# Curso: Técnicas de Aprendizaje Automático – Machine Learning

## Tema 8: Redes Convolucionales

Sonia I. Mariño, Rafael Perez, Leonardo Gomez Chavez

FaCENA - UNNE - 2023

# CNN

Fundamentos: surgimiento de CNN

Si se utiliza una RN MLP,

- El contexto espacial se pierde
- Incremento de pesos para imágenes más grandes
- Más pesos, requiere mayor tiempo y datos

Inspiración biológica

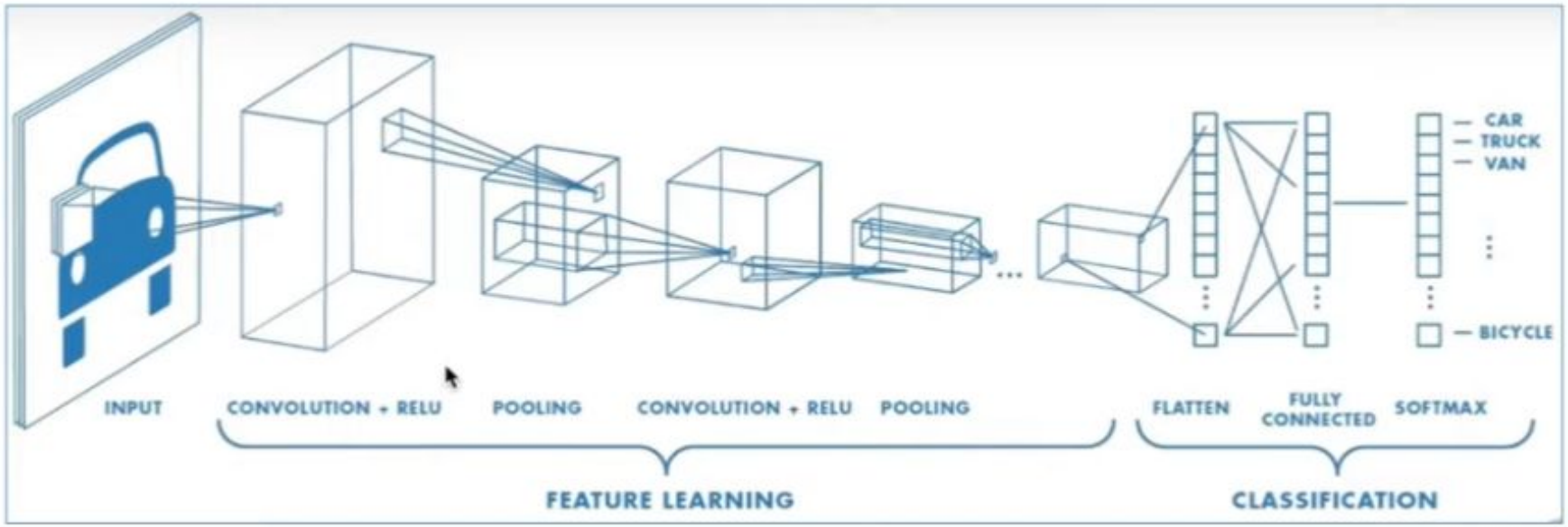
# Red neuronal convolucional (CNN)

Convolutional Neural Network (CNN)

- Arquitectura de redes neuronales profundas
- Algoritmo utilizado en ML / AA / DL para la detección de patrones: imágenes, señales de audio, …
    - *Deep Learning,* fomenta el aprendizaje de patrones mediante capas sucesivas, en las que capa a capa se van aprendiendo características más complejas de los datos.
    - Aplica para el procesamiento de datos con estructura de cuadrícula [matrices, vectores]
- Se originan en 1998

# CNN - ConvNet


| Líneas, curvas | formas | figuras | objeto |
|---|---|---|---|
| V1 | V2 | V4 | IT |



# CNN – Inspiración biológica

Las CNN se inspiran en la organización del sistema visual biológico

En la corteza visual, las neuronas:

- tienen un campo receptivo, no miran toda la imagen a la vez.
- se organizan jerárquicamente, y progresivamente se complejizan
  - capas inferiores, reconocen líneas horizontales y verticales, bordes, esquinas, curvas
  - capas superiores, se combina la información reconociendo formas, y se logra la representación para identificación de los objetos

*Es decir que las características se combinan progresivamente en las capas posteriores para formar representaciones más abstractas y jerárquicas de los datos.*

# CNN

***Convolución***

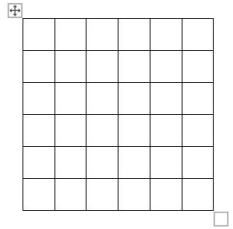
una operación matemática que toma dos funciones y genera una nueva función, la cual expresa como la primera es modificada por la segunda.

La primer función usualmente es sobre la cual se ejerce el cambio [canal / tensor] mientras que la segunda función [kernel] modifica a la primera.

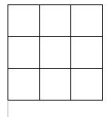
En ML, el canal y el kernel son matrices, por ello se trata de una convolución de matrices.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

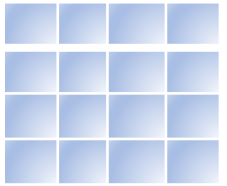
mapa de características o feature map:
salida o resultado de la operación o convolución



I



K



I * K

# CNN: Reconocimiento de imágenes

*An image is just a big grid of numbers between [0, 255]:*

- B & N: 800 x 600
- RGB: 800 x 600 x 3 (3 canales RGB)

