técnica tiene bastantes ventajas, entre ellas mejorar aun mas la generalización y permitir cuantificar la incerteza de modelo. Al costo de aumentar bastante el tiempo de inferencia al requerir de aproximar vía integración monte-carlo, es decir, mediante el cálculo de la esperanza sobre varias muestras de la salida de la red manteniendo la técnica de regularización activada. A continuación se describen casos encontrados de esta técnica junto a otras técnicas probabilísticas como las redes generativas adversariales[20].

Dentro del aprendizaje profundo bayesiano [4], pone la primera piedra al destilar la esperanza de una red bayesiana. Para conseguirlo, entrena usando KD sobre la media de varias muestras de los *logits* de una red bayesiana, logrando una leve mejora con respecto a la base determinista en ambos casos. Esto es completado mas tarde en [20] donde se logra destilar la incerteza junto con la esperanza de modelo. En [65] se usa además GAN junto con *Markov Chain Monte Carlo* [18] para destilar la posterior completa del modelo.

Con respecto al uso de GAN para casos no bayesianos, en [69] se emplea en clasificación de imágenes y en [70] imágenes usando la arquitectura *conditional GAN*, además de [8] que lo usa

[19] Se desarrolla un poco mas esto en anexos, 6.2.

[20] En esta sección debería incluirse también investigaciones que usen VAE sin embargo no se encontraron

junto con destilación de mapas de características . [68] lo usa para detección de objetos, con el fin de aprovechar mas las características del dominio. En otras tareas, [17] lo usa en reidentificacion y [62] lo usa en destilación sobre vídeos. Por otra parte, fuera de aprendizaje supervisado [42] lo usa en aprendizaje semi supervisado y [49] en *zero shot* transfer.

Capítulo 3

Marco Metodológico

Como fue mencionado anteriormente, el objetivo de la memoria es el de realizar un estudio sobre el funcionamiento de la destilación usando mapas de características para tareas de clasificación, probando distintos modelos, técnicas y condiciones. Al respecto de esto, se implementaron tres experimentos pensados para responder a tres preguntas distintas.

- Con el fin de cuantificar el efecto de la destilación y realizar una comparación con el aprendizaje supervisado y entre distintos tipos de técnicas, se comenzó con la destilación sobre la base de datos Cifar 10, usando distintos modelos, técnicas de destilación y configuraciones.
- Se genero 3 tipos distintos de datasets artificiales, mediantes los cuales se probó el desempeño de cada técnica en ausencia de datos reales.
- Se estudió el efecto del ruido en el dataset de entrenamiento sobre el desempeño de las distintas técnicas de destilación.

Todo esto permite abrir paso hacia realizar una ortogonalización de las distintas configuraciones que tienen las técnicas, con el fin de dar lineamientos hacia que aporta cada tipo de perdida en logits o mapas de características , cuanto afectan la temperatura o la capa destilada y cuanto afecta el dataset y el modelo al desempeño final de la técnica.

Finalmente, y de manera bastante superficial, se probaron algunas de estas técnicas sobre parte del dataset Imagenet y modelos de mayor complejidad.

Sobre lo que sigue del capitulo, este se separa en 2 secciones, una dedicada a la descripción de algunos aspectos generales de la implementación de los experimentos, y una dedicada a la descripción en detalle de cada experimento.

3.1. Implementación

Bases de datos

Para la mayoría de los experimentos se usó la base de datos Cifar-10 [36], compuesta de 60.000 imágenes de 32x32 píxeles RGB etiquetadas en 10 clases balanceadas mas 10,000 imágenes de similares características para test. Para el entrenamiento se hizo uso de *data augmentation* vía uso de cortes y giros aleatorios . Además, tanto en entrenamiento como en test el procesamiento tensorial de pytorch convierte los valores enteros de RGB en valores fraccionarios entre $(0, 1)$, a lo que se agregó una normalización a nivel de píxeles.

Se generaron tres bases de datos usando tres técnicas distintas; ruido (*noise*), un Autoencoder Variacional Convolucional (VAE) [12] y un Red Generativa Adversarial con Clasificación Auxiliar (GAN) [53]. La primera consistió en el uso de un arreglo de gaussianas de potencia 1 y de la misma dimensionalidad de Cifar10. Para la segunda técnica se usó un VAE consistente en un *encoder* de 4 capas convolucionales seguidos de una capa *fully connected* con un 128 unidades en la dimensión latente, desde la cual naturalmente se muestrean 128 gaussianas, mientras que en la salida se usó un *decoder* consistente en 3 capas de convolución transpuesta y 4 capas *fully connecterd*. La tercera técnica consistió en un GAN compuesta de un generador condicional[1] consistente en una *fully connected* que recibe ruido y la clase que se quiere generar seguida de 4 capas de convolución transpuesta y un discriminador de 4 capas convolucionales con *batch normalization* y *leaky ReLU*[2] seguidas de dos capas *fully connected* paralelas, una para discriminar entre imágenes verdaderas y falsas como en un ReLU normal y otra para determinar la clase a la que pertenece la imagen. Ambas redes se entrenaron sobre Cifar-10 por un numero fijo de épocas y luego se tomó la época que generara las imágenes con apariencia mas real. Ambas redes fueron adaptadas de versiones libres puestas a disposición por usuarios de Github[3]. Para ambos casos se muestrearon 100.00 imágenes generadas desde semillas aleatorias sobre los modelos para generar las bases de datos.

Para el caso con ruido se usó la misma base de datos sobre la que se entrenó y se agregó ruido en entrenamiento.

Modelos utilizados

Sobre CIFAR10 se usó como red tutora una red ResNet-101 [24], la cual como su nombre lo indica, tiene 101 bloques residuales y *batch normalization* como de red tutora. Como de estudiantes se usó dos redes, una ResNet-18 y una Mobilenet [29], de 13 módulos de convolución separable en profundidad, ambas con *batch normalization*. Todas estas redes fueron entrenadas por 400 épocas usando *cross entropy* sobre la base de datos sin modificaciones, para tener un punto de comparación contra el cual evaluar la destilación. Como puede verse en la tabla 3.1, la diferencia en desempeño

[1] Un en que además del ruido de entrada de cualquier GAN, se recibe la clase de la imagen que se quiere generar.

[2] Una versión de ReLU donde en vez de tomar valor 0, la parte negativa toma una pendiente positiva muy pequeña. Se diferencia de ReLU en que además de rectificar transmite de mejor manera el gradiente en la parte negativa.

[3] https://github.com/chaitanya100100/VAE-for-Image-Generation y https://github.com/King-Of-Knights/Keras-ACGAN-CIFAR10.

base entre las redes Resnet no es de mas del 0.01 % de accuracy, mientras que la diferencia con Mobilenet es de cerca del 3 % en test y 12 % en entrenamiento. Por otro lado, la reducción en cantidad de parámetros es de cerca de 4 veces de Resnet-101 a Resnet-18 y de 13 veces para Mobilenet. La reducción en peso en memoria, contando parámetros y el espacio usado para el tensor de computación es de unas 5 veces para Resnet y de 13 veces para Mobilenet.

Tabla 3.1: Características de redes utilizadas en Cifar-10.

Modelo	Resnet-101	Resnet-18	Mobilenet
Capas	209	41	55
Capas Convolucionales	104	20	27
Parámetros	42,512,970	11,173,962	3,217,226
Entrada [MB]	0.1	0.1	0.1
Peso [MB]	262.31	53.89	20.25
Peso de Parámetros [MB]	162.17	42.64	12.27
Peso de Tensor en Forward [MB]	100.13	11.25	7.97

3.1.1. Aspectos técnicos de la implementación

En una primera fase se realizaron algunas pruebas preliminares sobre MNIST usando Tensorflow, sin embargo al cambiar a las pruebas sobre mapas de características en CIFAR-10 se decidió cambiar todo a PyTorch, por la facilidad que ofrece para integrarse naturalmente con python. Esto se aprovechó para crear una librería, reutilizable entre las distintas bases de datos, arquitecturas y funciones de perdidas. Para esto se partió desde el trabajo de dos fuentes, una librería de modelos canónicos en la literatura facilitada por un usuario de github[4] y el código fuente de algunas investigaciones de destilación que algunos de los autores fueron lo suficientemente amables de compartir. Todo esto junto con las notas sobre los papers leídos quedó guardado y documentado en el repositorio abierto https://github.com/jpcosec/Memoriahttps://github.com/jpcosec/Memoria.

Para la extracción de las mapas de características se consideraron dos opciones de diseño. La primera consiste en extraer estas previamente desde la red tutora y guardarlas en una base de datos en el disco duro, para luego cargarlas en memoria al momento de entrenar la red estudiante. La segunda consiste en extraer estas bajo demanda, es decir, cargar en memoria la red tutora junto con la estudiante y obtener la mapa de característica realizando la computación simultáneamente en ambas. En principio, la primera opción debiese ser mas rápida y eficiente energéticamente ya que la computación de la red tutora debe realizarse una sola vez, sin embargo el tamaño de la base de datos resultante puede ser demasiado grande. Dado que además de pruebas sobre el dataset puro se hicieron pruebas que incluían ruido, se priorizó la implementación usando la segunda opción. No se hicieron pruebas empíricas para comparar la eficiencia de ambas opciones.

Todo el entrenamiento se realizó en un computador equipado con una GPU Nvidia GeForce RTX 2080. Se usó un optimizador Adam con tasa de aprendizaje de 0.001 en batches de

[4]https://github.com/kuangliu/pytorch-cifar

entrenamiento de 128 a excepción de algunos casos donde fue necesario bajar el tamaño de batch.

3.2. Destilaciones usadas y sus configuraciones

Estudio de destilación de logits

Recordando que para la salida de una capa de *logits* cualquiera, softmax modificado con una temperatura T es:

$$q_i = \frac{exp\,(z_i/T)}{\sum_j exp\,(z_j/T)}$$

La perdida KD es:

$$\mathscr{L}_{KD} = D_{KL}\,[q(Z_S), q(Z_T)]$$

Y KD_CE, denotando CE como *cross entropy*, y_{GT} como las etiquetas reales y λ como un coeficiente para balancear entre ambas perdidas es:

$$\mathscr{L}_{KDCE} = D_{KL}\,[q(Z_S), q(Z_T)] + \lambda \mathrm{CE}\,(Z_s, y_{GT})$$

El primer experimento consistió en probar dos tipos de entrenamiento, KD y KD con *cross entropy* sobre solo la ultima capa para tener una base y comparar los tipos de arquitectura, el uso o no información de las etiquetas originales y el hiperparámetro T en la destilación. T varió en un rango de 1 a 100.

Estudio de destilación usando mapas de características

Para el estudio de destilación en mapas de características se separó la perdida en dos partes. $\mathscr{L}_{\mathrm{F}}$, correspondiente a la destilación de mapas de características y $\mathscr{L}_{\mathrm{L}}$ correspondiente a la perdida de logits. Luego la perdida total se calculó como la suma directa y sin ponderación de ambas.

$$\mathscr{L} = \mathscr{L}_{\mathrm{F}} + \mathscr{L}_{\mathrm{L}}$$

$\mathscr{L}_{\mathrm{L}}$ varió según prueba, usándose tanto KD como KD_CE, siempre con la temperatura de la destilación fija en 8. Por otro lado, para $\mathscr{L}_{\mathrm{F}}$ se implementaron 4 de las técnicas expuestas en el marco teórico, las cuales pueden verse resumidas en el cuadro 3.2.

Tabla 3.2: Perdidas de destilación con mapas de características usadas.

Paper	Perdida	Distancia	T_T	T_S
[59]	Hints	L_2	-	r
[74]	Mean_Att	L_2	$\sum_{i=1}^{C} \mid A_i \mid^2 \frac{1}{C}$	$\sum_{i=1}^{C} \mid A_i \mid^2 \frac{1}{C}$
[74]	Max_Att	L_2	$\text{máx}_{i=1,c} \mid A_i \mid^p$	$\text{máx}_{i=1,c} \mid A_i \mid^p$
[56]	PKT	D_{KL}	$\text{KDE}_{\cos}$	$\text{KDE}_{\cos}$
[31]	NST_lin	MMD	K_{lin}	K_{lin}
[31]	NST_poly	MMD	K_{poly}	K_{poly}

Siguiendo la categorización propuesta en 2.3.3, las perdidas implementadas corresponderían a un solo caso de adaptación por red neuronal y 6 casos de adaptación vía reducción matemática[5]. En el caso de *Hints* [59], se usa una capa regresora de 1×1 R para la adaptación. En [74] se usan dos versiones de mapa de atención, una basada en la media del cuadrado de las activaciones por canal y otra que toma simplemente la máxima activación. PKT [56] usa un kernel coseno $K_{\cos} = \frac{1}{2}\frac{a^\top b}{\|a\|_2\|b\|_2} + 1 \in [0]$ de estimación de densidad a través de los canales para crear una distribución que aproxima vía divergencia KL.

$$\text{KDE} = \frac{K_{\cos}(F_t^i F_t^j; 2\sigma_t^2)}{\sum_{k=1,k\neq j}^{N} K_{\cos}(F_t^i F_t^j; 2\sigma_t^2)}$$

Y finalmente NST [31] que usa como perdida un estimador de MMD, sobre 2 kernels[6] distintos aplicados en la dimensión c del tensor mapa de característica ; lineal $K_{\text{lin}}(X,Y) = XY^\top$, polinomial de $d = 2$ y $c = 0$ $K_{\text{poly}}(X,Y) = \left(XY^\top\right)^2$ Para mas detalle sobre cada una y algunas mas de estas técnicas se dejó en anexo una revisión un poco mas detallada sobre algunas técnicas de destilación en mapas de características 6.

