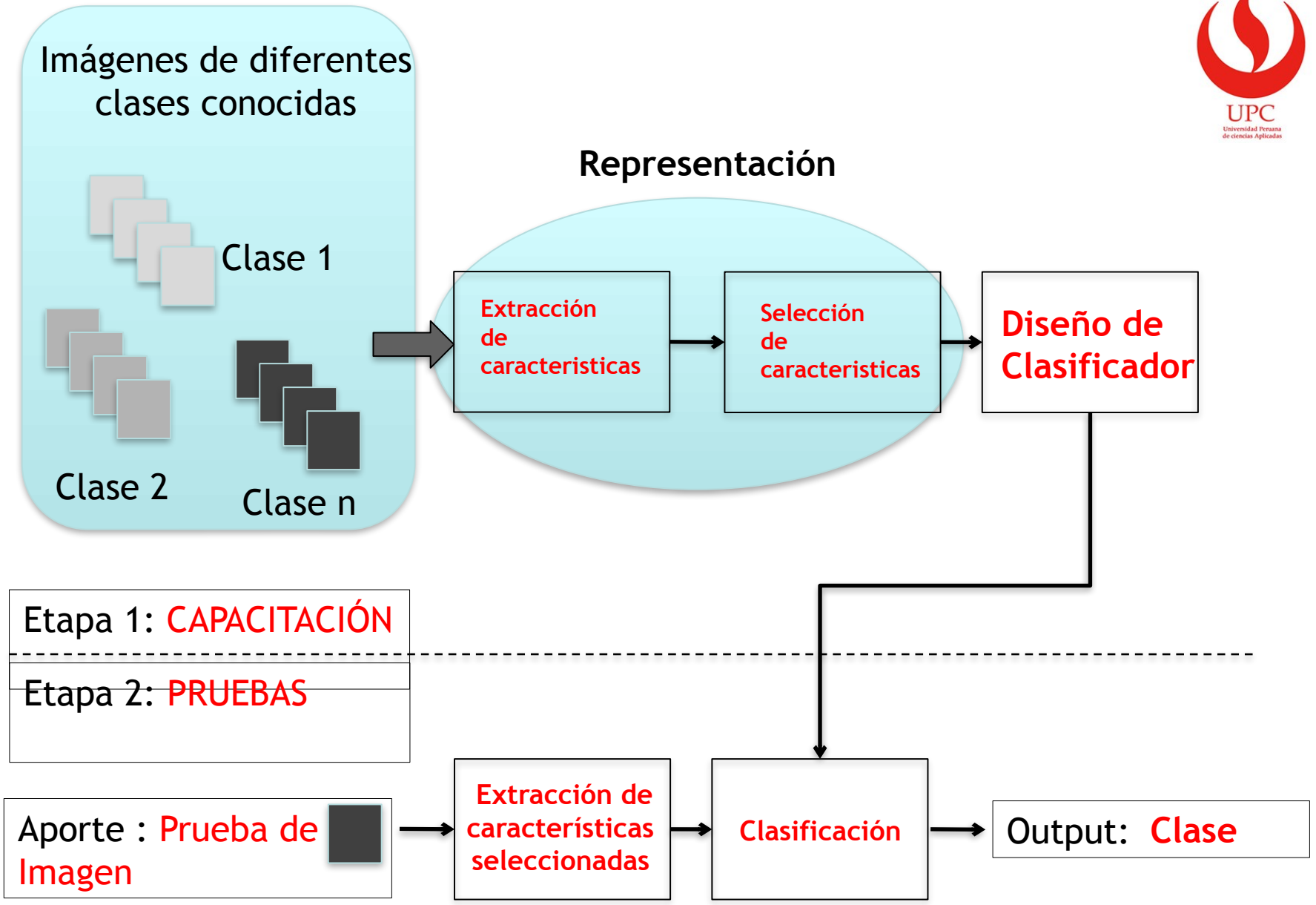
Visión por Computador

CNN

PhD. Carlos Fernando Montoya Cubas

Motivación


Imágenes de diferentes clases conocidas
Clase 1
Clase 2
Clase n
Representación
Extracción de caracteristicas
Selección de caracteristicas
Diseño de Clasificador
Etapa 1: CAPACITACIÓN
Etapa 2: PRUEBAS
Aporte : Prueba de Imagen
Extracción de características seleccionadas
Clasificación
Output: Clase