# CNN - Desafios en imágenes

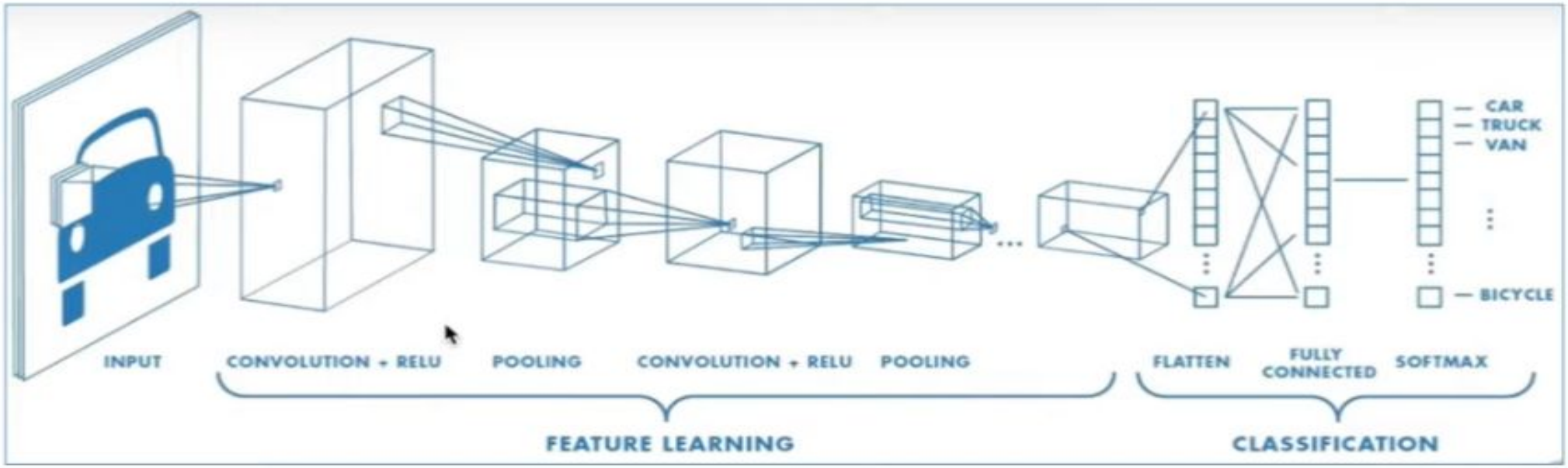
Según foco de cámara: los pixeles varían

- Iluminación
- Deformación de imágenes
- Ocultación
- Fondo
- Variación intraclase

# CNN -

Proceso de una CNN

- Fase Extracción de características, RN realiza operaciones de convoluciones y pooling (reducción)
- Fase Clasificación, capas *fully connected* actúan como un clasificador de las características detectadas.



# CNN – Capas y operaciones

- Capa de convolución: Aplica filtros para detectar características específicas en la entrada.
- Capa de activación: Introduce no linealidad en la red neuronal mediante funciones de activación, como la *función ReLU (Rectified Linear Unit).*
- Capa de agrupación/reducción (pooling) - Sub sampling: Reduce la dimensionalidad espacial de las características y preserva la información más relevante. Métodos: promedio, mínimo, máximo
- Capa totalmente conectada (Fully Connected Layer): Combina las características extraídas para realizar la clasificación o predicción final. Flatten: permite aplanar el volumen, aplica Backpropagation, Aplica *función SoftMax*

Adicional:

- Dropout
- Flatten
- Softmax

# Fase Extracción de características

RN realiza operaciones de convoluciones y pooling (reducción)

Capa convolución + RELU

Capa pooling

# Capa Convolución

En CNN, Capa de convolución,

- Filtros o kernel
- Neuronas
- Operación de convolución
- ReLU

Función rectificadora (*Rectified Lineal Unit, ReLU*):

$$\text{ReLU}(x) = \max(0, x)$$

# Capa Convolución

En CNN, Capa de convolución, **Filtros o kernel**

matrices que se “deslizan” sobre la imagen.

kernel o tensor móvil formado por pesos,

Inicio: se eligen filtros al azar

Y, se entrenan -como los pesos-. Es decir, los filtros son modificados por la RN, para extraer las características más relevantes.

El filtro debe tener la misma profundidad que la matriz de la imagen

# Capa Convolución

En CNN, Capa de convolución, **Neuronas**

Cada neurona de la red tiene asociado un filtro, como entrada un mapa de características y como salida un mapa de características.

En el aprendizaje,

- se actualizan los valores del filtro asociados a cada neurona, dado que son los pesos de NN
- las neuronas de la capa comparten los pesos o valores de sus filtros. Es decir, extraen el mismo patrón independientemente de la región de la imagen en la que se encuentre.

Se optimiza el entrenamiento, solo se realiza la operación de actualización de los pesos de un filtro para toda una capa.

# Operación de convolución

*Recordar: Una imagen es una matriz.*

Operación de convolución.

**Superponer un filtro sobre una imagen y deslizarlo sobre ella.**

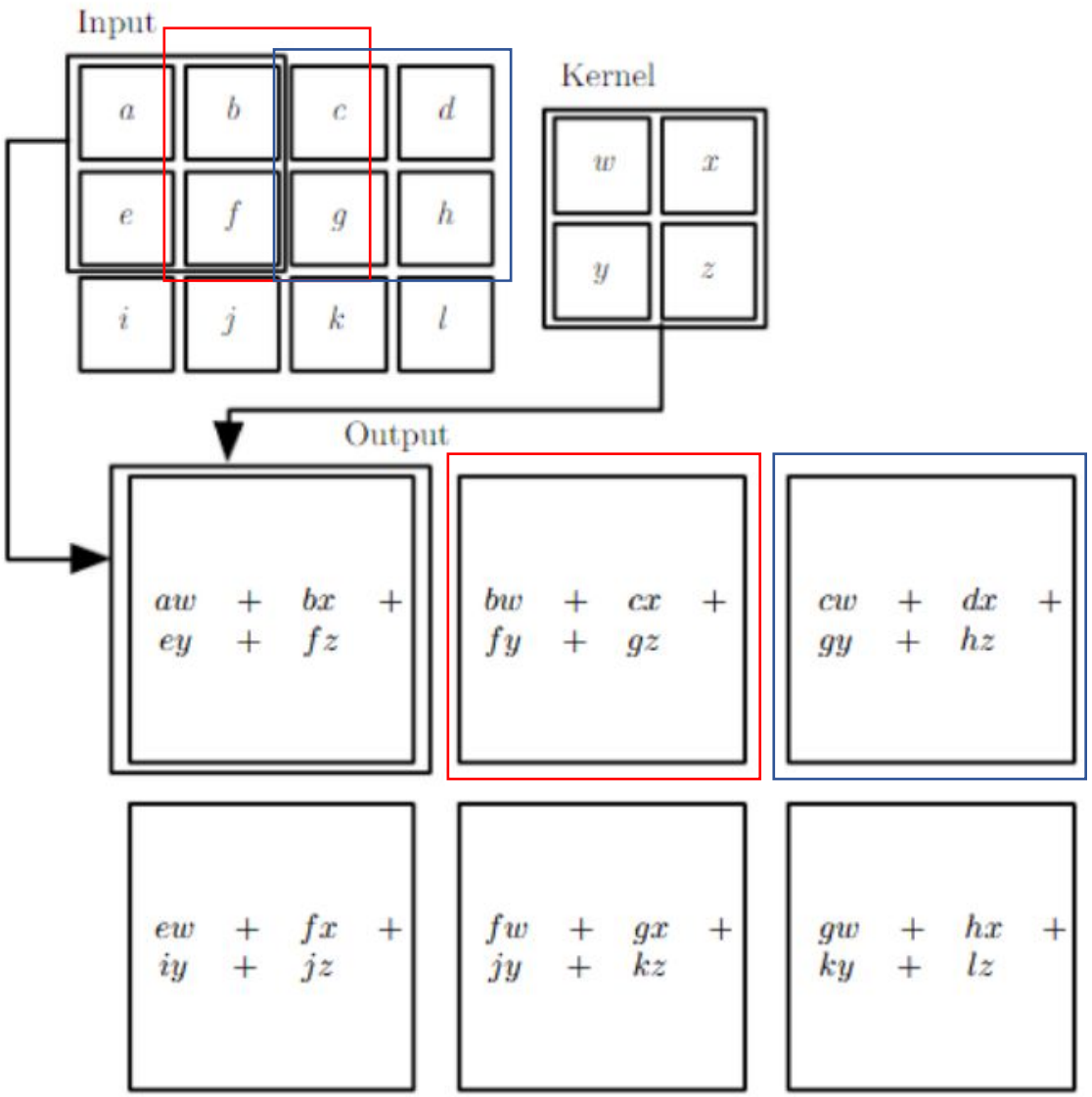
PROCESO:

Recorrer todas las regiones de la imagen (el filtro se desplaza sobre todos los píxeles de la imagen)

Calcular el valor resultante: la suma de todas las multiplicaciones de cada píxel por el coeficiente correspondiente del filtro [w, x, y ó z] que superpone a ese píxel.

M**apa de características**: el calculo de un nuevo valor se almacena en la siguiente posición de la matriz resultado

Filtro [2x2]




Goodfellow, I. , Bengio, Y. y Courville, A.: Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org

# Operación de convolución

*imágenes color*

La imagen original y su filtro cuentan con 3 planos de color [RGB]

Entrada: imagen de 3 canales, filtro 3 canales

Proceso: similar al descripto

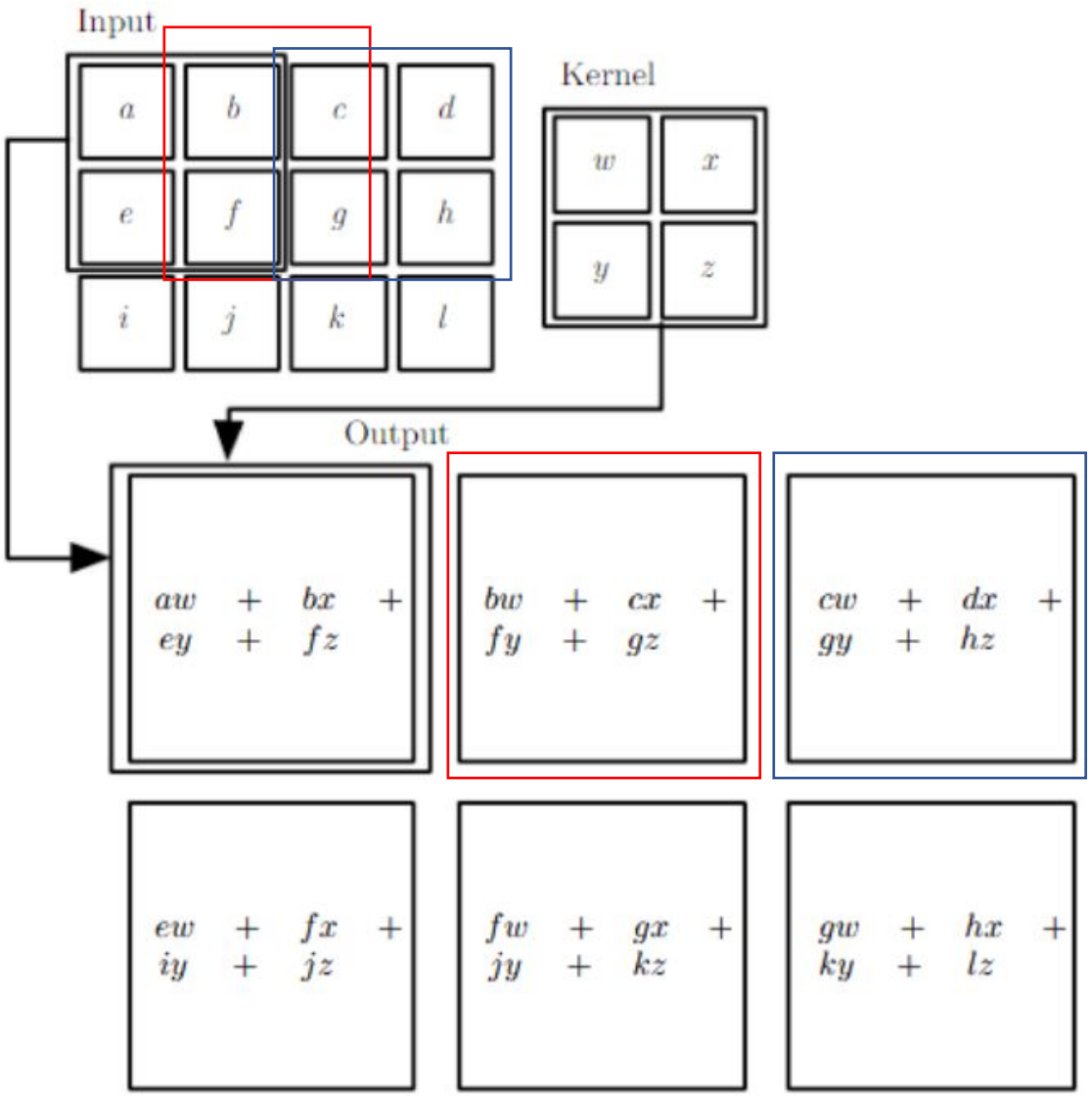
Salida: imagen reducida  a 1 plano.