Se destiló en todos los casos usando una capa a la vez. Se eligió para este fin la ultima capa de cada bloque, es decir, la ultima de todas las capas de la misma dimensionalidad en w,h, las cuales son para ResNet-101 4 capas. Se usó todas las perdidas de la tabla para todos los bloques en cada caso. En la tabla 3.3 se puede ver los detalles se los experimentos realizados.

Tabla 3.3: Setting experimental de destilación usando mapas de características .

	Épocas	Destilación logits	T	Preprocesamiento
Cifar10	50	KD, KD_CE, CE	8	*Data augmentation*, Normalización
Ruido	50	KD	8	Normalización , Ruido aditivo de SNR en grilla de 0.1 a 1
GAN	50	KD	8	*Data augmentation*, Normalización
VAE	50	KD	8	*Data augmentation*, Normalización

[5]Los casos que involucraban modificaciones en la activación de las redes, si bien estaban documentados con código abierto distaban mucho del esquema utilizado. Se se prefirió priorizar aquellas perdidas que requirieran menos tiempo para implementarse.

[6]Se usó también un kernel gaussiano pero el tiempo de ejecución de este ultimo era demasiado alto por lo que se uso solo en un experimento

Indicadores medidos

Se consideran en general 3 valores interesantes que medir,

- La diferencia del accuracy en validación entre la red tutora y los modelos destilados: Mide cuanto logra aprender el método en la configuración usada.
- La diferencia del accuracy en validación entre el modelo entrenado en CE y los modelos destilados. Mide cuanto mejora el desempeño del modelo con respecto a su propia base usando el método.
- El ratio entre el accuracy en entrenamiento y en validación del modelo destilado: Mide cuanto *overfitting* presenta el modelo.
- Época en que se alcanza el accuracy de validación mas alto o la perdida mas baja: Permite dar una estimación vaga de cuando el modelo logra converger.

Capítulo 4

Resultados

Se exponen a continuación los resultados de los experimentos descritos en el capitulo de metodologías. Se analiza cada experimento por separado para luego comenzar a comparar entre técnicas y experimentos. En algunos gráficos de accuracy se pueden ver lineas de color en uno o mas ejes, estas corresponden a los valores base del modelo entrenado usando cross entropy.

4.1. Experimentos sobre Cifar 10

4.1.1. Entrenamiento de redes con cross entropy

Lo primero que se hizo fue el entrenamiento del modelo a ser entrenado como red tutora y los modelos estudiantes usando el dataset sin ninguna modificación y CE como perdida por 400 épocas. Los datos medidos en esta fase se adjuntan en la siguiente tabla, donde ΔT es la diferencia contra el valor obtenido en CE por ResNet101 y test/train es el ratio entre el accuracy de entrenamiento y el de validación .

Tabla 4.1: Valores medidos sobre entrenamiento de modelos en cross entropy.

Modelo	Epocas	Epoca de perdida minima	Epoca de Acc. maxima	Δ T	test/train
ResNet101	400	339	391	0.0	0.940566
ResNet18	400	360	303	-0.6	0.934840
MobileNet	400	391	306	-4.0	0.903520

De la tabla es posible ver que se alcanzan en todos los casos valores relativamente altos y no tan distantes entre si de accuracy en la base de datos de entrenamiento. Además, es posible notar un buen nivel de ratio nivel bastante cercano a 1 en el ratio entre el accuracy de entrenamiento y validación. Los valores mínimos de perdida y máximo de accuracy se se obtuvieron durante el ultimo tramo de entrenamiento, sin embargo, tal y como es posible apreciar en el gráfico siguiente estos valores se estabilizan bastante durante las primeras 100 épocas de entrenamiento. Al ver estos

en tanto tiempo es posible notar que el tiempo de entrenamiento para ResNet101 es unas 3 veces mayor que el tiempo de entrenamiento en los otros modelos.

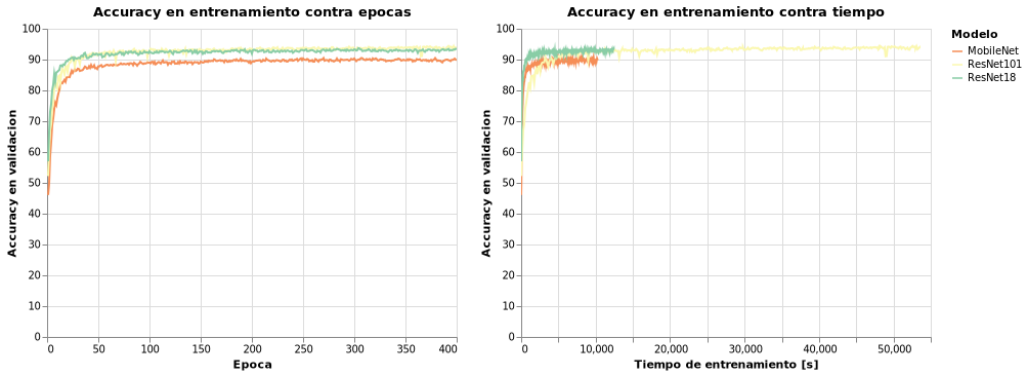

Figura 4.1: Entrenamiento de modelos usando cross entropy

4.2. Destilación de Logits

El segundo paso del experimento fue la destilación de la ResNet101 entrenada usando ambos modelos estudiantes, MobileNet y ResNet18. Se usó perdida KD y KD con CE probando distintos valores de temperatura en la perdida de destilación.

Modelo	D. logits	ΔS
ResNet18	KD, 10	-0.25
MobileNet	KD, 10	-0.33
ResNet18	KD, 8	-0.43
MobileNet	KD, 8	-0.52
ResNet18	KD, 5	-0.69

Tabla 4.2: Mejores resultados sobre logits segun diferencia con CE

Modelo	D. logits	ΔT
ResNet18	KD, 10	-0.85
ResNet18	KD, 8	-1.03
ResNet18	KD, 5	-1.29
ResNet18	KD_CE, 1	-1.63
ResNet18	KD_CE, 50	-2.06

Tabla 4.3: Mejores resultados sobre logits segun diferencia con tutor

De la observación de la tabla de los mejores según cantidad de aprendizaje con respecto a la red tutora es posible notar que en los mejores casos se logra una diferencia bastante leve con respecto a lo que es capaz de de aprender el modelo por si solo desde los datos crudos usando CE. Viendo el gráfico de accuracy con respecto a la red tutora es posible notar que además de esto, los casos que usan ResNet18 suelen obtener valores mas altos que aquellos que usan MobileNet. Además, en general el uso de CE junto con KD no beneficia mucho el resultado final. Sobre la temperatura, es posible notar que dentro de los valores bajos el rango no varía mucho, pareciendo ser que los casos mas cercanos a 10 son los que presentan mejores resultados. Interesantemente, una temperatura mas alta parece mejorar el ratio entre entrenamiento y validación, sin embargo esto parece relacionarse mas la perdida sufrida en el accuracy de test que con una mejora general del algoritmo.

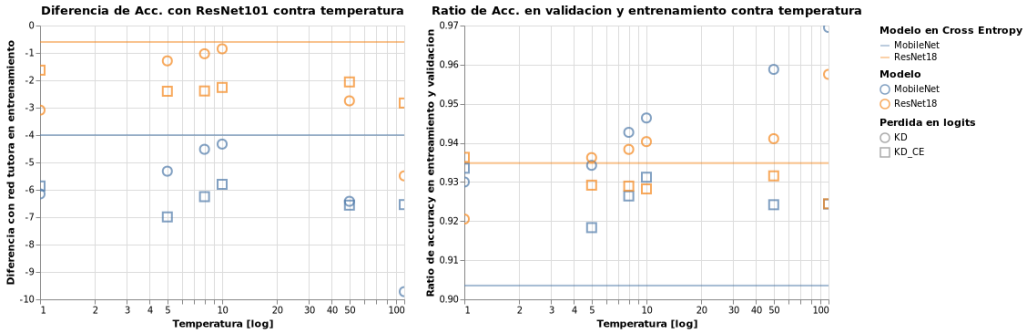

Figura 4.2: Aprendizaje con respecto a tutor y sobreajuste en destilacion de logits

De la observación del segundo par de gráficos salta a la vista en primer lugar que en el caso de las diferencia de accuracy del modelo con respecto a su base los valores para ambos modelos son bastante cercanos. Además, una vez mas el uso de CE dentro de la perdida parece no afectar positivamente el valor dentro de la zona de mejor funcionamiento. Con respecto al tiempo de convergencia, no es posible ver que ningún hiperparametro del experimento, ni el modelo, ni la perdida ni la temperatura afecte positivamente.

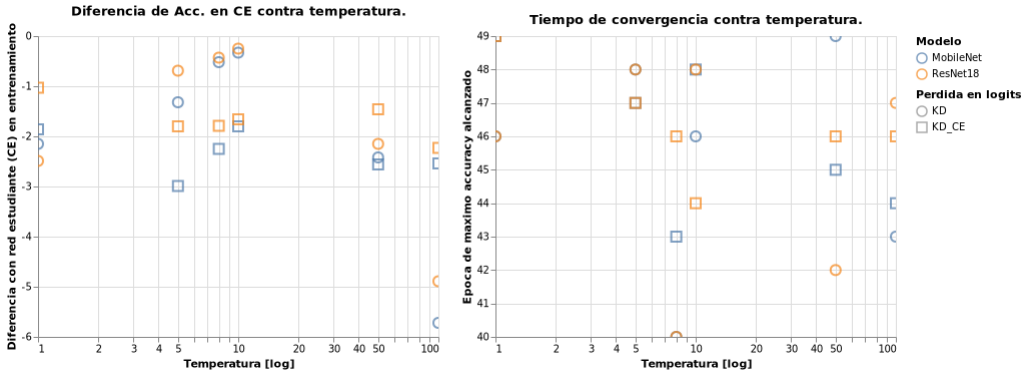

Figura 4.3: Aprendizaje con respecto a modelo estudiante y convergencia en destilacion de logits

4.2.1. Destilación usando mapas de características

Una vez fijado el valor de temperatura en 8, se procedió a probar los distintos tipos de destilaciones de features implementados.

Modelo	D. logits	D. Conv.	ΔS
MobileNet	KD, 8	hint, 1	0.68
MobileNet	KD, 8	nst_poly, 1	0.55
ResNet18	KD, 8	nst_linear, 2	0.42
MobileNet	KD, 8	nst_linear, 3	0.39
MobileNet	KD, 8	att_mean, 3	0.26

Tabla 4.4: Mejores en mapas de características segun diferencia con CE

Modelo	D. logits	D. Conv.	ΔT
ResNet18	KD, 8	nst_linear, 2	-0.18
ResNet18	KD, 8	att_mean, 3	-0.47
ResNet18	KD, 8	PKT, 1	-0.53
ResNet18	KD, 8	hint, 1	-0.55
ResNet18	KD, 8	att_max, 3	-0.65

Tabla 4.5: Mejores en mapas de características segun diferencia con tutor

Lo primero que salta a la vista es que al contrario del caso de destilación solo en logits, donde todos los casos estudiados obtienen menos incremento contra la base que pone el modelo estudiante entrenado en CE, en este hay 8 casos que lograron superarse, sin alcanzar a superar a la red tutora. Al ver la diferencia de accuracy obtenido con respecto a la red tutora, es notorio que ResNet18 funciona mejor en todos los casos que MobileNet, cosa que se nota mas aun al mirar el ratio de entrenamiento y validación, donde los únicos casos en que este empeoró fue en los que usaban MobileNet. Aparentemente mas importante aun que el tipo de destilación usada es el tipo arquitectura.

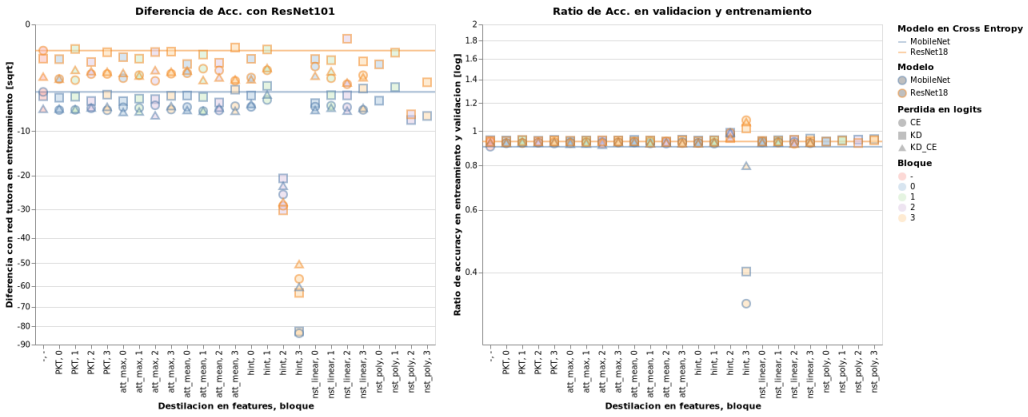

Figura 4.4: Aprendizaje con respecto a tutor y sobreajuste en destilación de mapas de características

Interesantemente, y como se puede ver en el gráfico de abajo, los casos de empeoramiento con MobileNet presentan también un tiempo menor para alcanzar el accuracy máximo obtenido, el cual de todas maneras es varios ordenes de magnitud mas bajo que el del resto de los casos.

