Visión por Computadora II - CEAI - FIUBA

Profesores:

- Cavalieri Juan Ignacio - juanignaciocavalieri@gmail.com
- Cornet Juan Ignacio - juanignaciocornet@gmail.com
- Seyed Pakdaman - khodadad.pakdaman@gmail.com

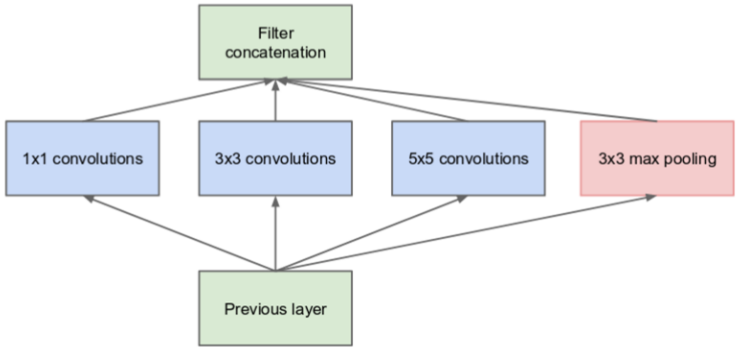
Segunda clase:

- Arquitecturas clásicas:
  - Inception
  - ResNet
  - Otras..
- Transfer Learning
- Implementación en PyTorch
  - Residual Network
  - Transfer Learning
- Presentación TP Integrador

Arquitecturas clásicas: Inception

En 2014 un equipo de Google presentó la primera versión de la serie de redes Inception en la competencia ImageNet, ganando en la tarea de clasificación. Como en todos los casos, la motivación principal estaba en obtener redes cada vez más profundas (más capas y más neuronas), que puedan mejorar las métricas actuales. Sin embargo, esto implicaba más probabilidad de sobreentrenamiento y más costo computacional.

Bajo estas premisas se creó lo que se conoce como un bloque Inception:



Szegedy, et al., 2014. Going deeper with convolutions. Link

Arquitecturas clásicas: Inception

Motivación

- El tamaño de las features de una misma clase puede variar entre distintas imágenes del dataset, lo cual dificulta la elección del tamaño de filtro ideal para el problema en cuestión.
- Las redes muy profundas son más propensas a sobreentrenarse. Esto también dificulta la propagación de gradientes a lo largo de toda la red.
- Las redes convolucionales con muchas capas apiladas son costosas computacionalmente.

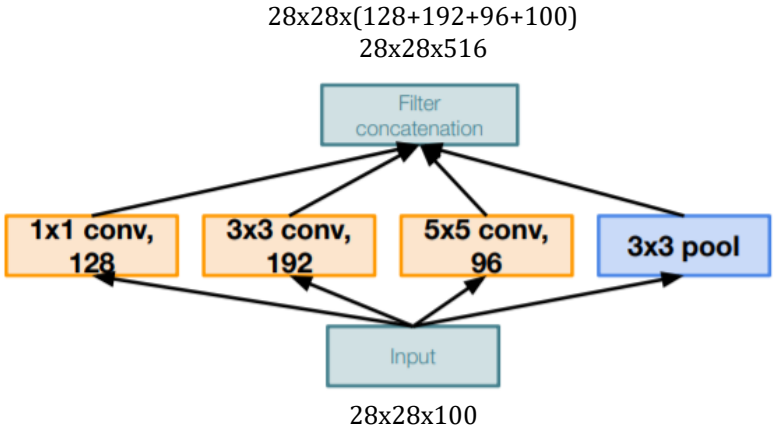
Arquitecturas clásicas: Inception

Características principales:

- Introducción del bloque Inception: conexiones más esparsas, con diferentes tamaños de filtros para captar mejor features de múltiples tamaños.
- Bottleneck layers basadas en convoluciones 1x1 para reducir la cantidad de operaciones necesarias para el cómputo de la red.
- Clasificadores auxiliares en capas tempranas para mejorar el entrenamiento.
- 22 capas con solo 6,7 M de parámetros.

Bloque Inception

- La red cuenta con 9 bloques Inception separados por capas de Max-Pooling en algunos lugares..
- Estos bloques no modifican el ancho y alto de los volúmenes, solo la cantidad de canales. Por lo tanto, cada una de las operaciones dentro de cada bloque debe utilizar el padding necesario para que esto ocurra.
- Las salidas de cada una de las operaciones se concatenan para formar la entrada de la próxima capa.