**En el ejemplo:**

Imagen: 4 x 4 x 3,

Filtro: 2 x 2 x 3

Imagen resultante: 3 x 3




Goodfellow, I. , Bengio, Y. y Courville, A.: Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org

# Capa convolución

La convolución genera imagen de menor tamaño que la original

**Padding**, técnica que asegura que imagen “original” y “resultante” sean de igual tamaño agregando “0” en filas y columnas.

**Stride**: desplazamiento del kernel, hacia izq a der, y arriba abajo, en cada iteración.

stride = 1, convolución original, por defecto

stride > 1, la imagen resultante será menor,

Ej: stride = 2, salto de 2 en 2 en filas y columnas

[+] reduce cantidad de datos a procesar entre las capas

[+] combinar **Padding & Stride**

stride = 1,

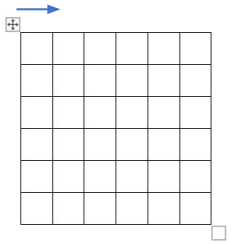
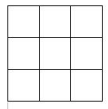


Imagen
6x6



Filtro

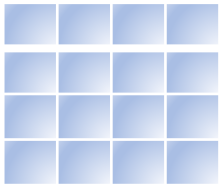
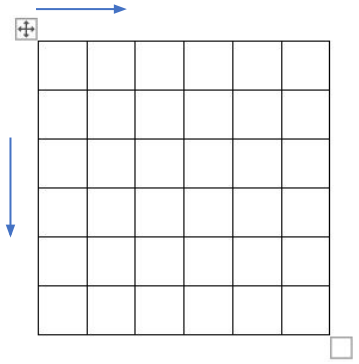


Imagen
4 x 4

stride = 2



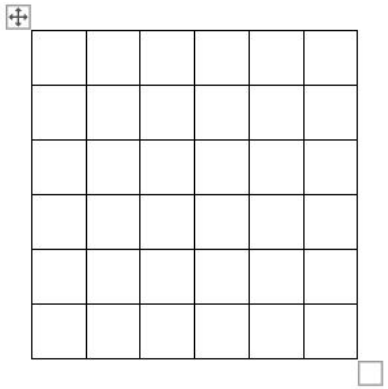
# Stacking

Cada filtro extrae una característica relevante

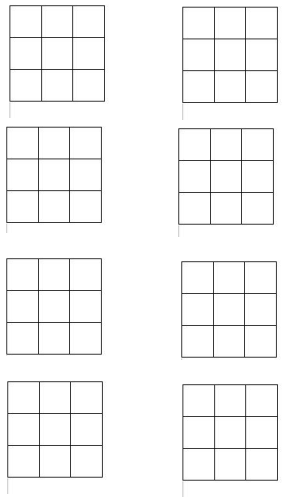
[+] disponer de numerosas características relevantes

Stacking: combinar / apilar las distintas imágenes  resultados de la convolución con múltiples filtros, 1 imagen por filtro

Si padding = 0 y strike = 1: ,

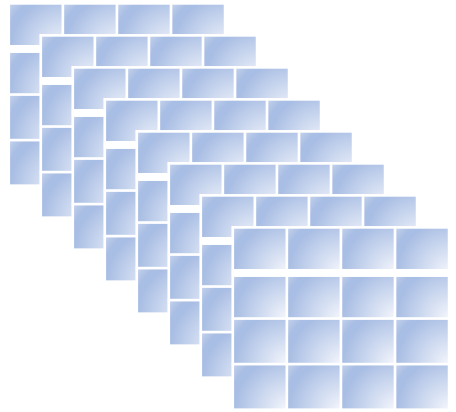


6 x 6



Se aplican 8 filtros, 3 x 3

Volumen: 8 imágenes de 4 x 4



La imagen es de 6 x 6, cada filtro es 3 x 3, resulta imágenes de 4 x 4

Se aplicaron 8 filtros, se obtiene un volumen  **4 x 4 x 8** [filas, columnas, profundidad/ filtros ]

# Capa de activación ReLU

La capa de activación ReLU (Rectified Linear Units):

Se utiliza para evitar la linealidad del modelo

La capa sustituye todos los valores negativos recibidos en la entrada por ceros.

# Capa Pooling - Agrupamiento

Paso para reducir la cantidad de neuronas antes de nueva convolución.

Analiza el contenido de imagen por regiones o bloques. Similar al uso de ***strides***, para reducir la cantidad de datos entre una capa y otra,

La capa de agrupación condensa el mapa de características adquirido en su información más esencial o representativa. El resultado o información, se alimenta a la red neuronal.

Se aplica en NN para reducir la varianza y la complejidad de la computación.

Reduce cantidad de datos, facilita procesamiento de imágenes, -> entrenamiento de la red

Métodos básicos:

- Max-pool: Agrupación máxima: selecciona el valor máximo de píxeles del lote (batch).
  - selecciona los píxeles más brillantes de la imagen. Valido si fondo de la imagen es oscuro e interés en píxeles más claros
  - Ej. datos MNIST, los dígitos representados en color blanco y fondo negro
- Min-pool: Agrupación mínima: selecciona el valor mínimo de píxel del lote.
  - selecciona los píxeles menos brillantes de la imagen. Valido si fondo de la imagen es claro e interés en los píxeles más oscuros
- Avg-pool: Agrupación promedio: selecciona el valor promedio de todos los píxeles del lote.
  - suaviza la imagen. Posibilidad que las características nítidas no se identifiquen

# Pooling - Agrupamiento

**Ej. Max-pool**: Agrupación máxima: selecciona el valor máximo de píxeles del lote (batch).

- selecciona los píxeles más brillantes de la imagen. Valido si fondo de la imagen es oscuro e interés en píxeles más claros
- Ej. datos MNIST, los dígitos representados en color blanco y fondo negro

| | | | | | |
|---|---|---|---|---|---|
| 4 | 2 | 2 | 1 | 4 | 6 |
| 3 | 1 | 5 | 7 | 8 | 8 |
| 9 | 12 | 14 | 6 | 9 | 11 |
| 4 | 3 | 4 | 11 | 4 | 6 |
| 4 | 7 | 15 | 2 | 8 | 13 |
| 3 | 2 | 10 | 1 | 5 | 8 |

| | |
|---|---|
| 1 | 1 |
| 1 | 1 |

Filtro, 2 x 2

| | | |
|---|---|---|
| **4** | **7** | **8** |
| **12** | **14** | **11** |
| **7** | **15** | **13** |

# Fase de clasificación

Las características extraídas deben definir a que clase o etiqueta corresponde, según el objetivo del proceso de entrenamiento.