Al parecer el incremento contra su propia base experimentado no depende tanto del modelo, por el contrario, influye mas el tipo de destilación en logits usada y la combinación destilación de features y bloque de destilación utilizado. Dentro de estos últimos es posible notar que Hints

funciona sostenidamente mal en las ultimas capas t sorprendentemente bien en la 2a, al igual que NST polinomial. Por el contrario, los mapas de atención y NST lineal funcionan mejor en las ultimas capas, mientras que PKT funciona de manera regular en todos los casos. En general, el uso de KD solo en logits funciona mejor que su combinación con CE o el uso solo de esta ultima.

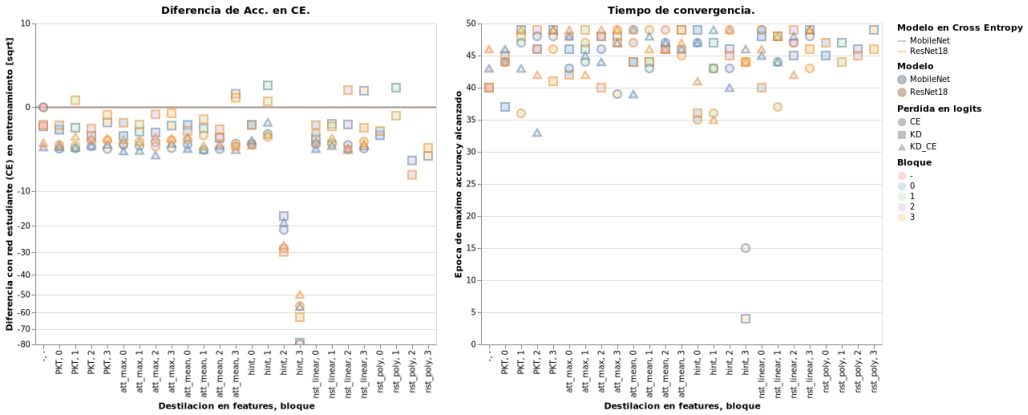

Figura 4.5: Aprendizaje con respecto a modelo estudiante y convergencia en destilacion de mapas de características

4.3. Destilación sobre bases de datos sintéticas

A diferencia de el aprendizaje supervisado regular, donde toda la información disponible para el aprendizaje proviene de los datos mismos, en destilación además del dataset de entrada, se dispone del conocimiento latente que contiene la red ya entrenada. La tarea cambia entonces desde aprender desde una base de datos a aprender desde las características que parámetros la salida o los mapas de características de la red. Una diferencia importante con respecto al caso original es que este permite usar el espacio de entrada fuera de los puntos que contiene el dataset, cosa que podría permitir por ejemplo hacer posible un entrenamiento sin necesidad de usar el dataset, cosa que podría ser ampliamente provechosa en casos en que este no se encuentre disponible.

Al respecto de esto, se probó tres casos de datasets sintéticos, uno basado en *auxiliary classifier* GAN, uno basado en VAE y uno consistente de ruido gaussiano de varianza 1. Solo en el caso de *auxiliary classifier* GAN si se dispone de una etiqueta al momento de generar la muestra. Como perdida de ultima capa solo se usa KD de temperatura 8, en las features se usó distintos tipos de destilación sobre distintas capas.

Modelo	D. Conv.	dataset	ΔS
MobileNet	hint, 1	cifar10	0.68
MobileNet	hint, 1	GAN	0.54
ResNet18	nst_linear, 2	cifar10	0.42
MobileNet	nst_linear, 3	cifar10	0.39
MobileNet	att_mean, 3	cifar10	0.26
ResNet18	att_mean, 3	cifar10	0.13
ResNet18	nst_linear, 2	GAN	0.12

Tabla 4.6: Mejores en datos sintéticos según diferencia con CE

Modelo	D. Conv.	dataset	ΔT
ResNet18	nst_linear, 2	cifar10	-0.18
ResNet18	att_mean, 3	cifar10	-0.47
ResNet18	nst_linear, 2	GAN	-0.48
ResNet18	PKT, 1	cifar10	-0.53
ResNet18	hint, 1	cifar10	-0.55
ResNet18	att_max, 2	GAN	-0.61
ResNet18	att_mean, 3	GAN	-0.65

Tabla 4.7: Mejores en datos sintéticos según diferencia con tutor

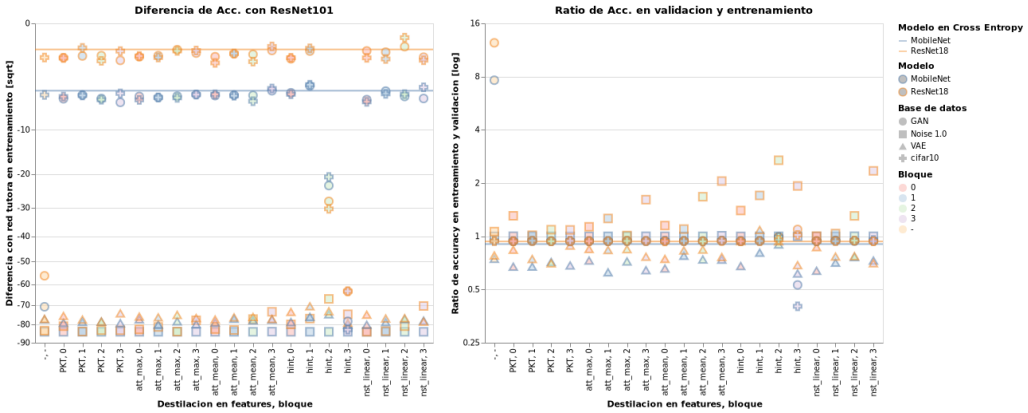

Figura 4.6: Aprendizaje con respecto a tutor y sobreajuste en destilación sobre datos sintéticos

El primer detalle interesante de notar es que en las tablas de mejores resultados ordenados según el la diferencia con respecto al tutor, la mitad de los mejores 10 son casos entrenados usando datos sintéticos generados con GAN. Comparando por configuración de capa y bloque en cada gráfico es posible notar que los resultados sobre los datos generados con GAN suelen seguir la tendencia que presentan estas sobre Cifar10, llegando en algunas pocas ocasiones incluso a superarse el valor con respecto al entrenar sobre datos reales y a la red tutora y, salvo excepciones, funcionando bien en las configuraciones que el caso real, y mal donde el caso real funciona mal. Sin embargo, al observar el desempeño sobre el caso con destilación solo en logits, es fácil notar que esta tendencia no se repite en absoluto, alcanzándose valores varias decenas de puntos mas bajos para ambos modelos. Además, el desempeño sobre las otras bases de datos no es comparable al desempeño logrado con GAN, llegándose a valores en torno a los 80 puntos de accuracy menos, siendo el valor de VAE ligeramente mayor.

De los comportamientos anteriores es posible desprender dos conclusiones. En primer lugar, es posible notar que mediante destilación no es posible aprender etiquetas aleatorias, tal y como hace notar la ausencia de un crecimiento muy grande en el *overfitting* en el gráfico del ratio entre accuracy en validación y entrenamiento y el hecho de que muchos casos entrenados sobre ruido dejaron de aprender a tan temprana época. Esto da indicios de que en destilación es bastante importante la vecindad sobre la que se trabaja, es decir, no es posible transferir el conocimiento de una red a otra sobre todo el espacio de entrada, pero, si se puede de manera bastante satisfactoria

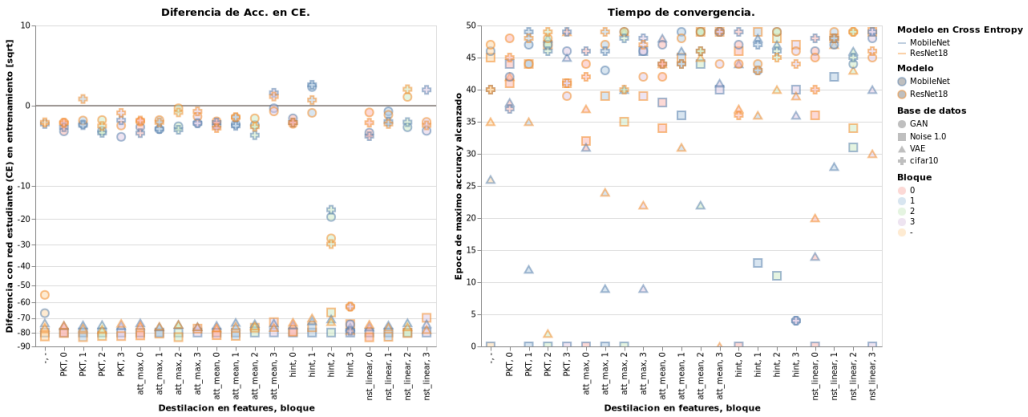

Figura 4.7: Aprendizaje con respecto a modelo estudiante y convergencia en destilación sobre datos sintéticos

sobre el espacio de las imágenes similares a la base de datos de entrenamiento donde se entrenó la red tutor. En segundo lugar, la información que la transferencia de mapas de características da, compensa la información que se pierde al usar imágenes no reales, cosa que se puede notar al comparar el desempeño sobre datos con colores similares pero sin forma como son los generados con VAE con los datos con formas similares que genera GAN y al notar el mal comportamiento en todos los casos que tiene la destilación sin features.

4.4. Destilación sobre datos con ruido

El ultimo experimento realizado fue la destilación sobre un dataset con distintos niveles de corrupción. Para esto se entrenó añadiendo ruido gaussiano de potencia 0 a 1 sobre Cifar10 al destilar sobre la red tutora entrenada en el dataset origina, sin ruido. Se destiló con perdida KD en los logits y distintos tipos de destilación de features sobre distintas capas.

Para estudiar el efecto del ruido se obtuvo las curvas de decaimiento para el accuracy en test comparado con el valor obtenido por la red tutora sin ruido. Dada la cantidad de distintas combinaciones de métodos de destilación y capas, se decidió resumir la información añadiendo a los gráficos una curva mas gruesa con la media y una banda de color con el intervalo de confianza del 95 %. Así, se presentan 4 graficos, uno por modelo reduciendo los datos por tipo de perdida de features y otro por modelo por bloque. En el caso de la destilación sin uso de mapas de características , naturalmente la curva se corresponde con la media y la banda de confianza tiene ancho 0[1]

Ambos modelos son capaces de aprender bien con un ruido de hasta 0.1, resistiendo incluso hasta 0.2 en el caso de ResNet18. Luego de esto la curva decae rápidamente hasta 0.6 en para

[1]El análisis se realizó usando la librería de visualización Altair en un jupyter notebook. La mencionada librería permite una implementación extremamente simple de gráficos dinámicos, los cuales pueden están disponibles online en el repositorio online de la memoria.

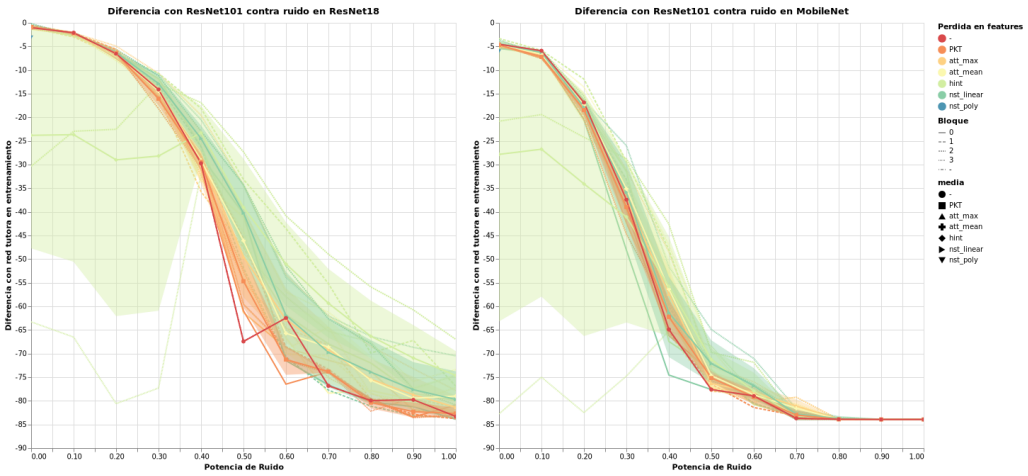

Figura 4.8: Decaimiento de destilación según cantidad de ruido, resumido por tipo de destilacion

ResNet18 y hasta 0.4 para MobileNet. A partir de esto la curva se suaviza, alcanzándose niveles estables dependientes del caso para ResNet18 y un nivel mínimo para MobileNet a partir del ruido de potencia 0.8. Esta zona parece responder al nivel de ruido donde el modelo ya deja de ser capaz desde la red tutora y depende fuertemente del nivel base que alcanza la destilación en logits sin uso de features.

Comparando los tipos de destilación, el tipo que alcanza niveles mas altos es Hints usando las primeras capas. Al mismo tiempo el modelo que sufre el decaimiento mas rápido es Hints usando las ultimas capas, cosa que probablemente se relaciona con el nivel de especificidad que estas toman hacia el final. Los otros tiposf de destilación sufren un poco mejor al que sufre el caso sin features.