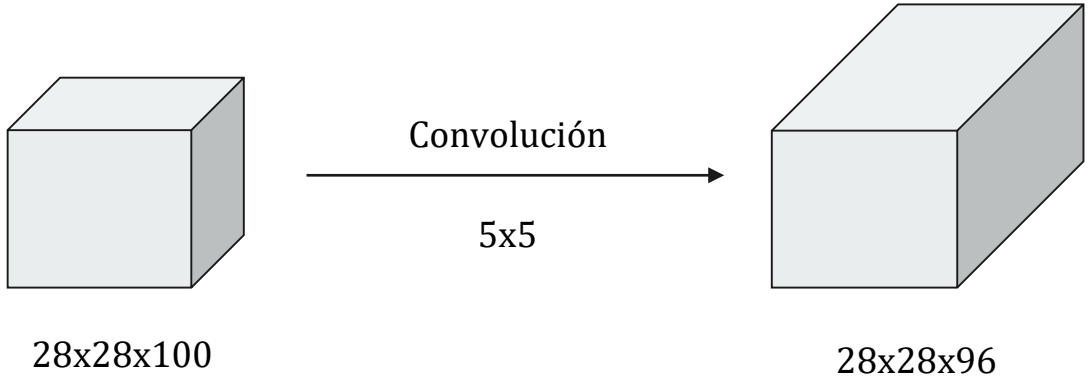


Sin embargo, este esquema tiene un gran costo computacional debido a la cantidad de convoluciones involucradas y, por otro lado, la capa de pooling fuerza a la salida a tener más cantidad de canales que la entrada.

conv 1x1: **1x1x128x28x28x100 = 10M ops**
conv 3x3: **3x3x192x28x28x100 = 135,4M ops**
conv 5x5: **5x5x96x28x28x100 = 188,1M ops**
Total: 333,5M operaciones

Bloque Inception: Costo Computacional

Si analizamos el caso de la convolución de 5x5:

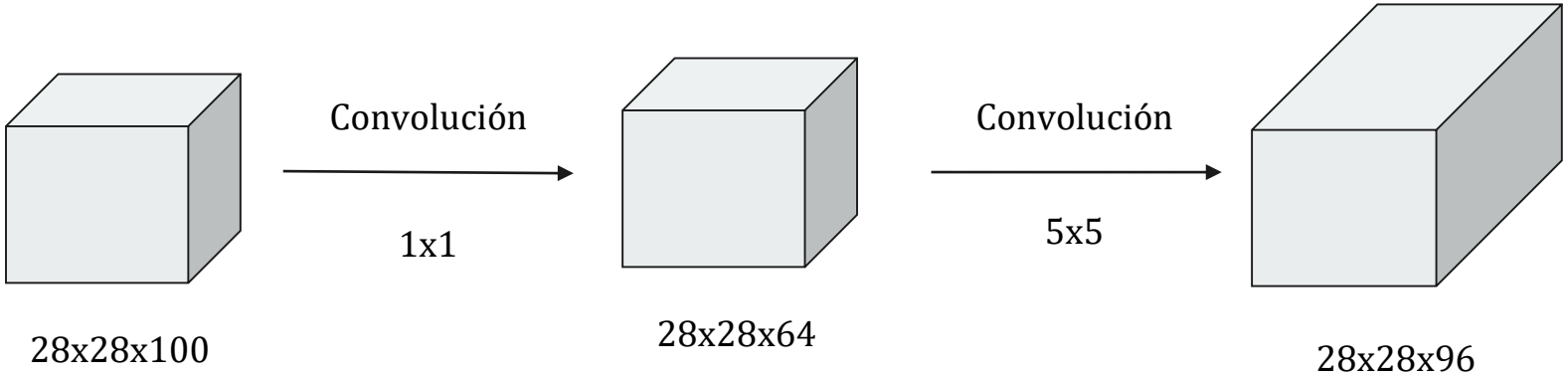


La cantidad de multiplicaciones involucradas es:

Para obtener un valor de la salida: 5 x 5 x 100 = 2500 ops
Para obtener todos los valores de la salida: 2500 x 28 x 28 x 96 = **188,16 M ops**

Bloque Inception: Costo Computacional

Si, en cambio, se antepone una convolución 1x1 para reducir la dimensionalidad antes de la convolución de 5x5:

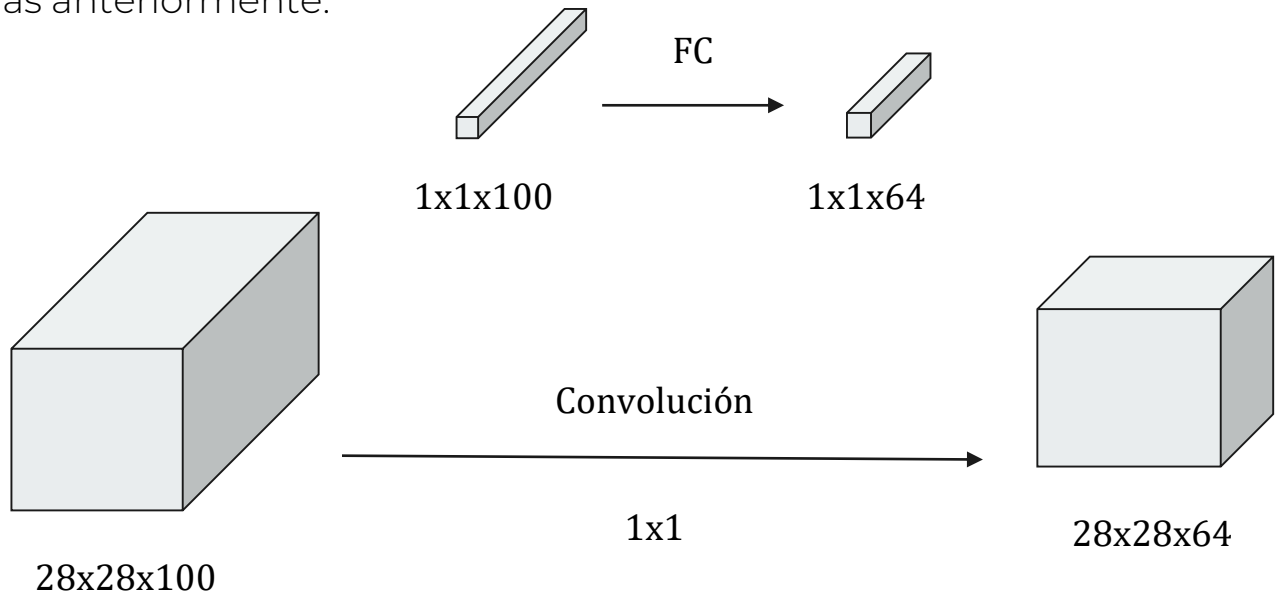


La cantidad de multiplicaciones involucradas es:

En la convolución de 1x1: 1 x 1 x 100 x 28 x 28 x 64 = **5 M ops**
En la convolución de 5x5: 5 x 5 x 64 x 28 x 28 x 96 = **120,4 M ops**
Total: 125,4 M ops

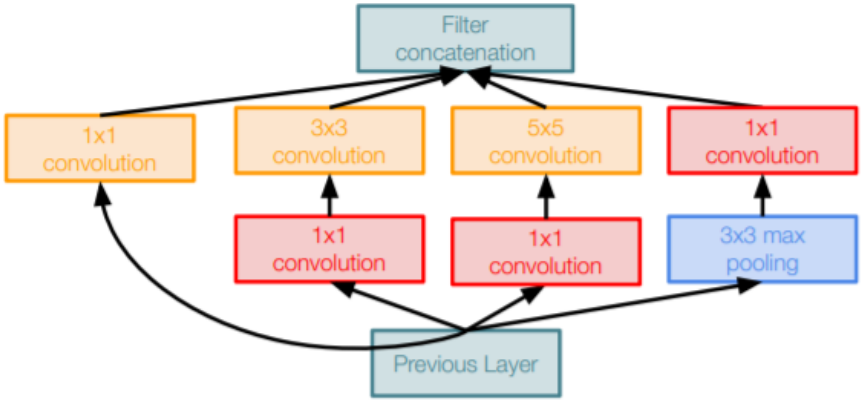
Bloque Inception: Costo Computacional

La convolución de cada filtro de 1x1 se puede interpretar como aplicar la misma capa FC a cada píxel de entrada. Estas convoluciones preservan las dimensiones horizontal y vertical pero reducen la dimensionalidad en profundidad combinando las features extraídas anteriormente.