Las neuronas funcionan como neuronas de un perceptrón multicapas,

$$Y_j = g\left( b_j + \sum_i w_{ij} + y_i \right)$$

$Y_j$ salida de una neurona j: se obtiene de la combinación lineal de las $y_i$ salidas de neuronas en la capa anterior cada una de ellas multiplicadas con un peso $W_{iji}$ correspondiente a esa conexión. Esta cantidad se suma a una influencia $b_j$

La función de activación g no-lineal – se aplica softMax

# CNN – Capas y operaciones: Fully Connected Layer

Capas al final de arquitectura CNN y conectan a todas las neuronas de salida (fully-connected).

Combina las características extraídas para realizar la clasificación o predicción final.

- Flatten: permite aplanar el volumen,
- Aprendizaje, basado en Backpropagation,
- Aplica *función SoftMax, calcula la distribución de probabilidad de un evento sobre "n" eventos*
- *Clasifica, utilizar la técnica on hot encoding. Se predice la clase según el mayor valor de probabilidad*

FC recibe vector en la entrada, y aplica sucesivamente una combinación lineal y una función de activación con la finalidad de clasificar la imagen

FC devuelve vector de tamaño correspondiente al número de clases en el que cada componente representa la probabilidad de la imagen pertenezca a una clase.

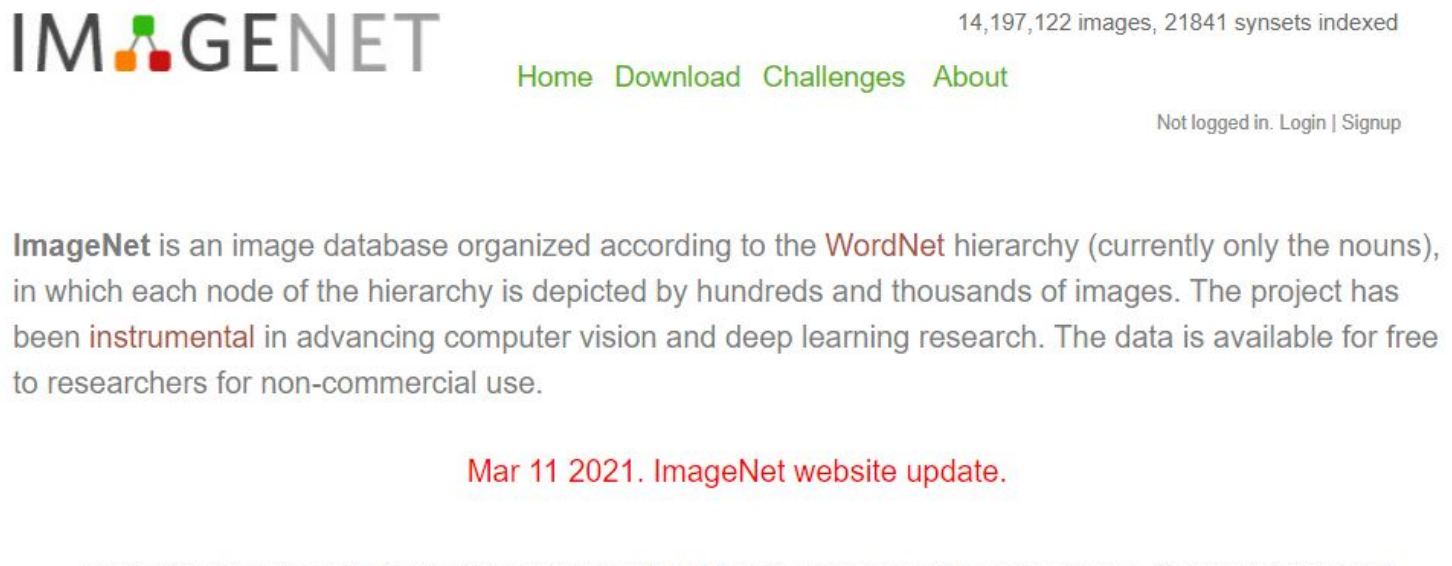
# Arquitecturas de una CNN

Proyecto **ImageNet.**

- base de datos visual diseñada para utilizar en el reconocimiento objetos.

**ImageNet Large Scale Visual Recognition Challenge (ILSVRC )**

- Competencia anual de software, detección correcta de objetos y escenas




ImageNet (image-net.org)

# Arquitecturas de CNN

- LeNet (1998)
- AlexNet (2012) – 8 capas
- VGG (2013), 16 capas
- GoogLeNet, Inception (2014) – 22 capas
- VGGNet (2014)  - 19 capas
- ResNet (2015) – 152 capas

**Tabla 1 - Arquitecturas de Redes Neuronales Convolucionales Profundas**

| Algoritmos | Nº de Capas | Índice de Error | Posición en Competencia ILSVRC | Año |
|---|---|---|---|---|
| AlexNet | 8 | 15.3 % | Primer Lugar | 2012 |
| VGGNet | 19 | 7.3 % | Segundo Lugar | 2014 |
| GoogleNet | 22 | 6.67 % | Primer Lugar | 2014 |
| ResNet | 152 | 3.57 % | Primer Lugar | 2015 |