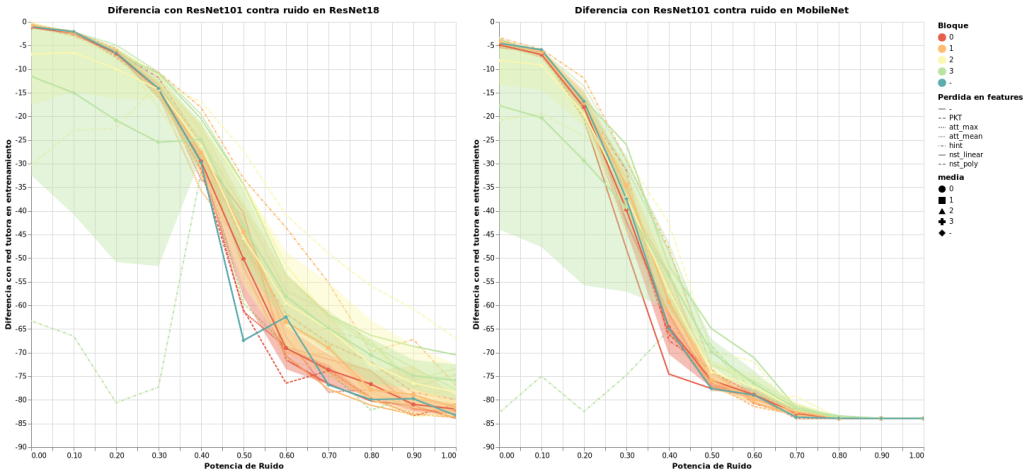

Figura 4.9: Decaimiento de destilación según cantidad de ruido, resumido por bloque de destilacion.

Comparando por bloque destilado, el que sufre mas el ruido con niveles menores de ruido es el de las ultimas capas, sin embargo, es el que obtiene valores mas altos en las zonas con mayor nivel de ruido para ResNet18. En las zonas intermedias, la destilación en el bloque 2 es la que mejor funciona cosa que se repite con menor diferencia en las zonas extremas.

Capítulo 5

Conclusiones y Trabajo Futuro

El objetivo de esta memoria era el de investigar la destilación en features, con énfasis en las técnicas que usan features. En detalle, investigar el estado del arte en torno a las técnicas existentes y realizar un estudio comparativo entre distintos tipos de destilación de modelo, en clasificación, usando distintas formas de perdidas sobre las features y la capa final. Los resultados al respectos son satisfactorios, se logró implementar un código lo suficientemente extensible para poder realizar mayores pruebas con distintas perdidas y modelos sin necesidad de entrar en grandes cambios y logró usarse para hacer pruebas contundentes que si bien pueden ser insuficientes para hablar de una exploración completa de la materia, permiten tener conclusiones interesantes.

En general se puede notar que la destilación es una técnica que funciona de buena manera, aún bajo ciertas condiciones de dificultad como puede ser la inclusión de ruido. Como todo, esta tiene sus limitaciones, las cuales parecen estar asociadas sobre todo al dataset sobre el que se trabaja y el modelo base que se usa. Interesantemente con esta técnica, la capacidad de generalización lejos de verse perjudicada, puede llegar incluso a ser mayor que la de la red tutora misma, al menos dentro de las condiciones en que esta fue probada.

Con respecto a la inclusión de features dentro de la destilación, sorprendentemente estas no aseguran inmediatamente una diferencia demasiado sustancial con respecto al accuracy logrado usando solo destilación de logits. El mejor caso logits encontrado da una diferencia de 0.85 entre la red estudiante y la red tutora, mientras que el mejor caso con features da una diferencia de 0.18, ambos usando ResNet. En el caso de las redes que obtuvieron un mayor incremento con respecto a su base, la diferencia es de 0.68 puntos en mapas de características , contra un -0.25 en logits, el primero en MobileNet y el segundo en ResNet

Es sin embargo al momento de uso de datos artificiales y en particular aquellos que generados con GAN que la inclusión de mapas de características logra dar sus frutos, lográndose una diferencia de mas de 70 puntos en la mayoría de los casos. Y logrando con respecto al caso entrenado sobre datos reales valores bastante similares. Así mismo la resistencia al ruido que le da al modelo, permite subir algunos puntos extra sobre el uso de destilación en logits sola, sin embargo, en algunos bloques y con algunas perdidas como hints puede llegar a ser perjudicial.

El desempeño de los distintos tipos de perdida en features parecen depender de cada caso, no se notó evidencia sobre si es mas conveniente usar uno u otro según red o capa. Con respecto al uso de cross entropy además de KD, no siempre permite mejores resultados. Y con respecto a las redes, si se pudo notar que ResNet18 logra mejores resultados que MobileNet, cosa que podría deberse o bien a una mayor similitud en la forma en que las redes extraen features o bien a que la base de ResNet18 es mas alta que la de MobileNet. Sobre diferencias entre tipos de destilacion usados y bloques, no fue posible notar muchas diferencias, a excepción de los casos de NST polinomial y hints en las ultimas capas, cosa probablemente relacionada con la especificidad de las mismas.

Profundizando un poco el análisis, fue posible llegar a dos resultados sumamente interesantes. El primero se relaciona con que es importante la vecindad sobre la que se trabaja, es decir, no es posible transferir el conocimiento de una red a otra sobre todo el espacio de entrada, pero, si se puede de manera bastante satisfactoria sobre el espacio de las imágenes similares a la base de datos de entrenamiento donde se entrenó la red tutor. El segundo, es que posiblemente la información que la transferencia de mapas de características da, compensa la información que se pierde al usar imágenes no reales. Ambas afirmaciones podrían tener relación con las transformaciones espaciales que hace sobre el espacio de entrada el uso de ReLU la composición de capas. Por un lado, la composición de funciones no reales crea una ”vecindad de buen funcionamiento.[en] la entrada que muy probablemente sea no convexa, de ahí el peor funcionamiento frente al ruido gaussiano sobre cierto nivel de potencia, y por otro el uso de transferencia de features permite facilitar la aproximación la función que parametriza la red tutora, al permitir aproximar sobre zonas posiblemente mayores en la entrada y al ”guiar”mas la extracción de features. Para poder afirmar la veracidad de esto ultimo es necesario entrar en una profundidad teórica mucho mayor de la que un estudio general sobre técnicas de destilación puede ofrecer.

Quedan bastantes tareas pendientes sobre las que convendría indagar. En primer lugar, se hacen necesarios estudios mas a fondo sobre como el dataset afecta al desempeño en este caso, usando mezclas de datasets sintéticos con reales y estudiando la cantidad de datos que se requieren del dataset real para poder destilar. Punto aparte es también el uso de una base de datos de mayor complejidad. Al respecto de esto, al cierre del documento quedaron pruebas hechas sobre imágenes del dataset Imagenet que no se alcanzaron a incorporar, pero que cuyos resultados quedarán disponibles en el repositorio online junto con la implementación[1].

[1] https://github.com/jpcosec/Memoria

Bibliografía

[1] Asami, Taichi, Ryo Masumura, Yoshikazu Yamaguchi, Hirokazu Masataki y Yushi Aono: *Domain adaptation of DNN acoustic models using knowledge distillation*. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, páginas 5185–5189, Marzo 2017, ISSN 15206149. http://ieeexplore.ieee.org/document/7953145/.

[2] Ba, Lei Jimmy y Rich Caruana: *Do deep nets really need to be deep?* Advances in Neural Information Processing Systems, 3(January):2654–2662, Diciembre 2014, ISSN 10495258. http://arxiv.org/abs/1312.6184.

[3] Bucilă, Cristian, Rich Caruana y Alexandra Niculescu-Mizil: *Model compression*. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006:535–541, 2006. https://www.cs.cornell.edu/%7B~%7Dcaruana/compression.kdd06.pdf.

[4] Bulò, Samuel Rota, Lorenzo Porzi y Peter Kontschieder: *Dropout distillation*. 33rd International Conference on Machine Learning, ICML 2016, 1:156–165, Junio 2016, ISSN 1938-7228. http://proceedings.mlr.press/v48/bulo16.html.

[5] Chen, Guobin, Wongun Choi, Xiang Yu, Tony Han y Manmohan Chandraker: *Learning efficient object detection models with knowledge distillation*. Advances in Neural Information Processing Systems, 2017-Decem:743–752, 2017, ISSN 10495258. http://papers.nips.cc/paper/6676-learning-efficient-object-detection-models-with-knowledge-distillation.

[6] Chen, Hanting, Yunhe Wang, Chang Xu, Chao Xu y Dacheng Tao: *Learning Student Networks via Feature Embedding*. IEEE Transactions on Neural Networks and Learning Systems, Diciembre 2020, ISSN 21622388. http://arxiv.org/abs/1812.06597.

[7] Chen, Hanting, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu y Qi Tian: *Data-free learning of student networks*. Proceedings of the IEEE International Conference on Computer Vision, 2019-Octob:3513–3521, 2019, ISSN 15505499. https://www.semanticscholar.org/paper/DAFL%3A-Data-Free-Learning-of-Student-Networks.-Chen-Wang/947928b4e7ff7ebb4a65d88d9c553a1fe5da7070.

[8] Chen, Wei Chun, Chia Che Chang y Che Rung Lee: *Knowledge Distillation with Feature Maps for Image Classification*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11363 LNCS:200–215, Diciembre 2019, ISSN 16113349. http://arxiv.org/abs/1812.00660.

[9] Chen, Yuntao, Naiyan Wang y Zhaoxiang Zhang: *DarkRank: Accelerating deep metric learning via cross sample similarities transfer*. 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, páginas 2852–2859, Julio 2018. http://arxiv.org/abs/1707.01220.

[10] Cheng, Yu, Duo Wang, Pan Zhou y Tao Zhang: *A Survey of Model Compression and Acceleration for Deep Neural Networks*. 2017. http://arxiv.org/abs/1710.09282.

[11] Cybenko, G.: *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems, 2(4):303–314, Diciembre 1989, ISSN 09324194. http://link.springer.com/10.1007/BF02551274.

[12] Doersch, Carl: *Tutorial on Variational Autoencoders*. Junio 2016. http://arxiv.org/abs/1606.05908.

[13] Dumoulin, Vincent y Francesco Visin: *A guide to convolution arithmetic for deep learning*. Marzo 2016. http://arxiv.org/abs/1603.07285.

[14] Farhadi, Mohammad y Yezhou Yang: *TKD: Temporal Knowledge Distillation for Active Perception*. undefined, 2019. http://arxiv.org/abs/1903.01522.

[15] Gal, Yarin y Zoubin Ghahramani: *Dropout as a Bayesian approximation: Representing model uncertainty in deep learning*. 33rd International Conference on Machine Learning, ICML 2016, 3:1651–1660, Junio 2016. http://arxiv.org/abs/1506.02142.

[16] Gao, Mengya, Yujun Shen, Quanquan Li, Junjie Yan, Liang Wan, Dahua Lin, Chen Change Loy y Xiaoou Tang: *An Embarrassingly Simple Approach for Knowledge Distillation*. Diciembre 2018. http://arxiv.org/abs/1812.01819.

[17] Ge, Yixiao, Zhuowan Li, Haiyu Zhao, Guojun Yin, Shuai Yi, Xiaogang Wang y Hongsheng Li: *FD-GAN: Pose-guided Feature Distilling GAN for Robust Person Re-identification*. Advances in Neural Information Processing Systems, 2018-Decem:1222–1233, Octubre 2018, ISSN 10495258. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gupta_Cross_Modal_Distillation_CVPR_2016_paper.pdf.

[18] Geyer, Charles J.: *Practical markov chain monte carlo*. Statistical Science, 7(4):473–483, 1992, ISSN 08834237. https://www.jstor.org/stable/2246094.

[19] Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville y Yoshua Bengio: *Generative adversarial nets*. Advances in Neural Information Processing Systems, 3(January):2672–2680, 2014, ISSN 10495258. http://www.github.com/goodfeli/adversarial.

[20] Gurau, Corina, Alex Bewley y Ingmar Posner: *Dropout Distillation for Efficiently Estimating Model Confidence*. 2018. http://arxiv.org/abs/1809.10562.

[21] Guzzi, Francesco, Luca De Bortoli, Stefano Marsi, Sergio Carrato y Giovanni Ramponi: *Distillation of a CNN for a high accuracy mobile face recognition system*. 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings, páginas 989–994, Mayo 2019. https://ieeexplore.ieee.org/document/8756937/.

[22] Hahnloser, Richard H.R. y H. Sebastian Seung: *Permitted and forbidden sets in symmetric threshold-linear networks*. Advances in Neural Information Processing Systems, 2001, ISSN 10495258. http://papers.nips.cc/paper/1793-permitted-and-forbidden-sets-in-symmetric-threshold-linear-networks.pdf.

[23] Hanin, Boris: *Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations*. Mathematics, 7(10):992, Agosto 2019, ISSN 2227-7390. http://arxiv.org/abs/1708.02691.

[24] He, Kaiming, Xiangyu Zhang, Shaoqing Ren y Jian Sun: *Deep residual learning for image recognition*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-Decem:770–778, Diciembre 2016, ISSN 10636919.