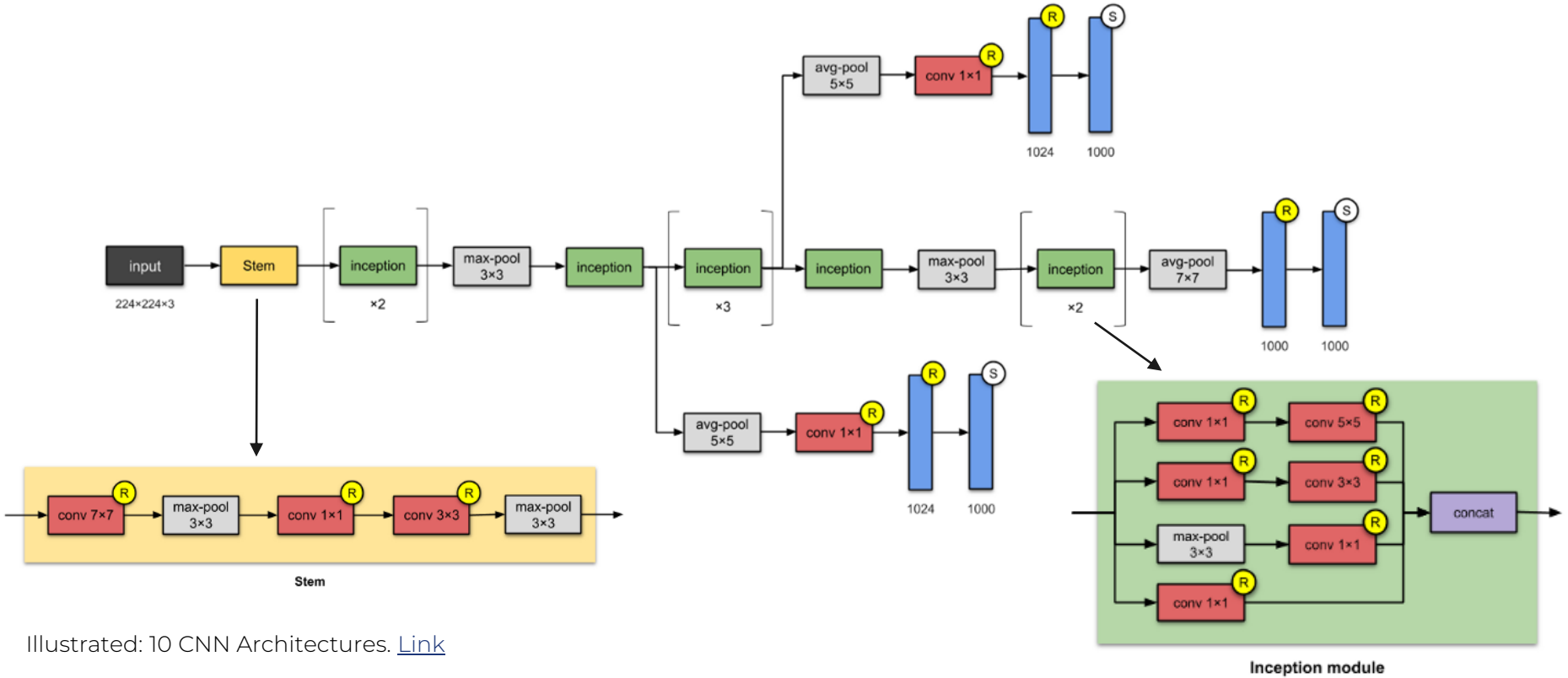


Bloque Inception: Bottleneck layer

- Permiten reducir la cantidad de canales luego de la capa de Max-Pooling.
- Reducen la dimensionalidad antes de las convoluciones de 3x3 y 5x5 para mejorar los tiempos de cómputo.
- Reducen la cantidad de parámetros dentro del bloque Inception.
- Utilizan funciones de activación ReLU por lo que agregan no-linealidad.



Inception V1



Illustrated: 10 CNN Architectures. [Link]

Arquitectura Inception: Clasificadores Auxiliares

Dada la gran profundidad que adquiere la red neuronal, se presenta el problema de que, a medida que transcurre el entrenamiento, los gradientes que llegan a las capas intermedias de la red se “apaguen” (vanishing gradient) ralentizando o, incluso, anulando su entrenamiento.

Para compensar esta situación, se le agregaron a la red, dos ramificaciones con clasificadores, las cuales influyen en el cómputo del error total en cada forward pass y, por lo tanto, reforzarán las señales de los gradientes provenientes de la salida original.

Para computar el error total de la red se suman los valores de las 3 salidas, ponderando a los clasificadores auxiliares por 0,3. Luego, cuando se infiere sobre la red ya entrenada, estos clasificadores auxiliares no son tenidos en cuenta.

Nuevas versiones de Inception

Luego de esta primera versión, el equipo de Google continuó trabajando y mejorando la performance del modelo, por lo que posteriormente publicaron papers con nuevas versiones de la red. Entre estas, las más destacadas son:

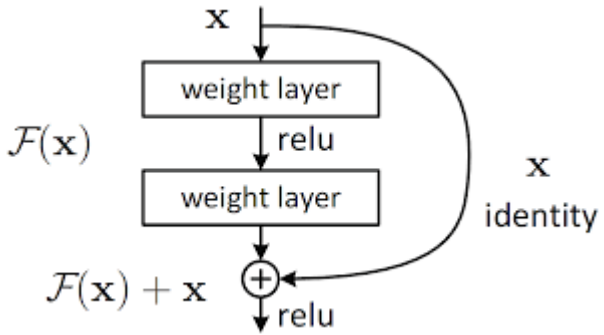
- **Inception V3** (Link Paper):
  - Agregaron Batch Normalization
  - Factorizaron ciertas convoluciones de $n \times n$ en convoluciones asimétricas de $1 \times n$ y $n \times 1$.
  - Redujeron el tamaño de los filtros de varias convoluciones, concatenando capas donde fuera necesario.
  - Se utilizaron 3 formatos de bloques inception diferentes.
  - Aumentaron a 25 M de parámetros.
- **Inception V4** (Link Paper):
  - Se introdujeron “reduction blocks”, es decir, bloques de tipo inception que reducen las dimensiones de la red.
  - Aumentaron a 43 M de parámetros.

Arquitecturas clásicas: ResNet

Arquitecturas clásicas: ResNet

Para 2015 agregar más capas a una red convolucional ya no garantizaba un modelo más preciso. Ese año un equipo de Microsoft introdujo las conexiones residuales en una red neuronal convolucional a través de las redes ResNet y obtuvo el primer lugar en la competencia ImageNet. En su paper se presentan redes de **hasta 150 capas** que mejoran en las métricas de error a cualquiera de las anteriores.