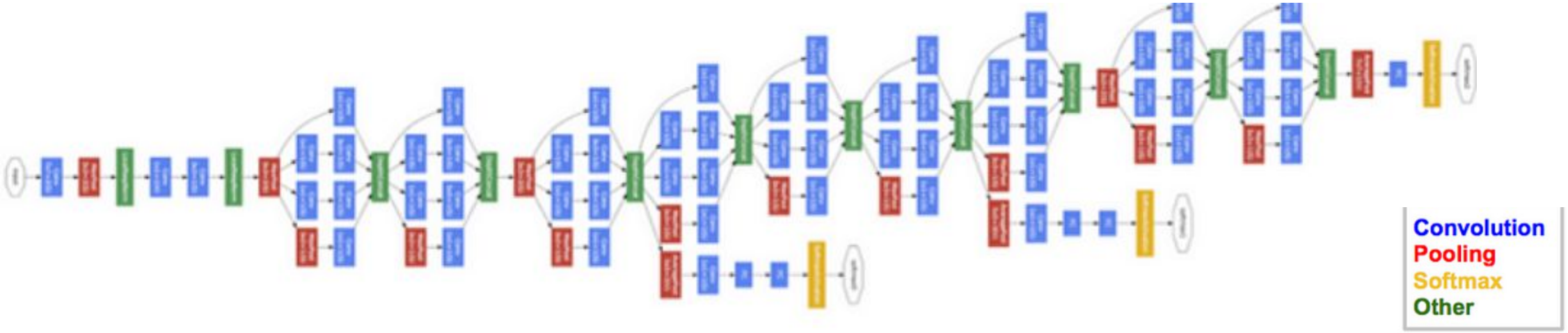
ImageNet (image-net.org)

Documento_completo.pdf-PDFA.pdf (unlp.edu.ar)

# Arquitecturas de una CNN

GoogLeNet o  Inception V1 (2014) - Google

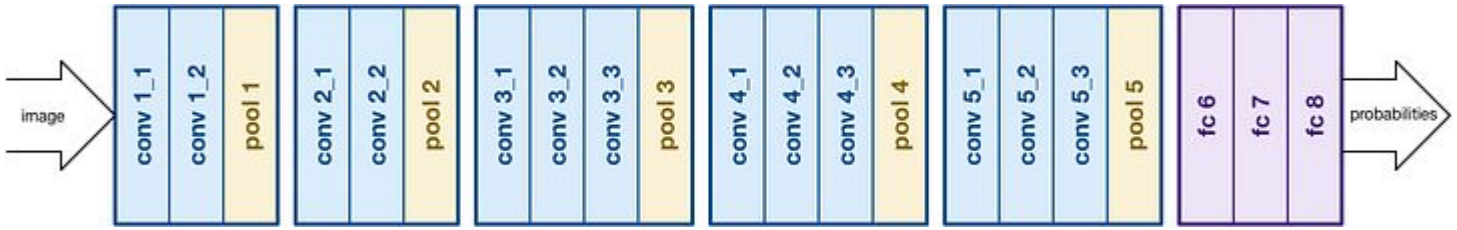
- Christian Szegedy, et al (Google), lograron los mejores resultados en deteccióon de objetos
- El modelo GoogLeNet, utilize el modulo y arquitectura inception
- Descripto en “Going Deeper with Convolutions.” https://arxiv.org/abs/1409.4842, https://arxiv.org/pdf/1409.4842.pdf
- La RN utilizó una CNN inspirada en LeNet, e implementó como elemento novedoso un módulo de inicio.
- Usó normalización por lotes, distorsiones de imagen y RMSprop. El módulo se basa en varias convoluciones muy pequeñas para reducir el número de parámetros.
- Arquitectura: 22 capas de profundidad, redujo el número de parámetros de 60 millones (AlexNet) a 4 millones.
- Mejoró tasa de error



# Arquitecturas de una CNN

VGGNet (2014) - Simonyan y Zisserman.

- Very Deep Convolutional Networks (VGGNet)
- VGGNet-16 o VGGNet-19, arquitecturas
  - VGG-16: consta de 13 capas convulcionales y 3 capas completamente conectadas [colab.research.google.com/drive/14rEK6b2lAjcvO1Px1NoqLr8mWqgVbPb2?hl=es]
- Similar a AlexNet,
- [+] Configuración de peso de VGGNet disponible públicamente y se ha utilizado en muchas otras aplicaciones y desafíos como un extractor de características de referencia.
- [-] Consta de 138 millones de parámetros, pueden ser difíciles de manejar.



# Arquitecturas de una CNN

Red Neural Residual, ResNet (2015)

- Kaiming et al. (2015)
- Inspiración biológica que algunas neuronas se conectan con neuronas en capas no necesariamente contiguas, saltando capas intermedias. Arquitectura de nivel con "conexiones de salto" y fuerte normalización de lotes.
    - Las conexiones de salto o unidades cerradas o unidades recurrentes cerradas, tienen una gran similitud con los elementos aplicados en las RNN.
    - Las conexiones del modelo facilita la propagación de la información entre las distintas capas de la red.
    - Una capa de red NN obtiene el mapa de características producido por la capa anterior y también el mapa de la anterior a esta última.
- Técnica que entrenó NN con 152 capas y controló el problema de Desvanecimiento de Gradiente (Vanishing Gradient). Alternativas: ResNet-50, ResNet-100, ResNet-152

# Funciones de perdida

Mide la distancia entre el valor real de las observaciones y el estimado por la RN, se aplica para optimizar los parámetros de la NN.

Definiciones

- Función de pérdida (Loss): se define para un ejemplo concreto.
- Función de coste (Cost): se define para una colección de ejemplos

Algunas más utilizadas

- Mean Squared Error (MSE)
- Binary Cross Entropy
- Mean Absolute Error (MAE)