[25] Heo, Byeongho, Jeesoo Kim, Sangdoo Yun, Hyojin Park, Nojun Kwak y Jin Young Choi: *A comprehensive overhaul of feature distillation*. Proceedings of the IEEE International Conference on Computer Vision, 2019-Octob:1921–1930, Abril 2019, ISSN 15505499. http://arxiv.org/abs/1904.01866.

[26] Heo, Byeongho, Minsik Lee, Sangdoo Yun y Jin Young Choi: *Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons*. Proceedings of the AAAI Conference on Artificial Intelligence, 33:3779–3787, Noviembre 2019, ISSN 2159-5399. http://arxiv.org/abs/1811.03233.

[27] Hinton, Geoffrey, Oriol Vinyals y Jeff Dean: *Distilling the Knowledge in a Neural Network*. Advances in Neural Information Processing Systems, 2015. http://arxiv.org/abs/1503.02531.

[28] Hornik, Kurt: *Approximation capabilities of multilayer feedforward networks*. Neural Networks, 4(2):251–257, Enero 1991, ISSN 08936080. http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf.

[29] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto y Hartwig Adam: *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Abril 2017. http://arxiv.org/abs/1704.04861.

[30] Hu, Zhiting, Zichao Yang, Ruslan Salakhutdinov y Eric P. Xing: *Deep neural networks with massive learned knowledge*. EMNLP 2016 - Conference on Empirical Methods in Natural

Language Processing, Proceedings, páginas 1670–1679, 2016. https://www.aclweb.org/anthology/D16-1173.

[31] Huang, Zehao y Naiyan Wang: *Like What You Like: Knowledge Distill via Neuron Selectivity Transfer*. Julio 2017. http://arxiv.org/abs/1707.01219.

[32] Kawaguchi, Kenji, Leslie Pack Kaelbling y Yoshua Bengio: *Generalization in Deep Learning*. Octubre 2017. http://arxiv.org/abs/1710.05468.

[33] Kim, Jangho, Seong Uk Park y Nojun Kwak: *Paraphrasing complex network: Network compression via factor transfer*. Advances in Neural Information Processing Systems, 2018-Decem:2760–2769, 2018, ISSN 10495258. https://arxiv.org/pdf/1802.04977.pdf.

[34] Kimura, Akisato, Zoubin Ghahramani, Koh Takeuchi, Tomoharu Iwata y Naonori Ueda: *Few-shot learning of neural networks from scratch by pseudo example optimization*. British Machine Vision Conference 2018, BMVC 2018, 2019. https://www.semanticscholar.org/paper/Few-shot-learning-of-neural-networks-from-scratch-Kimura-Ghahramani/ebc50c378d25e780d5adb7e54a1c8c153761097b.

[35] Kingma, Diederik P. y Max Welling: *Auto-encoding variational bayes*. 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings, Diciembre 2014. http://arxiv.org/abs/1312.6114.

[36] Krizhevsky, Alex: *One weird trick for parallelizing convolutional neural networks*. Abril 2014. http://arxiv.org/abs/1404.5997.

[37] Li, Hao Ting, Shih Chieh Lin, Cheng Yeh Chen y Chen Kuo Chiang: *Layer-level knowledge distillation for deep neural network learning*. Applied Sciences (Switzerland), 9(10):1966, Mayo 2019, ISSN 20763417. https://www.mdpi.com/2076-3417/9/10/1966.

[38] Li, Quanquan, Shengying Jin y Junjie Yan: *Mimicking very efficient network for object detection*. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-Janua:7341–7349, Julio 2017. http://ieeexplore.ieee.org/document/8100259/.

[39] Li, Quanquan, Shengying Jin y Junjie Yan: *Mimicking very efficient network for object detection*. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-Janua:7341–7349, Julio 2017. http://ieeexplore.ieee.org/document/8100259/.

[40] Li, Tianhong, Jianguo Li, Zhuang Liu y Changshui Zhang: *Few Sample Knowledge Distillation for Efficient Network Compression*. undefined, páginas 1–14, 2018. http://arxiv.org/abs/1812.01839.

[41] Liu, Miao, Xin Chen, Yun Zhang, Yin Li y James M. Rehg: *Paying More Attention to Motion: Attention Distillation for Learning Video Representations*. Abril 2019. http://arxiv.org/abs/1904.03249.

[42] Liu, Peiye, Wu Liu, Huadong Ma, Tao Mei y Mingoo Seok: *KTAN: Knowledge Transfer Adversarial Network*. Octubre 2018. http://arxiv.org/abs/1810.08126.

[43] Liu, Yifan, Changyong Shun, Jingdong Wang y Chunhua Shen: *Structured Knowledge Distillation for Dense Prediction*. Marzo 2019. http://arxiv.org/abs/1903.04197.

[44] Lopes, Raphael Gontijo, Stefano Fenu y Thad Starner: *Data-Free Knowledge Distillation for Deep Neural Networks*. 2017. http://arxiv.org/abs/1710.07535.

[45] Lu, Liang, Michelle Guo y Steve Renals: *Knowledge distillation for small-footprint highway networks*. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, páginas 4820–4824, Agosto 2017, ISSN 15206149. http://arxiv.org/abs/1608.00892.

[46] Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu y Liwei Wang: *The expressive power of neural networks: A view from the width*. Advances in Neural Information Processing Systems, 2017-Decem:6232–6240, 2017, ISSN 10495258. http://arxiv.org/abs/1709.02540.

[47] Luo, Zelun, Jun Ting Hsieh, Lu Jiang, Juan Carlos Niebles y Li Fei-Fei: *Graph distillation for action detection with privileged modalities*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11218 LNCS:174–192, Noviembre 2018, ISSN 16113349. http://arxiv.org/abs/1712.00108.

[48] Mehta, Rakesh y Cemalettin Ozturk: *Object detection at 200 frames per second*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11133 LNCS:659–675, 2019, ISSN 16113349. https://arxiv.org/pdf/1805.06361.pdf.

[49] Micaelli, Paul y Amos Storkey: *Zero-shot Knowledge Transfer via Adversarial Belief Matching*. undefined, 2019. http://arxiv.org/abs/1905.09768.

[50] Mikhael, John G. y Rafal Bogacz: *Learning Reward Uncertainty in the Basal Ganglia*. PLoS Computational Biology, 12(9):1–11, 2016, ISSN 15537358. http://arxiv.org/abs/1608.04363%0Ahttp://dx.doi.org/10.1109/LSP.2017.2657381%0Ahttp://www.orangetide.com/smallc/articles/chaps/chap1/chap1.htm%0Ahttp://131.111.33.187/groups/cnbh/research/publications/pdfs/SVOSAnnexB1988.pdf%0Ahttp://www.mdpi.com/2079-.

[51] Mun, Jonghwan, Kimin Lee, Jinwoo Shin y Bohyung Han: *Learning to specialize with knowledge distillation for visual question answering*. Advances in Neural Information Processing Systems, 2018-Decem:8081–8091, 2018, ISSN 10495258. https://papers.nips.cc/paper/8031-learning-to-specialize-with-knowledge-distillation-for-visual-question-answer

[52] Nayak, Gaurav Kumar, Konda Reddy Mopuri, Vaisakh Shaj, R. Venkatesh Babu y Anirban Chakraborty: *Zero-shot*

knowledge distillation in deep networks. 36th International Conference on Machine Learning, ICML 2019, 2019-June:8317–8325, 2019. https://www.semanticscholar.org/paper/Zero-Shot-Knowledge-Distillation-in-Deep-Networks-Nayak-Mopuri/eced39bd4b6aa1097a7615dd0c502e4a7510796a.

[53] Odena, Augustus, Christopher Olah y Jonathon Shlens: *Conditional Image Synthesis With Auxiliary Classifier GANs*. Octubre 2016. http://arxiv.org/abs/1610.09585.

[54] Pan, Xingyuan y Vivek Srikumar: *Expressiveness of rectifier networks*. 33rd International Conference on Machine Learning, ICML 2016, 5:3595–3606, Noviembre 2016. http://arxiv.org/abs/1511.05678.

[55] Pascanu, Razvan, Guido Montúfar y Yoshua Bengio: *On the number of response regions of deep feedforward networks with piecewise linear activations*. 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings, Diciembre 2014. http://arxiv.org/abs/1312.6098.

[56] Passalis, Nikolaos y Anastasios Tefas: *Learning Deep Representations with Probabilistic Knowledge Transfer*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11215 LNCS:283–299, 2018, ISSN 16113349. http://openaccess.thecvf.com/content_ECCV_2018/html/Nikolaos_Passalis_Learning_Deep_Representations_ECCV_2018_paper.html.

[57] Qin, Zheng, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng y Jian Sun: *ThunderNet: Towards real-time generic object detection on mobile devices*. Proceedings of the IEEE International Conference on Computer Vision, 2019-Octob:6717–6726, 2019, ISSN 15505499. http://arxiv.org/abs/1903.11752.

[58] Redmon, Joseph, Santosh Divvala, Ross Girshick y Ali Farhadi: *You only look once: Unified, real-time object detection*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-Decem:779–788, Junio 2016, ISSN 10636919. http://arxiv.org/abs/1506.02640.

[59] Romero, Adriana, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta y Yoshua Bengio: *FitNets: Hints for thin deep nets*. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, Diciembre 2015. http://arxiv.org/abs/1412.6550.

[60] Srinivas, Suraj y François Fleuret: *Supplementary material for "knowledge Transfer with Jacobian Matching*

. 35th International Conference on Machine Learning, ICML 2018, 11:7524–7525, Julio 2018, ISSN 1938-7228. http://proceedings.mlr.press/v80/srinivas18a.html.

[61] Tan, Sarah, Rich Caruana, Giles Hooker, Paul Koch y Albert Gordo: *Learning Global Additive Explanations for Neural Nets Using Model Distillation*. Enero 2018. http://arxiv.org/abs/1801.08640.

[62] Tavakolian, Mohammad, Mohammad Sabokrou y Abdenour Hadid: *AVD: Adversarial Video Distillation*. Julio 2019. http://arxiv.org/abs/1907.05640.

[63] Telgarsky, Matus: *Benefits of depth in neural networks*. Journal of Machine Learning Research, 49(June):1517–1539, Febrero 2016, ISSN 15337928. http://arxiv.org/abs/1602.04485.

[64] Tran, Vinh, Yang Wang y Minh Hoai: *Back to the Future: Knowledge Distillation for Human Action Anticipation*. Abril 2019. http://arxiv.org/abs/1904.04868.

[65] Wang, Kuan Chieh, Paul Vicol, James Lucas, Li Gu, Roger Grosse y Richard Zemel: *Adversarial distillation of Bayesian neural network posteriors*. 35th International Conference on Machine Learning, ICML 2018, 12:8239–8248, 2018. https://arxiv.org/pdf/1806.10317.pdf.

[66] Wang, Tao, Li Yuan, Xiaopeng Zhang y Jiashi Feng: *Distilling object detectors with fine-grained feature imitation*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019-June:4928–4937, Junio 2019, ISSN 10636919. http://arxiv.org/abs/1906.03609.

[67] Wang, Tongzhou, Jun Yan Zhu, Antonio Torralba y Alexei A. Efros: *Dataset Distillation*. undefined, 2018. http://arxiv.org/abs/1811.10959.

[68] Wang, Wanwei, Wei Hong, Feng Wang y Jinke Yu: *GAN-Knowledge Distillation for One-stage Object Detection*. IEEE Access, páginas 1–1, Junio 2020, ISSN 21693536. http://arxiv.org/abs/1906.08467.

[69] Wang, Xiaojie, Yu Sun, Rui Zhang y Jianzhong Qi: *KDGAN: Knowledge distillation with generative adversarial networks*. Advances in Neural Information Processing Systems, 2018-Decem:775–786, 2018, ISSN 10495258. http://papers.nips.cc/paper/7358-kdgan-knowledge-distillation-with-generative-adversarial-networks.

[70] Xu, Zheng, Yen Chang Hsu y Jiawei Huang: *Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks*. 6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings, Septiembre 2018. http://arxiv.org/abs/1709.00513.

[71] Yim, Junho, Donggyu Joo, Jihoon Bae y Junmo Kim: *A gift from knowledge distillation: Fast optimization, network minimization and transfer learning*. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-Janua:7130–7138, 2017. http://openaccess.thecvf.com/content_cvpr_2017/html/Yim_A_Gift_From_CVPR_2017_paper.html.

[72] Yoshioka, Kentaro, Edward Lee, Simon Wong y Mark Horowitz: *Dataset Culling: Towards Efficient Training of Distillation-Based Domain Specific Models*. Proceedings - International Conference on Image Processing, ICIP, 2019-Septe:3237–3241, 2019, ISSN 15224880. https://www.semanticscholar.org/paper/Dataset-Culling%3A-Towards-Efficient-Training-Of-Yoshioka-Lee/dfd55d4c8fd41ad549f93b555a4d7a4c3b88f99d.

[73] Yosinski, Jason, Jeff Clune, Yoshua Bengio y Hod Lipson: *How transferable are features in deep neural networks?* Advances in Neural Information Processing Systems, 4(January):3320–3328, Noviembre 2014, ISSN 10495258. http://arxiv.org/abs/1411.1792.

[74] Zagoruyko, Sergey y Nikos Komodakis: *Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer*. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, Diciembre 2019. http://arxiv.org/abs/1612.03928.