MOTIVACIÓN

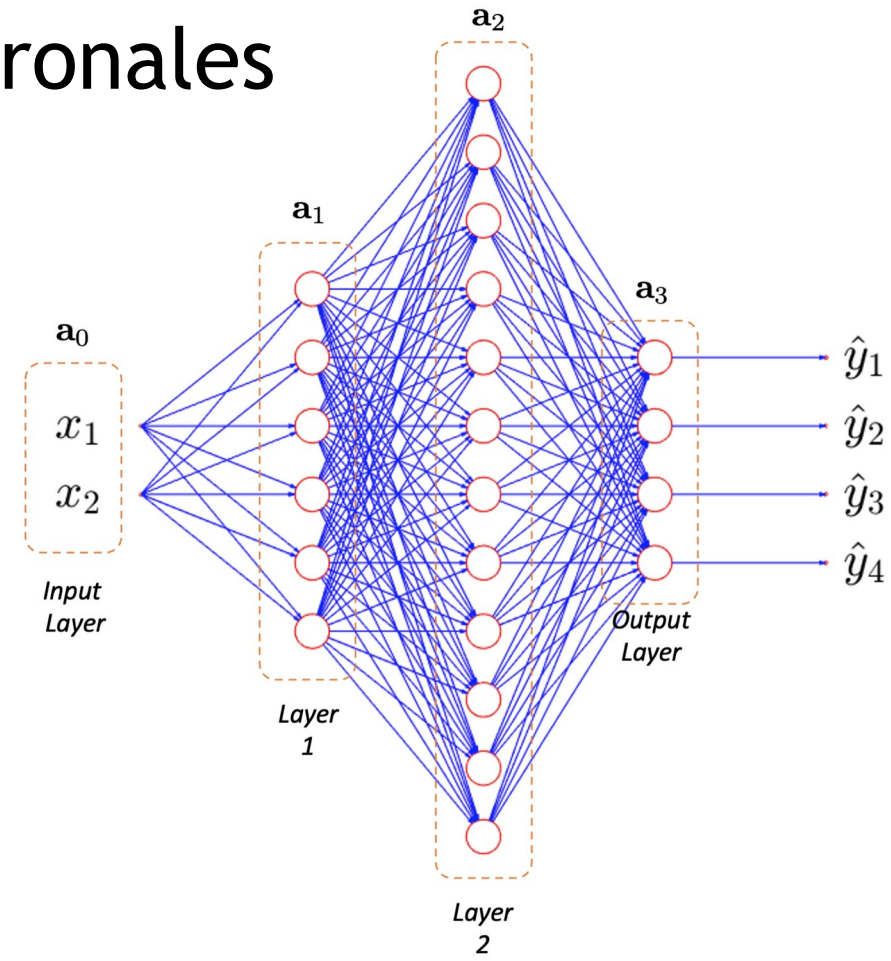
- Características manuales vs. aprendidas
- LBP, Haralick, Fourier, Gabor, etc