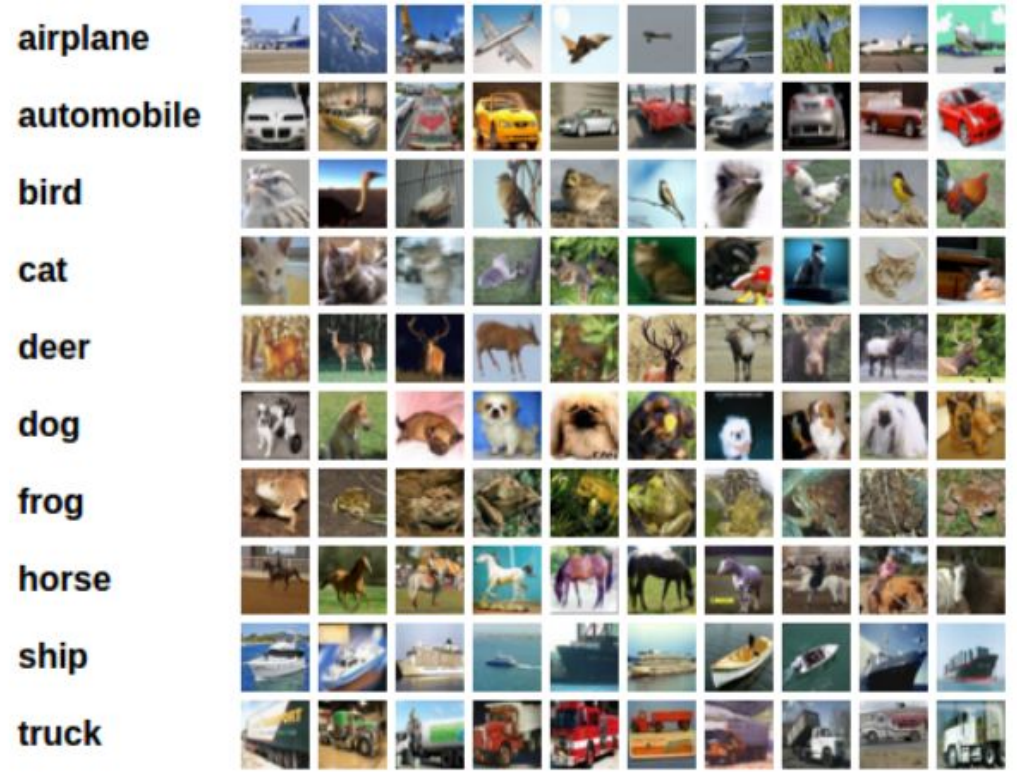
# Optimizadores

- SGC (Descenso de gradiente estocástico / ***Stochastic gradient descent***). Método iterativo para optimizar función objetivo diferenciable o subdiferenciable. Aproximación del DG, dado que sustituye el gradiente real [calculo sobre todo el conjunto de datos] por estimación obtenida a partir de un subconjunto de datos seleccionados aleatoriamente.
- Adam (Adaptive Moment Estimation): Combina las ventajas del algoritmo AdaGrad y RMSProp. Ampliamente utilizado, buena elección por defecto.
- RMSProp (Root Mean Square Propagation): Adapta la tasa de aprendizaje para cada parámetro individualmente según la magnitud de sus gradientes recientes.
- Adagrad (Adaptive Gradient): Adapta la tasa de aprendizaje de forma individual para cada parámetro en función de las sumas acumulativas de los gradientes cuadráticos anteriores.
- Adadelta: Mejora el método Adagrad al resolver el problema de la reducción de la tasa de aprendizaje de forma exponencial.
- AdamW: Es similar a Adam, pero con una corrección para el sesgo de los momentos de primer y segundo orden.
- Adamax: Variante de Adam que utiliza el infinito norma en lugar de la norma 2 de los gradientes.

# Pipeline CNN

Training an image classifier

1. Load and normalize the training and test datasets using torchvision
2. Preprocesing data
3. Define a Convolutional Neural Network
4. Define a loss function
5. Train the network on the training data
6. Test the network on the test data
7. Evaluate model