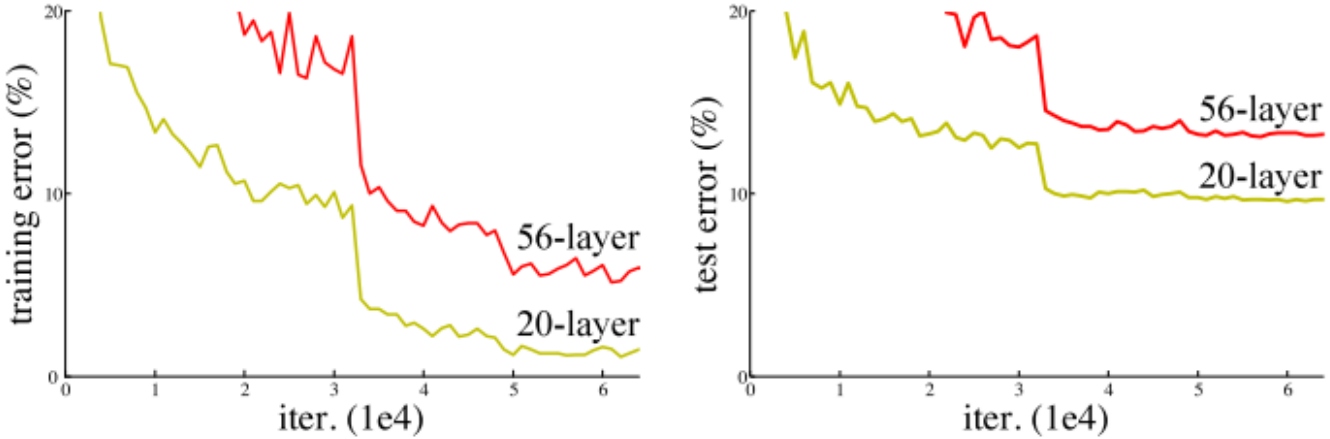
Las conexiones residuales son, hoy en día, uno de los conceptos que se siguen aplicando en el desarrollo de redes neuronales y que marcaron un antes y un después en su desempeño.



He, et al., 2015. Deep Residual Learning for Image Recognition. Link

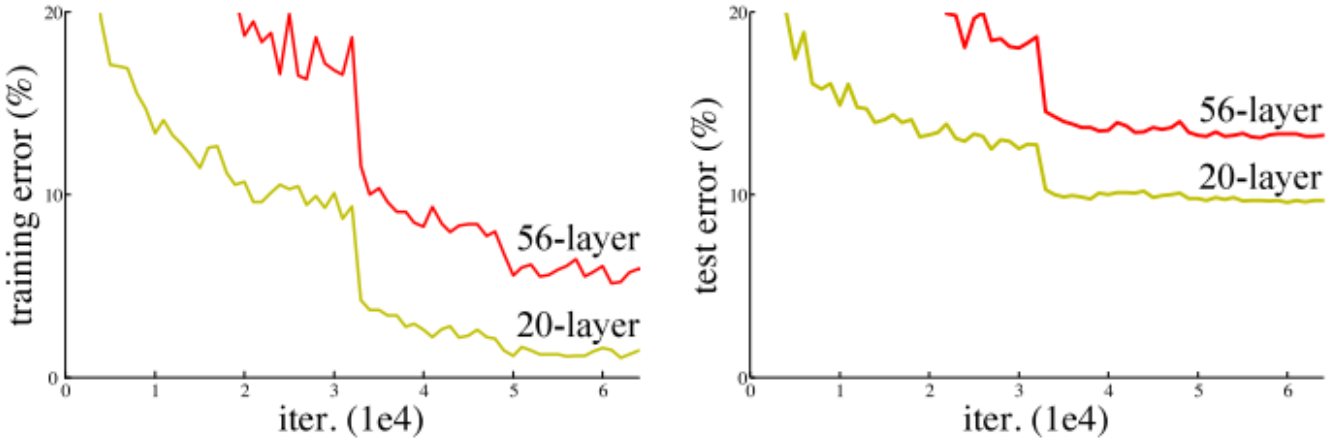
Degradation Problem

Un comportamiento contraintuitivo ocurre cuando se le agregan más capas a una red neuronal profunda, llegando al punto que el error sobre cualquiera de los conjuntos de datos no disminuye o, incluso, en algunos casos aumenta luego de varias iteraciones. Esto, claramente, no es debido a un **sobreentrenamiento** del modelo y tampoco estaría completamente relacionado a los **vanishing/exploding gradients**.



Degradation Problem

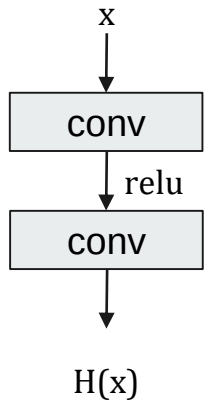
A este comportamiento se lo denomina **degradation problem**. A medida que la profundidad de la red crece, el accuracy que esta puede alcanzar se “satura”, lo cual se puede interpretar como que la red aprende todo lo que puede antes de llegar a la última capa y luego comienza a empeorar a medida que su profundidad aumenta. A este problema se lo considera un problema de optimización.



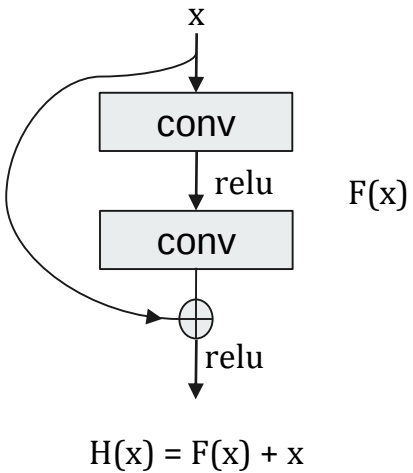
Conexiones Residuales

La solución propuesta es, entonces, utilizar las conexiones residuales de forma tal que la información fluya a través de la red, lo que reduce la cantidad de capas que deben aprender la misma información.