[75] Zhang, Chiyuan, Benjamin Recht, Samy Bengio, Moritz Hardt y Oriol Vinyals: *Understanding deep learning requires rethinking generalization*. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, Noviembre 2019. http://arxiv.org/abs/1611.03530.

[76] Zhou, Zhi Hua y Ji Feng: *Deep forest*. National Science Review, 6(1):74–86, Febrero 2019, ISSN 2053714X. http://arxiv.org/abs/1702.08835.

Capítulo 6

Anexos

Al momento de escribir esta memoria, se priorizó escribir un documento sintético antes que caer en la reproducción cruda de fuentes. Esto con el objetivo de hacer un documento mas legible y coherente, al costo de perder detalles importantes, pero no fundamentales para una correcta comprensión del trabajo realizado. Si bien estos están en su mayoría referenciados dentro del cuerpo principal de la memoria, se aprovechan las páginas siguientes para exponer algunos de estos.

6.1. Notas sobre investigaciones en destilación usando mapas de características

Se exponen a continuación algunas notas sobre destilación usando mapas de características . Estas se encuentran clasificadas según tarea en dos grandes grupos, clasificación y otros.

6.1.1. Usando mapas de características en tarea de clasificación.

Se exponen a continuación algunas notas sobre destilación usando mapas de características en tarea de clasificación. Estas se encuentran separadas en tres secciones según el tipo de transferencia usada.

Transferencia usando red neuronal

FITNETS: HINTS FOR THIN DEEP NETS [59] Año: 2015

Autores: Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta y Yoshua Bengio

En: ICLR

La primera investigación en destilación en usar mapas de características es *Fitnets* del 2015. Aquí, los mapas de características son usados a modo de indicio o *hints* de lo que una red estudiante aun no entrenada, debiese aprender de una red entrenada. El estudio se centra en el uso de redes estudiantes mas profundas que la red tutora, pero de menor ancho y por lo mismo menor cantidad de parámetros. Todo esto con el objetivo de mejorar la regularización, para evitar el *overfitting* propio de las redes de mayor profundidad, en un tiempo en que el aprendizaje profundo estaba bastante centrado en lograr redes cada vez mas profundas.

Definen un *hint* como la salida de una capa convolucional F_T, desde la cual la capa de la red estudiante debe aprender. Recomiendan usar las salidas de la parte media de la red, sin especificar mucho criterios ni a que nivel de la parte media se debe extraer el mapa de característica . Usando esto se define una perdida de *hints* con la siguiente forma.

$\mathscr{L}_{HT} = \frac{1}{2} \mid\mid F_T - r(F_s) \mid\mid^2$

En la que r es un regresor consistente de una capa convolucional, con la misma funcion de activacion de F_T usado para poder ajustar el tamaño de F_S al de F_T. Este se entrena separadamente, antes de entrenar la red estudiante. La perdida se suma a *cross-entropy* siendo ponderado por un λ para ajustar los niveles de cada perdida.

$\mathscr{L}_{NST}(W_S) = \mathscr{L}_{ce}(Y_{true}, ps) + \lambda \mathscr{L}_{HT}(F_T, F_S)$

Layer-Level Knowledge Distillation for Deep Neural Network Learning [37] Año: 2019

Autores: Hao-Ting Li, Shih-Chieh Lin, Cheng-Yeh Chen y Chen-Kuo Chiang

En: MDPI

El modelo consiste de dos partes, LSP y ALS. LSP es un procedimiento para determinar las mejores capas a ser conectadas. Para esto se usa el argumento mínimo del grammiamo entre clases sumado al grammiano entre capas.

ALS consiste en usar capas para proyectar las features de un modelo al otro. Mediante una capa de proyección por modelo, se mapean los features a un vector de una dimensionalidad común en $\mathbb{R}^n$. A esto se suma otra una capa densa de alineamiento mediante la cual se toma distancia y se aplica la perdida, en una manera similar a la usada en [59].

$\mathscr{L}^{(K)}_{Align}(t,s) = \|X^{(i)}_t - X^{(j)}_s\|_2$

La perdida final incorpora la suma de todas las perdidas de todas las etapas de ALS .

$\mathscr{L}_{total} = \sum_{k=1}^{n} \mathscr{L}^{(k)}_{Align} + \mathscr{L}^{(p)}_{model}$

Paraphrasing Complex Network: Network Compression via Factor Transfer [33] Año: 2018

Autores: Jangho Kim, SeongUk Park, Nojun Kwak

En: NIPS

Destila a nivel de mapas de características , pero proponiendo en el acoplamiento un sistema un poco mas complejo. Sobre una capa de la red tutora se usa un *autoencoder* para crear factores, una especie de versión condensada de el mapa de característica real. En la red estudiante se usa un regresor como el de [59] para realizar el acople dimensional.

El traspaso de información se realiza en 2 fases. Primero se entrena el *autoencoder* en la red tutora reconstruyendo el input desde en la capa anterior. En esta etapa se minimiza la siguiente perdida , $P(x)$.

$\mathscr{L}_{rec} = \|x - P(x)\|^2$

Luego, se entrena la red estudiante usando a modo de perdida alguna norma p entre la salida normalizada en L_2 del *autoencoder* y el regresor, a lo cual se suma *cross entropy* en los *logits*. Esta técnica tiene la ventaja de que transferir entre distintos bloques ajustando el *autoencoder*, al costo de ser mas engorroso para entrenarse.

Transferencia usando reducción matemática

Learning Deep Representations with Probabilistic Knowledge Transfer [56] Año: 2018

Autores: Nikolaos Passalis, Anastasios Tefas

En: ECCV

En esta investigación, el problema del acoplamiento dimensional se trata obteniendo una distribución de los mapas de características de las capas que se quieren comparar. Esto permite realizar la destilación usando cualquier tipo de característica. A modo de ejemplo, en el paper mencionan los siguientes, probándose el caso regular y el uso de mapas de características manuales.

- Transferencia desde distintos dominios, un ejemplo extremo de esto es realizar transferencia desde un modelo con información de texto a uno con imágenes.
- Features creados a mano como SIFT, ORB, etc.
- Transferir mapas de características usando tareas distintas a la original usada.

Al ser difícil de obtener de manera directa la distribución completa de las features de la red se minimiza una distribución condicional. Esta se estima usando *kernel density estimation*. Tomando un kernel K, la distribución condicional para la red tutora es:

$p^t_{i|j} = \frac{K(F^i_t F^j_t; 2\sigma^2_t)}{\sum^N_{k=1,k\neq j} K(F^i_t F^j_t; 2\sigma^2_t)} \in [0,1]$

La distribución para la red estudiante usa la misma formula, pero usando los features F^i_s y la varianza σ^2_s. Como kernels se proponen el uso del kernel gaussiano y la distancia coseno, la cual fue la usada en esta memoria. $K = \frac{1}{2}\frac{a^\top b}{\|a\|_2\|b\|_2} + 1 \in [0]$.

Ambas distribuciones se aproximan minimizando la divergencia Kullback-Leibler sobre cada batch, $D_{\mathrm{KL}}(P\|Q) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} p(x) \ln \frac{p(x)}{q(x)}$.

A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning [71] Año: 2017

Autores: Junho Yim, Donggyu Joo, Jihoon Bae, Junmo Kim

En: CVPR

En vez de destilar directamente las features de la red tutora, centran su problema en la destilación del flujo del procedimiento de resolución FSP *Flow of Solution Procedure*)de la red tutora, definida como la relación que van tomando estos features. Matemáticamente hablando, el FSP entre dos capas se define como la matriz Grammiana entre los features de ambas. En el caso de dos capas 1 y 2, sus pesos W y un input x, este valor sería:

$$G_{i,j}(x,W) = \sum_{s=1}^{h} \sum_{t=1}^{w} \frac{F^1_{s,t,i}(x,W) F^2_{s,t,i}(x,W)}{h \times w}$$

Nótese que la formula se define para capas donde no hay reducción dimensional en w ni h, como es el caso de las capas dentro de los bloques ResNet. De tal forma la perdida se define como la distancia L_2 entre las FSPs de ambas redes, donde se deja libre un parámetro λ_i sobre las n posiciones de las matrices para poder ponderar entre redes. En el caso expuesto este lambda se deja libre.

$$\mathscr{L}_{FSP} = \frac{1}{N} \sum_x \sum_{i=1}^{n} \lambda_i \left\| G^T_i(x;W_T) - G^S_i(x;W_S) \right\|_2$$

Al momento del entrenamiento, el entrenamiento se divide en dos fases, inicialización y entrenamiento sobre la tarea principal. La inicialización se realiza seteando W_S de tal forma que minimice $\mathscr{L}_{FSP}$ entre ambas redes, para seguidamente realizar un *fine tunning* entrenando de manera regular contra las etiquetas originales.

Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer [74] Año: 2016

Autores: Sergey Zagoruyko, Nikos Komodakis

En: ICLR

Desde la percepción humana, se distinguen en 2 tipos de procesos de percepción; no atencionales y atencionales. Los primeros permiten observar generalidades de una escena mientras que los segundos permiten recolectar información de alto nivel sobre detalles. Sumando ambos se logra navegar en ciertos detalles de una escena recolectando lo importante sin necesidad de gastar energía en escenarios complejos.

En el contexto de CNNs, se considera la atención como mapas espaciales que permiten codificar donde enfocar mas el procesamiento. Estos mapas se pueden definir con respecto a varias capas de la red y según en que se basen se dividen en 2 tipos, de activación y de gradiente. En esta

investigación se realiza un estudio sobre como los mapas de atención varían según arquitecturas y como estos mapas pueden ser transferidos a redes estudiantes desde una red tutora ya entrenada. Dentro del paper mencionan que en general las redes de mayor precisión suelen tener la atención mas marcada. Además, comparando entre capas, las iniciales se "fijan.[en] detalles como ojos o narices mientras que las mas profundas se fijan en objetos de mayor nivel como caras.

Se centraron en arquitecturas *fully convolutional*, es decir, redes donde la clasificación o regresión final se realiza sin el uso de capas densas, si no aprovechando la reducción dimensional que dan las convoluciones.

- **Mapa de atencion basado en activaciones**

  Considerando una capa de una red y su activación F_T, se define una función sobre la activación $\mathscr{F} : \mathscr{R}^{C\times H\times W} \rightarrow \mathscr{R}^{H\times W}$. Esta pondera la importancia de la activación a nivel de todos los canales en una posición h, w . Algunas de las funciones propuestas son.

  1. Suma de los absolutos entre los C canales: $\mathscr{F}_{sum}(A) = \sum_{i=1}^{C} \mid A_i \mid$
  2. Suma de potencias: $\mathscr{F}^{p}_{sum}(A) = \sum_{i=1}^{C} \mid A_i \mid^p$
  3. Máximo de potencias: $\mathscr{F}^{p}_{max}(A) = \text{máx}_{i=1,c} \mid A_i \mid^p$

  La perdida de entrenamiento en este caso es la siguiente, donde $\frac{Q_i^j}{\|Q_i^j\|_2}$ es simplemente una normalización de las activaciones:

  $\mathscr{L}_{NST}(WS) = \mathscr{L}_{ce}(Y_{true}, ps) + \frac{\lambda}{2}\mathscr{L}_{at}(F_T, F_S)$

  Donde $\mathscr{L}_{at}(F_T, F_S) = \sum_{j\in\mathscr{C}} \left\| \frac{Q_S^j}{\|Q_S^j\|_2} + \frac{Q_T^j}{\|Q_T^j\|_2} \right\|_p$

- **Mapa de atención basado en gradiente**

Para el caso de gradiente, se asume que el gradiente de la perdida de clasificación con respecto a una entrada permite medir la "sensibilidad"de la red ante el estimulo, para esto se define el gradiente como.

$J_i = \frac{\partial \mathscr{L}_{ce}(W_i, x)}{\partial x}$

La perdida toma la siguiente forma, la cual puede ser difícil para analizarse analíticamente ya que implica realizar backpropagation dos veces. Sin con las técnicas modernas de diferenciación automática, como las que trae cualquier librería de aprendizaje profundo moderna, el computo de esto es trivial. La perdida en este caso es la siguiente:

$\mathscr{L}_{NST}(WS) = \mathscr{L}_{ce}(Y_{true}, ps) + \frac{\lambda}{2}\|J_s - J_T\|_2$

Like What You Like: Knowledge Distill via Neuron Selectivity Transfer [31] Año: 2017

Autores: Zehao Huang, Naiyan Wang

En: ICLR

Definen el conocimiento selectivo de las neuronas, como la activación de cada neurona para un patrón particular encontrado en la entrada X bajo un una tarea particular. Desde esto el método propuesto lo bautizan transferencia de selectividad neuronal o NST. En NST interpretan el mapa de activación de cada posición en ancho y alto del tensor mapa de característica como si fuera un muestreo de como la red neuronal interpreta la imagen de entrada. De esta manera y en una forma similar a los mapas de atención, pueden ver en que se centra la red neuronal para realizar la detección. De esta forma buscan realizar una especie de alineamiento de las distribuciones de la red tutora y estudiante.