cifar10

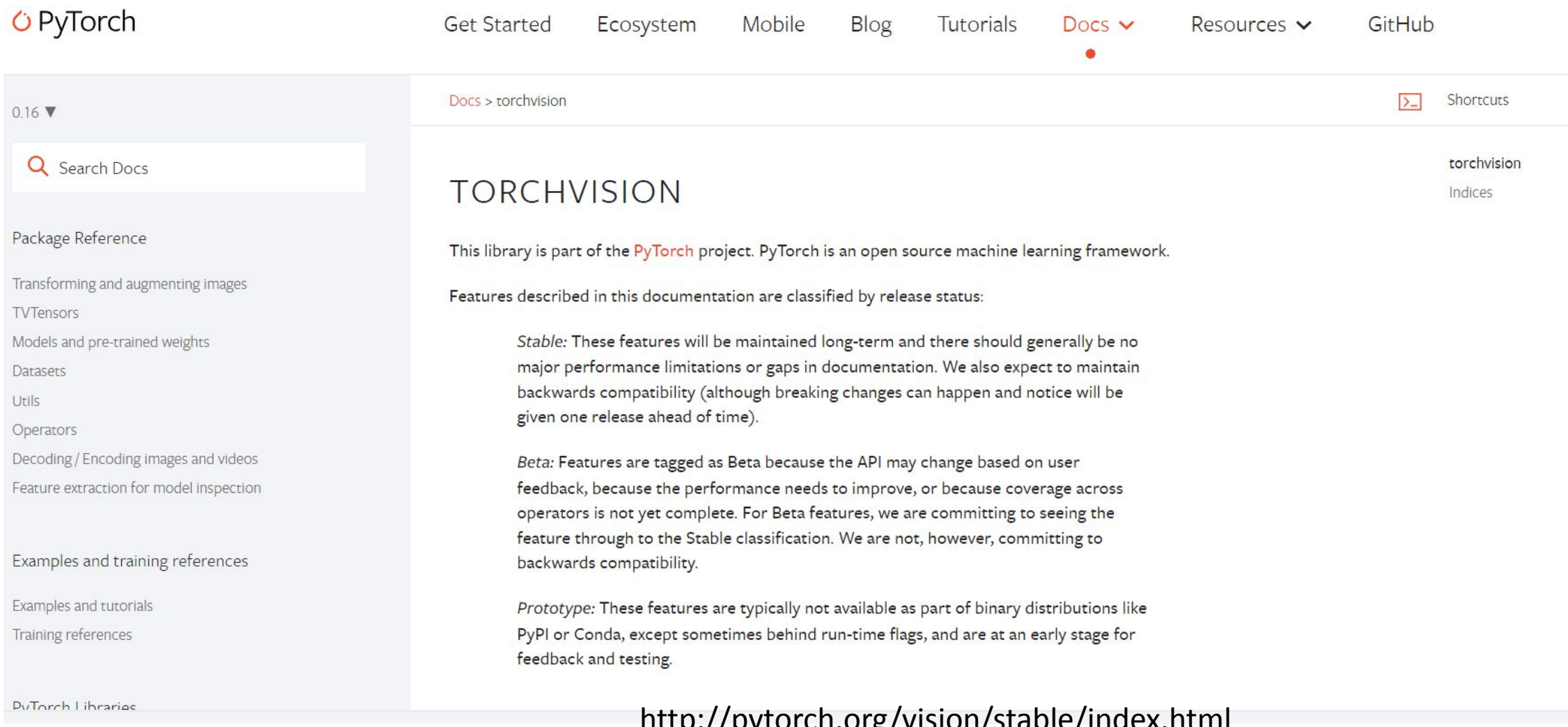
# Preprocesar los datos

Normalizar los datos

Aumento de datos

- Incrementa la cantidad de imágenes en un conjunto de datos determinado. Algunas técnicas son:
- Crooping
  - Rotation
  - Flipping
  - Noise injection
  - Color space transformation

# PyTorch




http://pytorch.org/vision/stable/index.html

# DenseNet

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has L(L+1) 2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet

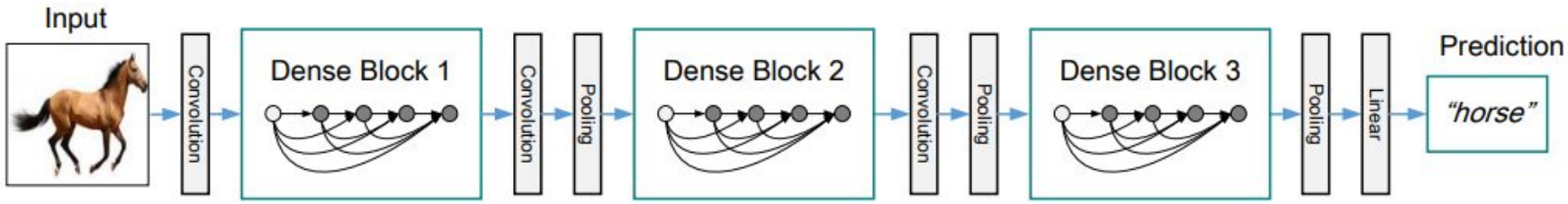
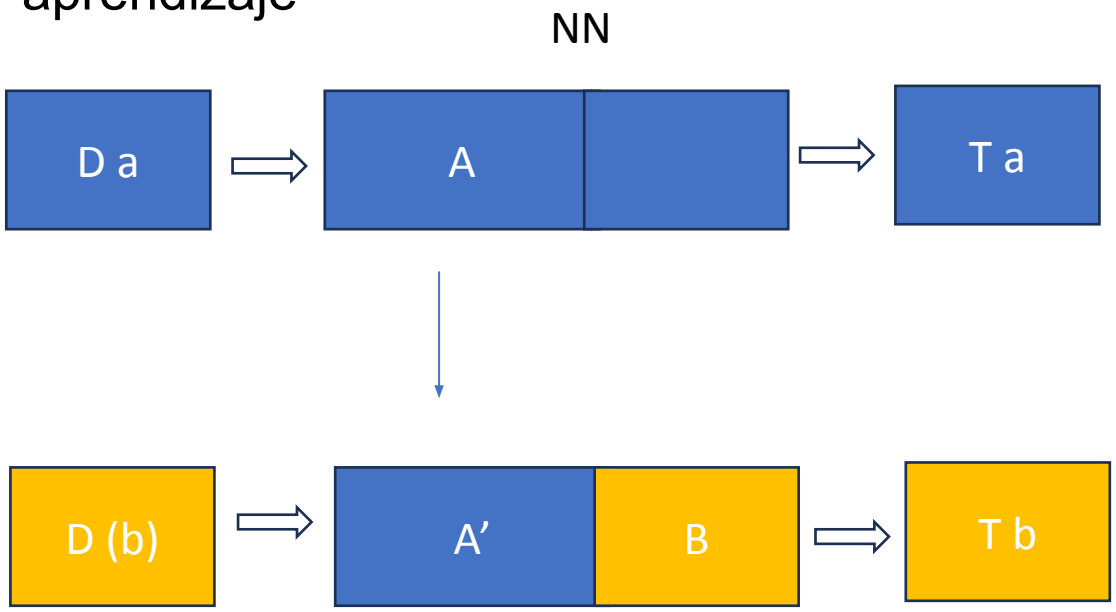



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

1608.06993.pdf (arxiv.org)

# Transfer Learning

- Utilizar un entrenamiento previo –realizado sobre un dataset genérico-
- Entrenamiento de bordes, colores, formas, y capitalizar lo aprendido e incorporar nuevo aprendizaje




Transformaciones,

- igual tamaño [resize]
- Normalizar valores
- Indicar el numero de canales en primer termino [permute]

import albumentations as A,

- Tamaño
- Color
- Brillo

# Fine Tunning

- Comparar el entrenamiento de un modelo aplicando TRUE / FALSE en pretrained
- Entrenamiento desde 0
- Uso pesos descargados
    - Se utilizan pesos entrenados. Es decir, la red “sabe ver” [utiliza conocimiento aprendido]

```
model = Model (pretrained = True, freeze =false)

fit(modelo, dataloader)
```

# Fine Tunning

- Comparar el entrenamiento de un modelo aplicando TRUE / FALSE en pretrained
- Entrenamiento desde 0
- Uso pesos descargados
  - Se utilizan pesos entrenados, por ello la red “sabe ver” [utiliza conocimiento aprendido]
- modificar learning rate [lr]

```
model = Model (pretrained = True, freeze = True)
fit(modelo, dataloader)
model.unfreeze()
fit(modelo, dataloader, lr = 1e-4)
```

# Fine Tunning

Diseñar un optimizador con lista de parámetros, modificando el ***lr***

```
optimizer = torch,optim.Adam([
        {‘params’ : model.resnet.parameters(), ‘lr’: 1e-4}
        {‘params’ : model.fc.parameters(), ‘lr’: 1e-3}
])
```

# Comparar experimentos

Comparar modelos combinando:

- Transfer Learning [T / F]
- Fine Tunning - Pesos descargados [ T / F]