Red “plana”

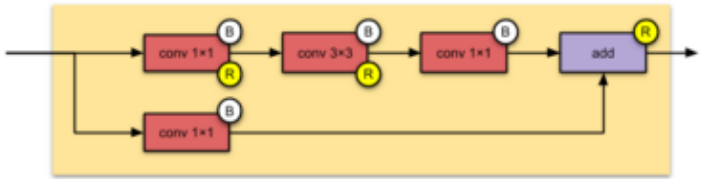


Red “residual”

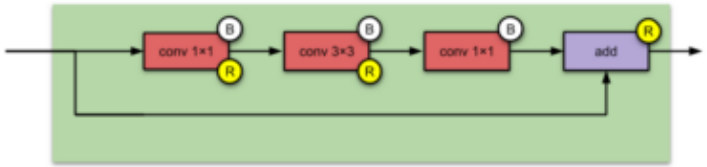


Además, las conexiones residuales también ayudan a estabilizar el entrenamiento de la red, lo que permite que la red converja más rápidamente y con mayor precisión.

ResNet-50



Conv block



Identity block

Illustrated: 10 CNN Architectures. Link

ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet

Demostración de funcionamiento de redes profundas con y sin conexiones residuales.

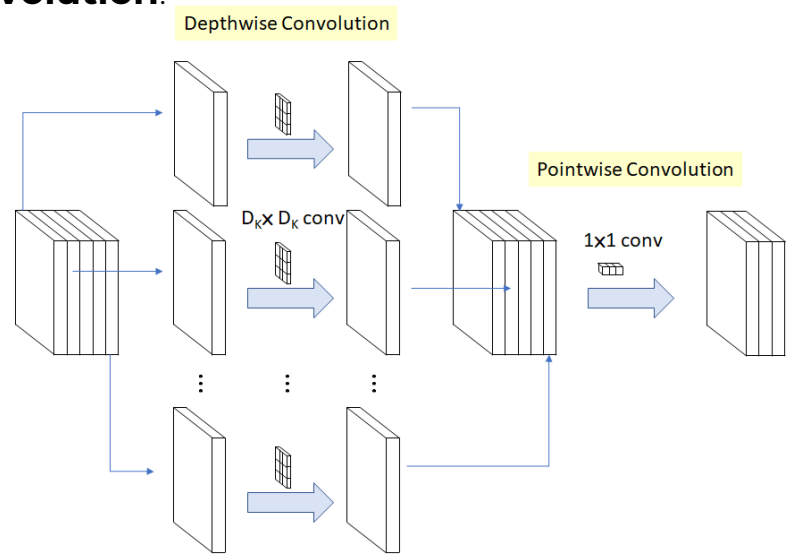
Colab ResNet

Otras arquitecturas: MobileNet

Publicada por un equipo de Google en el 2017. Buscaba optimizar al máximo el costo computacional y la memoria requerida para correr una red convolucional sin degradar las métricas de la misma. Para ello reemplazaron las convoluciones convencionales por **depthwise separable convolution**.

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

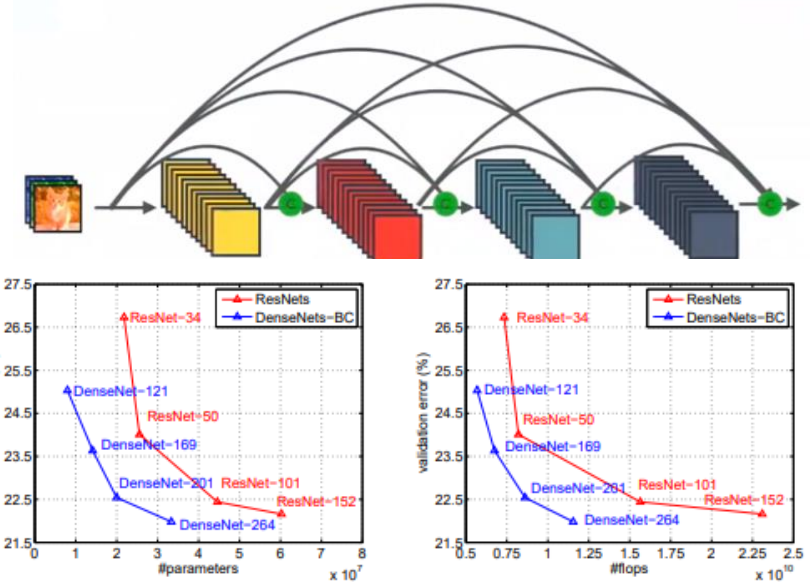


Howard, et al., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Link

Otras arquitecturas: DenseNet

Esta arquitectura lleva el concepto de conexión residual un paso más allá. Está formada por **bloques densos**, en los cuales, la salida de cada capa convolucional es trasladada a la entrada de todas las capas posteriores y concatenada con los resultados de la última.