Para la transferencia de los mapas de características usan MMD, discrepancia máxima de media. El MMD se calcula de la siguiente forma.

$$\mathscr{L}_{MDD^2}(S_p, S_q) = || \frac{1}{N}\sum_{i=1}^{N}\phi(p^i) - \frac{1}{M}\sum_{j=1}^{M}\phi(q^j) ||$$

Donde $\phi(.)$ es una función explicita de mapeo y S_p, S_q son dos colecciones de muestras $S_p = \{p^i\}_{i=1}^{N}$ $S_q = \{q^i\}_{i=1}^{m}$. Usando el kernel trick, esto se puede reformular como:

$$\mathscr{L}_{MDD^2}(S_p, S_q) = || \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N}k(p^i, p^{i'}) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M}k(q^j, q^{j'}) - \frac{1}{MN}\sum_{i=1}^{M}\sum_{j'=1}^{M}k(p^i, q^j)$$

Lo cual finalmente se aplica usando muestreos desde F_T y F_S, muestrando la activación a través de todos los canales y normalizando $p^i = \frac{f_T^i}{||f_T^i||_2}$. Al igual que en la mayoría de estos casos, se entrena añadiendo *cross entropy* sobre la capa final.

$$\mathscr{L}_{NST}(WS) = \mathscr{L}_{ce}(Y_{true}, ps) + \frac{\lambda}{2}\mathscr{L}_{MDD^2}(F_T, F_S)$$

Sobre el k usado, se usaron kernel lineal, polinomial de $d = 2$ y $c = 0$ y gaussiano con σ^2 igual al error cuadrático medio entre los pares. El caso lineal es idéntico al caso de [74] que usa mapas de atención de media y el caso polinomial de orden 2 es igual a la matriz de gramm usada en [71].

Transferencia de activacion o borde

Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons
Año: 2018

Autores: Byeongho Heo, Minsik Lee, Sangdoo Yun, Jin Young Choi

En: AAAI

Partiendo desde el hecho que la combinación de las activaciones en una capa forman una función binaria de ”bordes”sobre el espacio de salida de esta, el paper propone la transferencia no de la magnitud de la respuesta de una neurona si no de las activaciones de esta. Comparan por ejemplo las activaciones logradas con MSE como es el caso de [59] con el método propuesto, logrando un mejor borde de separación de clases, al costo de mover la magnitud de las activaciones.

La perdida es similar a la de [59], reemplazando la función de activaciones de la capa σ por

una binarización de la misma.

$$\rho(x) = \begin{cases} 1 & \text{si x ¿0} \\ 0 & \text{si no} \end{cases}$$

Dado que ρ no es diferencial se usa en el entrenamiento una perdida similar a Hinge Loss de SVM, donde μ representa un margen para estabilidad en el entrenamiento, $\vec{1}$ es un vector de largo M y valor 1 en cada elemento y $\odot$ es el producto punto entre los elementos. Notese que en este caso tanto F_S como F_T representan la salida de una capa previo a la función de activación.

$$\mathscr{L}(I) = \|\rho(F_T) \odot \sigma\left(\mu\vec{1} - F_S\right) - \left(\vec{1} - \rho(F_T)\right) \odot \sigma\left(\mu\vec{1} - -F_S\right)\|_2^2$$

Con respecto a la neurona iésima en la red tutora t_i y estudiante s_i, esto tiene una derivada por partes igual a.

$$-\frac{\partial \mathscr{L}}{\partial s_i} = \begin{cases} 2(s_i - \mu) & \text{si } \rho(t_i) = 1 \text{ y } s_i < \mu \\ -2(s_i + \mu) & \text{si } \rho(t_i) = 0 \text{ y } s_i > -\mu \\ 0 & \text{en cualquier otro caso} \end{cases}$$

En casos en que hayan tamaños distintos en la cantidad de neuronas se usa un regresor en la misma manera que en Fitnets.

A Comprehensive Overhaul of Feature Distillation [25] Año: 2019

Autores: Byeongho Heo, Jeesoo Kim, Sangdoo Yun, Hyojin Park, Nojun Kwak, Jin Yor Choi

En: ICCV

La investigación parte definiendo la destilación de mapas de características de manera generalizada como un subconjunto de transfer learning en el cual se usan los mapas de características de la red tutora para el entrenamiento de la red estudiante. Definiendo una medida de distancia d y una función de adaptación dimensional para cada red T_T, T_S, las cuales son aplicadas sobre las features de cada red F_T y F_S, la perdida de cualquier destilación de features puede ser definida de la siguiente forma:

$$\mathscr{L}_{FD} = d\left[T_T\left(F_T\right), T_S\left(F_S\right)\right]$$

Además, definen toda perdida en los términos expuestos, es decir, las transformaciones T_T y T_S y la distancia d. Un detalle interesante en el que reparan es que todas estas técnicas tienen como costo algún nivel de perdida de información al momento de realizar la transferencia, como por ejemplo las dimensiones de canal en la transferencia de mapas de atención [74] o el valor de los features en la transferencia de bordes de activación [26].

En función de esto definen un metodo nuevo de destilacion basado en el trabajo anterior del mismo autor, [26]. En vez de esto, definen una activación llamada Margin Relu.

$$\sigma_{\text{MReLU}} = max(x, m)$$

Donde m es un margen de valor negativo y x la activación de la red. En el estudio el margen se deja en la esperanza las respuestas negativas sobre todas las imágenes, el cual puede ser calculado

en lina por cada batch durante en el entrenamiento o por época usando los parámetros del ultimo *batch normalization*. La activación se usa en vez de la ReLU normal la capa a ser transferida de la red tutora, mientras que en la red estudiante usa un regresor de 1x1 y *batch normalization*.

Además, como función de distancia se propone una distancia L2 parcial, que transfiere el conocimiento de la red tutora para una posición i-esima en el tensor de activación en caso que la respuesta del estudiante sea menor que la del tutor y menor que 0.

$$d_p(T,S) = \sum_i^{WHC} \begin{cases} 0 & \text{si } F_{S_i} \leq F_{Ti} \leq 0 \\ (F_{Ti} - F_{S_i}) & \text{en cualquier otro caso} \end{cases}$$

Reducción dentro de batch

Los casos en esta seccion siguen formas distintas al esquema $d(T_T(F_T), T_S(F_S))$ que siguen los casos presentados anteriormente.

Learning Student Networks via Feature Embedding [6] Año: 2018

Autores: Hanting Chen, Yunhe Wang, Chang Xu, Chao Xu and Dacheng Tao

En: IEEE

Los autores realizan transferencia del conocimiento de los features de una manera un poco distinta a la vista hasta el momento ya que lejos de intentar acercar la respuesta de la red estudiante a la de la red tutora según alguna forma, buscan replicar la distancia relativa entre las respuestas de cada red a dos imagen es x_j,x_i del *batch* arbitrarios. De esta forma introducen una perdida llama *local preserving loss*, la cual toma la siguiente forma:

$$\mathscr{L}_{\text{LP}} = \frac{1}{m} \sum_{i,j} \alpha_{i,j} \| f_s^i - f_s^j \|_2^2$$

Donde m es el tamaño del *minibatch*, y $\alpha_{i,j}$ es un coeficiente que toma un valor positivo en caso que las features de la muestra j en la red tutora, f_T^j sea el vecino mas cercano de los mapas de características de la muestra i, f_T^i y toma valor 0 en cualquier otro caso, es decir:

$$\alpha_{i,j} = \begin{cases} exp(\| f_T^i - f_T^j \|_2^2) & \text{si } j \in N(i) \\ 0 & \text{en otro caso} \end{cases}$$

6.1.2. Destilación usando mapas de características en tareas mas complejas

Segmentación Semántica

Structured Knowledge Distillation for Semantic Segmentation [43] Año: 2019

Autores: Yifan Liu, Ke Chen, Chris Liu, Zengchang Qin, Zhenbo Luo, Jingdong Wang

En: CVPR

El problema de segmentación semántica consiste en la predicción de un mapa Q score de clases para cada uno de los píxeles de una imagen, cosa que hace inaplicable el uso de cualquier técnica usada en clasificación o detección. Sin embargo, es bastante usual en el campo el comenzar un modelo reutilizando la porción de la red que extrae mapas de características de alguna red, previamente entrenada en otra tarea. El paper usa varias perdidas distintas, una de destilación en feature map y en la predicción, una de segmentación y otra discriminativa para destilar un modelo de segmentación semántica.

Perdida de ground truth, se usa simplemente *cross entropy* entre la salida de la red y el *ground truth* para cada uno de los pixeles.

Destilacion a nivel de pixel en cada pixel se usa una destilación idéntica a KD de [27] es decir, la divergencia KL entre los logits de la red tutora y la red estudiante. $L_{\text{pi}} = \frac{1}{N_w N_H} \sum_{i,j \in W,H} KL\left(q^S_{i,j} \| q^T_{i,j}\right)$.

Pairwise distillation, se usa una medida de similaridad entre dos píxeles calculada como $a_{ij} = f_i^\top f_j / \left(\|f_i\|_2 \left\|f_j\right\|_2\right)$, la cual luego es comparada para cada uno de los píxeles de los mapas de caracteristicas de ambas redes. $L_{\text{pa}} = \frac{1}{(N_w N_H)^2} \sum_{i \in W,H} \sum_{j \in W,H} \left(a^S_{ij} - a^T_{ij}\right)^2$

Hollistic (Adversarial) distillation, consiste en el uso de un discriminador D que actúa a modo de red de *embeding* usando a la red de segmentación como si fuese un generador. Luego, usando la distancia wasserstein se compara la distribución de ambas salidas. $L_{\text{ho}} = \mathbb{E}[D(Q^S)] - \mathbb{E}[D(Q^T)]$

Detección

Distilling Object Detectors with Fine-grained Feature Imitation [66] Año: 2019

Autores: Tao Wang, Li Yuan, Xiaopeng Zhang, Jiashi Feng

En: CVPR

El estudio se centra en destilación en el contexto de detección de objetos. Ya que a diferencia de la clasificación, en la cual se obtiene una sola etiqueta de clase para toda la imagen, en detección el objetivo es diferenciar los objetos del fondo y clasificar cada uno por separado (paralela o secuencialmente). De partida esto introduce dos diferencias con respecto a la clasificación, las cuales tienen consecuencias importantes al momento de destilar;

1. Los fondos en las imágenes pueden ser mucho mas variados que los objetos de primer plano, de esta manera en detección gran parte de la información de la imagen es o información inútil o ruido que puede condicionar al detector a obtener respuestas falsas.

2. La destilación propuesta por Hinton en [27] debe ser modificada ya que la etiqueta deja de ser solo un valor de clase, si no de *bounding box* y clase por cada objeto presente en la

imagen.

El método de destilación propuesto imita los mapas de características de una manera no muy distinta a como se imitan los mapas de características en [59], pero lo hace solamente en una región particular. Es decir, para una posición i,j en el feature map se define la perdida local l como.

$l_{ij} = \sum_{c=1}^{C} \left(r(F_s)_{ijc} - F_{T_{ijc}} \right)^2$

Así la perdida completa del metodo es la siguiente.

$L_{\text{FGFI}} = \frac{1}{2N_p} \sum_{i,j \in W,H} \mathbb{I}_{ij} * l_{ij}$

Donde $\mathbb{I}_{i,j}$ es un valor binario que es 1 cuando la posición pertenece a la mascara de imitación y 1 cuando no pertenece y $N_p = \sum_{i,j \in W,H} \mathbb{I}_{ij}$ es el área de la mascara. La mascara de imitación se calcula combinado mediante *OR* todos los *anchors* que tienen un IOU mayor que algún umbral con el *bounding box*.

Mimicking Very Efficient Network for Object Detection [39] Autores: Quanquan Li, Shengying Jin, Junjie Yan

Año: 2017

En: CVPR

Parten desde la arquitectura de R-CNN, es decir una red de propuesta de regiones y una red de clasificación sobre esas propuestas. Hacen básicamente lo mismo que en [59] pero considerando solo aquellas zonas donde hay un región propuesta por la *Region Proposing Network*.

Definen sus perdidas como $L = \lambda_1 L_m + L_{gt}$, donde $L_{gt} = L_{cls} + \lambda_2 L_{reg}$, es decir la perdida de clasificación y regresión de las regiones propuestas y la destilación se realiza similarmente a [59] usando un regreso r, sobre los espacios de las features originales indicadas por el *pooling* espacial de la RPN. $L_m = \frac{1}{2N} \sum_i \frac{1}{m_i} \|u_T - r(u_s)\|_2^2$

Learning Efficient Object Detection Models with Knowledge Distillation [5] Año: 2017

Autores: Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, Manmohan Chandraker

En: NIPS

Proponen un método de destilación sobre Faster-RCNN, una de las redes estado del arte en detección de objetos compuesta por 3 módulos, un *backbone* común de extracción de mapas de características , una red de propuesta de regiones **RPN** y una red de clasificación y regresión del bounding box **RCN** sobre las regiones propuestas de la **RPN** y el backbone. Partiendo desde Fitnets [59], proponen una serie de modificaciones para adaptarse al objetivo compuesto (en regresión y clasificación) de la detección y el desbalance de clases.

Tomando N y M como tamaños de batch para **RCN** y **RPN**, L_{cl} como la perdida de

clasificación, L_{reg} como perdida de regresión y λ con γ como hiperparámetros de balance entre perdidas fijados en 1 y 0.5 respectivamente, las perdidas de los módulos **RCN** y **RPN** toman las siguientes formas respectivamente:

$$\mathscr{L}_{RCN} = \frac{1}{N}\sum_i L_{cl}^{RCN} + \lambda \frac{1}{N}\sum_i L_{reg}^{RCN}$$

$$\mathscr{L}_{RPN} = \frac{1}{M}\sum_i L_{cl}^{RPN} + \lambda \frac{1}{M}\sum_i L_{reg}^{RPN}$$

Con lo que la perdida total toma la forma.

$$\mathscr{L} = \mathscr{L}_{RPN} + \mathscr{L}_{RCN} + \gamma \mathscr{L}_{Hints}$$

Se explicaran en detalle el balance de clases y la perdida destilada con margen.