- Mejora el flujo de los gradientes dentro de la red.
- Permite que las capas tomen como entradas features más diversas.
- Mantiene presentes las features de más bajo nivel, lo cual, hace que la red performe mejor cuando hay poca cantidad de datos para entrenar.



Huang, et al., 2018. Densely Connected Convolutional Networks. Link

Otras arquitecturas

Además de las que vimos hasta ahora, existen otras arquitecturas, por lo general más nuevas, basadas en capas convolucionales, que se desarrollaron para problemas de clasificación. Entre ellas están:

- Xception: Paper
- MobileNet v2: Paper
- MobileNet v3: Paper
- ResNeXt: Paper
- EfficientNet: Paper

Y muchas otras más... este campo sigue siendo estudiado activamente por lo que siempre hay papers con propuestas nuevas.

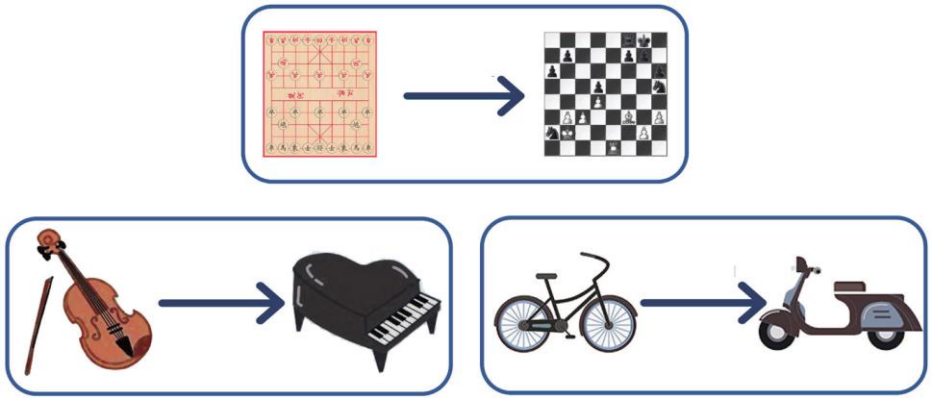
Resumen

- **AlexNet**: Nos mostró que las CNN son la mejor opción para el procesamiento de imágenes.
- **VGGNet**: Nos mostró que modelos más grandes funcionan mejor.
- **Inception**: Nos mostró cómo podemos optimizar la performance de las redes neuronales en cuanto a cómputo realizado, sin prescindir de los resultados en métricas.
- **ResNet**: Nos mostró cómo entrenar redes cada vez más grandes y que el beneficio disminuye a medida que las red crece.

Transfer Learning

¿Qué es?

La transferencia de conocimiento (transfer learning) consiste en mejorar la performance de un modelo objetivo sobre un dominio objetivo a partir de reutilizar el conocimiento ya aprendido por otro modelo sobre un dominio distinto, pero que guarda cierta relación con el primero. Este concepto está derivado de la psicología humana relacionada al aprendizaje.



Zhuang, et al., 2020. A Comprehensive Survey on Transfer Learning. Link

Transfer Learning

Cuando se trata de redes neuronales profundas aplicadas a problemas de visión, **es necesario contar con una enorme cantidad de datos etiquetados** para lograr un entrenamiento satisfactorio y que el modelo pueda resolver la tarea en cuestión.

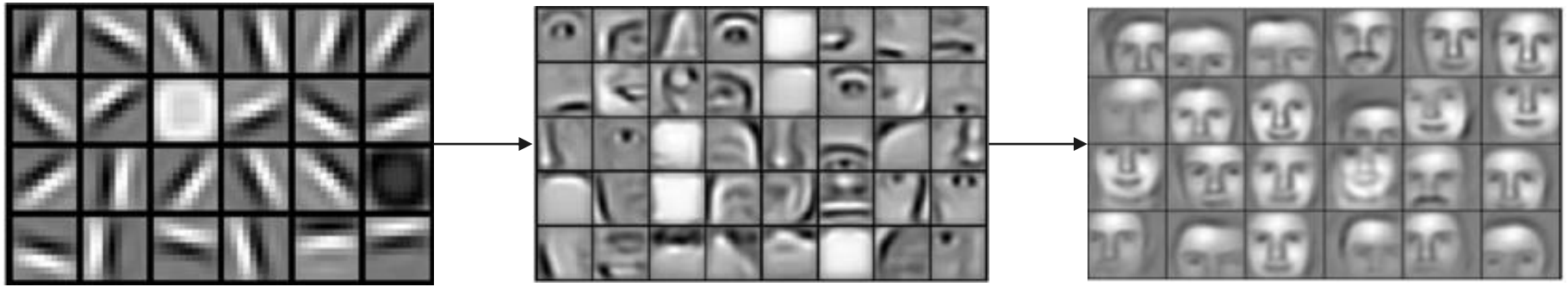
Sin embargo, en la mayoría de los casos, conformar un dataset con la cantidad suficiente de datos etiquetados y que estos tengan una distribución similar a los datos de testeo, **resulta bastante complicado y costoso**. Incluso, a veces, el simple hecho de conseguir los datos sin las etiquetas es todo un desafío.

Por estas razones, en la práctica, es muy común utilizar este tipo de técnicas para lograr modelos que resuelvan la tarea específica que nosotros requerimos.

Ribani, et al., 2019. A Survey of Transfer Learning for Convolutional Neural Networks. Link

Transfer Learning

En redes convolucionales, lo que ocurre normalmente, es que **las primeras capas aprenden patrones básicos** que, en principio, **se repiten en cualquier conjunto de datos de imágenes** que tengamos. Por lo tanto, a la hora de entrenar un nuevo modelo con un conjunto de datos distinto, **podríamos aprovechar esas capas ya entrenadas en la nueva red**.



Transfer Learning

Transfer Learning

Cuando se trabaja con imágenes, existen dos estrategias ampliamente utilizadas a la hora de realizar transfer learning:

- **Feature extraction**

Consiste en tomar un modelo ya preentrenado en algún dataset lo suficientemente genérico (Ej: ImageNet) y eliminar la última o últimas capas. Luego la nueva salida de la red se utiliza como vector de features para alimentar un nuevo algoritmo que realice la clasificación del problema objetivo.

- **Fine Tuning**

En este caso no solo se reemplaza la última o últimas capas de la red sino que el proceso de entrenamiento se realiza, también, sobre una porción de las capas ya pre entrenadas.

Transfer Learning

En términos generales las situaciones se pueden resumir en:

1. **Dataset pequeño y con datos similares:** No es conveniente realizar fine tuning dado el riesgo de sobreentrenamiento por los pocos datos. Por lo general, se entrena un clasificador basado en las features extraídas de una de las últimas capas.
2. **Dataset pequeño y con datos distintos:** Al igual que en el caso anterior, no es conveniente realizar fine tuning. Además, tampoco es recomendable extraer las features de alguna de las últimas capas dado que estas han aprendido características más específicas del dominio original.
3. **Dataset grande y con datos similares:** Dado que hay mayor cantidad de datos podemos analizar el uso de fine tuning sin riesgo de caer en sobreentrenamiento.
4. **Dataset grande y con datos distintos:** En este caso, el fine tuning se puede realizar reentrenando aún más capas. También se podría analizar realizar un entrenamiento con parámetros inicializados aleatoriamente.

Transfer Learning

Precauciones a tener en cuenta para su implementación

- Cuando se realiza transfer learning, **el learning rate suele configurarse en valores más bajos** comparados a los que se utilizan para entrenar desde cero el mismo modelo. Esto para evitar que los pesos de la red ya entrenados se modifiquen mucho.

- Hay que tomar precauciones con los **tamaños de imágenes** que se usan para entrenar, teniendo en cuenta el tamaño de imagen que se utilizó para pre-entrenar el modelo:
  - Dependiendo de la variación puede ser necesario **reemplazar una determinada cantidad de capas** de la red.
  - Dependiendo de la variación puede que las **features aprendidas** por las primeras capas convolucionales correspondan a una escala de imagen diferente.

Transfer Learning

Negative Transfer

El Negative Transfer hace referencia a las ocasiones en las que el proceso de transfer learning lleva a una disminución de la performance del modelo sobre la tarea objetivo, en comparación con las tareas originales para las que fue entrenado.

Esto ocurre porque el conocimiento que ya se tiene puede interferir con las nuevas tareas que se intentan aprender o porque la forma en la que se realiza el transfer learning hace que dicho conocimiento no se transfiera correctamente.

Por ejemplo, supongamos que se tiene una red pre-entrenada para clasificar imágenes de perros y gatos y se quiere usarla para clasificar imágenes de plantas. Es posible que las características aprendidas por la red pre-entrenada no sean relevantes para la tarea de clasificación de plantas y que, en cambio, obstaculicen el rendimiento del modelo.

Wang, et al., 2019. Characterizing and Avoiding Negative Transfer. Link

Few Shot Learning

Cuando se cuenta con muy pocos datos de entrenamiento las técnicas tradiciones de Transfer Learning ya no nos sirven. Una forma de resolver este tipo de problemas es utilizar Few Shot Learning (FSL).

Algunos casos especiales, dentro de las técnicas de este estilo son **One Shot Learning**, referido a cuando solo tenemos un ejemplo etiquetado de cada clase que queremos clasificar, y **Zero Shot Learning** para cuando no contamos con ningún dato etiquetado para que nuestro modelo aprenda.

A diferencia de un problema tradicional, en donde intentamos que el modelo aprenda a clasificar a partir de los datos, en un problema de FSL vamos a buscar que el modelo aprenda a clasificar a partir de soluciones de tareas similares en donde los datos etiquetados son escasos. Este tipo de aprendizaje se lo denomina **Meta Learning**.

Links de utilidad: link, link y link

Ejemplos prácticos

Implementación de Transfer Learning con VGG19 sobre dataset de perros y gatos.

Colab Transfer Learning VGG19

Implementación de Transfer Learning con ResNet18 sobre dataset CIFAR10.

Colab Transfer Learning ResNet18

Ejercicio

Ejemplo de Negative Transfer

Colab Negative Transfer