Balance de clases en clasificación

A diferencia del caso de destilación original donde el objetivo es el de clasificación de imágenes, el caso de detección en imágenes tiene que lidiar con un desbalance de clases entre el fondo y los objetos. Para esto la destilación original propuesta por Hinton $\mathscr{L}_{cl} = \lambda L_{ce} + (1-\lambda) L_{dist}$ se modifica para balancear mejor las predicciones en la perdida de destilación, introduciendo un w_c que amplifica la penalización de la perdida cuando se trata de clase fondo.

$$L_{dist} = -\sum w_c P_t log P_S$$

Perdida destilada con margen

En casos de regresión del *bounding box* la perdida se modifica para penalizar solo en casos que la perdida de la red estudiante sea mayor que la perdida de la red tutora mas un margen m.

$$L_{bb}(R_S, R_T, y) = \begin{cases} \|R_S - y\|_2^2 & \text{si } \|R_S - y\|_2^2 + m > \|R_T - y\|_2^2, \\ 0 & \text{si no} \end{cases}$$

Con lo que la perdida de regresión se pasa a ser una combinación entre la regresión de la perdida L_1 sobre las etiquetas reales L_S y la perdida destilada L_b.

$$L_reg = L_S + \nu L_b$$

Object detection at 200 Frames Per Second [48] Año: 2018

Autores: Rakesh Mehta, Cemalettin Ozturk

En: CVF

Realizan algunas modificaciones a tiny YOLO [58] y lo entrenan usando destilación para obtener un modelo que funciona a mas de 200 FPS sin perder mAp. Entre las modificaciones cuentan reducción de profundidad y cantidad de canales en las capas de extracción de mapas de características . Uso de *stacking layers*, es decir adición de primeras capas a capas finales para suplir poder de computo. La introducción de una modificación de *non maximum supression* basada en los mapas de características . Además de la modificación de la función de perdida, aprovechando la componente de objectness de YOLO. Se explicaran las 3 ultimas modificaciones.

- **Destilación y perdida escalada por objectness**

La forma de predecir de YOLO es bastante especial ya que divide la imagen de entrada en un campo de 13x13 celdas sobre las que se predicen simultáneamente los puntajes de la clasificación de clases, las *bounding boxes* que ubican los objetos dentro de las imágenes y el *objectness* que da una medida de probabilidad de que en cierta zona de la imagen haya algo. Para clasificación la perdida suele ser *cross entropy*, para *bounding box* y objectness distancia L_1 o L_2. Así, la perdida total queda como

$$\mathscr{L}_{yolo} = \mathscr{L}_{cl}(p_i, \hat{p}_i) + \mathscr{L}_{bb}(b_i, \hat{b}_i) + \mathscr{L}_{obj}(o_i, \hat{o}_i)$$

Una problematica de la formulacion de la perdida en este caso es que al entrenar simulataneamente todo, se realiza aprendizaje de clases y bounding boxess en aquellas celdas donde no hay nada . Evitan este problema escalando según objectness las perdidas distintas de objectnes. Definiendo una ganancia λ_D, o^{gt} como el ground truth, $\hat{o}$ como la salida de la red estudiante y o^T como la salida de la red tutora la perdida de objectness pasa a ser.

$$\mathscr{L}'_{obj}(o_i, \hat{o}_i) = \mathscr{L}_{obj}(o_i^{gt}, \hat{o}_i) + \lambda_D \mathscr{L}_{obj}(o_i^T, \hat{o}_i)$$

Similarmente, las perdidas de clasificacion y bounding box pasan a ser las siguientes. Notese la presencia de objectness escalando el peso de la perdida.

$$\mathscr{L}'_{cl}(o_i, \hat{o}_i) = \mathscr{L}_{cl}(o_i^{gt}, \hat{o}_i) + o_i^T \lambda_D \mathscr{L}_{cl}(o_i^T, \hat{o}_i)$$

$$\mathscr{L}'_{bb}(o_i, \hat{o}_i) = \mathscr{L}_{bb}(o_i^{gt}, \hat{o}_i) + o_i^T \lambda_D \mathscr{L}_{bb}(o_i^T, \hat{o}_i)$$

- **Modificaciones de arquitectura**

Para reducir la profundidad de la arquitectura original, en la red estudiante ocurre una reducción dimensional importante desde las primeras a las ultimas capas. Esto trae consigo perdida en la capacidad de cómputo de la red, lo cual es suplido por conexiones o *stacking layers* que permiten llevar información desde las primeras capas a las últimas. Esto ultimo se realiza usando convoluciones de 1x1 para comprimir la información y escalarla a las dimensiones de las capas posteriores.Estas se componen de algunas capas convolucionales de 1x1 para computaciones antes de llegar al *header* de la red.

- **FN-NMS**

En YOLO, en los casos que un objeto corresponda a múltiples celdas o *bounding boxes* la transferencia de conocimiento a una red estudiante puede resultar en redundancia que ocasione perdida de desempeño. YOLO usa *non máximum supression* para entrenar en estos casos. En el paper esto se mejor usando un filtro que mapea desde todas las detecciones cercanas de una misma clase a la de mayor *objectness*. El entrenamiento entonces se realiza sobre solo esta detección, evitando toda redundancia y al costo de poder pasar por alto objetos múltiples en celdas.

Reproduccion de mapas de características

An Embarrassingly Simple Approach for Knowledge Distillation [16] Año: 2019

Autores: Mengya Gao, Yujun Shen, Quanquan Li, Junjie Yan, Liang Wan, Dahua Lin, Chen Change Loy, Xiao Tang

En: Arxiv, aun no publicado.

En general en los aprendizajes multi tareas es necesario usar varias perdidas distintas, cada una de ellas ponderada por un hiperparametro λ para ponderar cuanto afecta cada una de las perdidas al momento de realizar backpropagation. La elección de estos hiperparametros puede ser una tarea difícil, especialmente considerando que el valor óptimo suele cambiar de dataset a dataset sin otra forma de encontrarlo que el uso de varias instancias de prueba y error. El paper postula una manera simple y eficiente de evitar este problema desacoplando las etapas de aprendizaje en el caso de destilación de features.

En general, la mayoría de las redes puede separarse entre una etapa de extracción de features llamada *backbone* y una de clasificacion o detección llamada *header*. Como se puede ver en [73] estas suelen ser bastante independientes de la tarea, cosa que se aprovecha en este paper para separar los entrenamientos en una etapa de aprendizaje *greedy* de los mapas de características de una red tutora y una etapa de aprendizaje de las etiquetas finales usando *ground truth* y el *backbone* fijo.

Para la primera etapa se hace uso de un entrenamiento bloque por bloque, es decir, se toma cierta profundidad de capa a reproducir y se reproducen las features de la red tutora en la red estudiante, luego se fijan los pesos en la red estudiante y se aprenden los features a una profundidad un poco mayor y así sucesivamente hasta destilar el *backbone* completo. Finalmente se reproduce la ultima capa usando solo la red estudiante y el *ground truth* de la tarea a entrenarse. El aprendizaje de los mapas de características se realiza usando una perdida de distancia L2 entre las features de la red tutora y estudiante.

$L_{SSKD} = \|f_t^2 - f_s^2\|_2$

Knowledge Distillation with Feature Maps for Image Classification Año: 2019

Autores: Wei-Chun Chen, Chia-Che Chang, Chien-Yu Lu, and Che-Rung Lee

En: ACCV

Usan GAN para entrenar redes que reproducen features. El sistema completo se compone de un generador G que se usa para imitar los features, un discriminador D que sirve para entrenar el generador y un clasificador C mediante el cual se clasifica y que toma como entrada las features generadas por G.

$L_{\text{adv}D} = \frac{1}{2}\left[D(G(x))\right]^2 - \frac{1}{2}\left[D(F_T(x)) - 1\right]^2$

$L_{\text{adv}G} = \frac{1}{2}\left[D(G(x))\right]^2$

La perdida total para el generador es.

$L_G = L_{\text{adv}G} + \alpha L_{KD}$

6.2. Aprendizaje Profundo Bayesiano

Una red neuronal bayesiana es una red neuronal donde los pesos en vez de ser valores puntuales, son variables aleatorias. Estas existen al menos desde el 2006, cuando Hinton propone un modelo llamado *Deep Belief Networks*, donde paradójicamente el concepto *deep* usado no tiene nada en común con como actualmente se entiende en *Deep Learning*. Estas cayeron relativamente en desuso durante bastante tiempo debido a lo difícil que era tanto el calculo de los pesos durante el entrenamiento como la inferencia con ellas. Sin embargo, la posibilidad que estas dan de poder cuantificar la incerteza de la predicción, además de la predicción misma, hacen de estas una alternativa bastante interesante al aprendizaje profundo convencional.

Si bien el uso de muestreos, estructuras probabilísticas y modelos gráficos usando probabilidad bayesiana dentro de aprendizaje profundo no tiene mucho de novedoso, ya que este tipo de técnicas se remontan al menos hasta VAE [35] del 2013 y GAN [19] del 2014. No es si no hasta las publicaciones de Yarin Gal en su paper del 2015 [15] y su tesis de doctorado [50] sobre modelación de la incerteza en aprendizaje profundo , que el concepto comienza a tomar peso propio. En resumidas cuentas, en el estado del arte del aprendizaje profundo se había estado usando de manera no intencionada redes neuronales bayesianas desde hace mucho tiempo, aparentemente sin reparar en las implicaciones que esto tenia.

Mas en especifico, toda red entrenada con algún tipo de técnica de regularización estocástica (SRT), como *Dropout* o ruido aditivo gaussiano es en esencia, una red neuronal bayesiana entrenada con inferencia variacional[1]. Luego, la única diferencia entre una red convencional y una red bayesiana es la forma en que se realiza la inferencia. Poniendo como ejemplo *Dropout*, convencionalmente se toma una ponderación de todas las neuronas de la capa con dropout, escalando cada salida por $\frac{1}{1-p}$, siendo p la probabilidad fijada para dejar una neurona fuera. En el caso de la red bayesiana se realizan varios muestreos de la salida $\hat{y}$ de la red y se usa un estimador para obtener la esperanza y otro para la dispersión. En regresión se usa media y varianza y en clasificación la moda de las muestras y alguna medida como *varation ratio*, *predictive entropy* o *mutual information*.

[1] No se profundizará mucho en esto por la extensión que toma explicarlo. Se recomienda referirse al blog de Yarin Gal para una explicación sucinta y concreta o bien a su tesis de doctorado [50] para una explicación en profundidad. `http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html`

6.2.1. Destilación Bayesiana

Dropout Distillation [4] Año 2016
Autores: Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder En: JMLR La destilación en este caso se centra en destilar una red bayesiana en la linea de lo expuesto por Yarin Gal en [15]. El paper se centra en la destilación de la esperanza de la red bayesiana en una red determinista. Esto con el objetivo realizar inferencia en tiempo real, sin necesidad estimar sobre una cantidad mayor de muestreos, no necesariamente buscando reducir el tamaño mismo de la red.

En su forma mas básica, esta tarea consiste en encontrar pesos de la red bayesiana $\hat{W}$ tal que el error de la inferencia sea el mínimo. Esto se hace entrenando una red de la misma estructura de la red original, usando divergencia KL entre la media de la red bayesiana y la salida de la red determinista. El paper también prueba destilando hacia una red mas pequeña, logrando un desempeño mejor que la red pequeña original usando *cross entropy* pero peor que el de la red original.

Zero-shot Knowledge Transfer via Adversarial Belief Matching [49] Año: mayo 2019
Autores: Paul Micaelli, Amos Storkey

La gran mayoría de los modelos interesantes que se usan hoy en día se entrenan vía el uso de una cantidad muy grande de datos, los cuales en muchas ocasiones son de datasets privados. Por esta razón, muchas veces entrenar una red usando el dataset original no es posible. El paper se centra en la destilación del conocimiento de una red tutora sin datos.

Toman una técnica de original de procesos gaussianos llamada *inducing point methods*, que básicamente consiste en tomar puntos representativos de la base de datos, los cuales son usados en vez de la base completa. Con esto se logra reducir la complejidad computacional al momento de la inferencia. En este contexto, se crea un método de destilación de dataset, el cual genera pseudoimagenes que cumplen el mismo objetivo que la técnica anteriormente expuesta. En la practica, los inducing points corresponden a muestras de una red generadora entrenada con la red para maximizar la divergencia KL entre la red estudiante y el generador.

Usan un generador, una red estudiante y una red tutora pre-entrenada. Suman a la perdida de la red estudiante el uso de una perdida que incluye attention maps en algunos bloques comunes de la red estudiante con la red tutoria, de la misma manera que las vistas en las de layer level distillation. La perdida para el generador es basicamente menos la divergencia KL entre la red estudiante y la red tutora. Para la red estudiante es la divergencia KL entre a red estudiante y la red tutora mas un termino de regularizacion sobre la atencion en las features de la red.