or

- Representación aprendida a partir de datos de entrenamiento

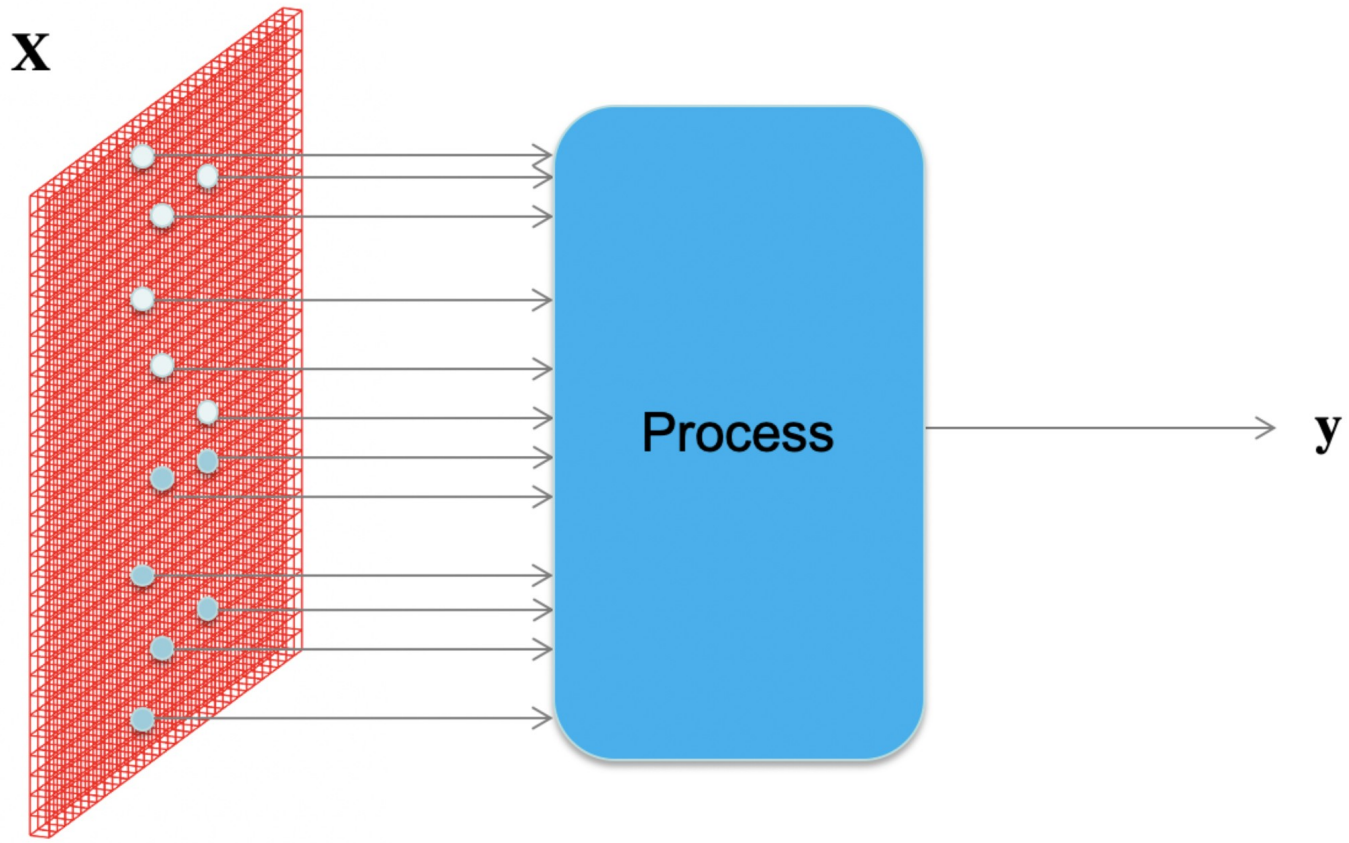
MOTIVACIÓN

- Redes neuronales



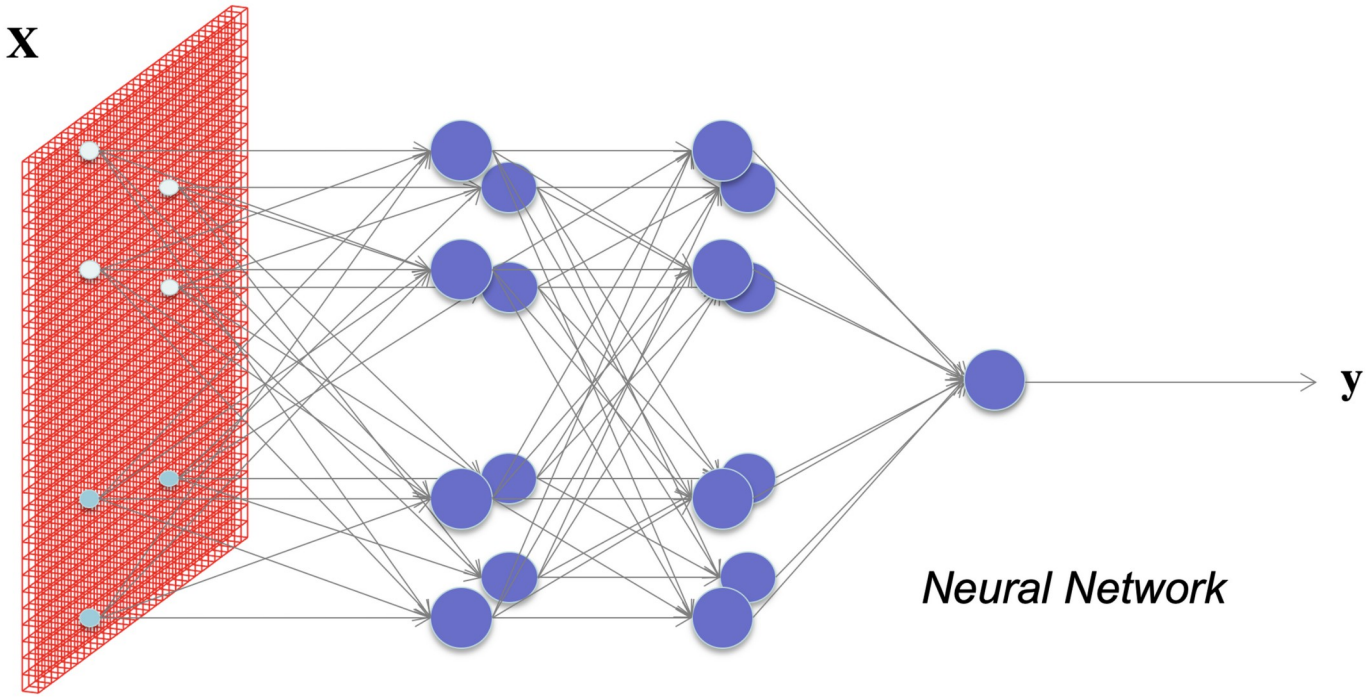
MOTIVACIÓN

- ¿Cómo procesar una imagen con redes neuronales?

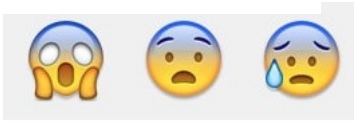


MOTIVACIÓN

- ¿Qué sucede si cada píxel es una entrada?



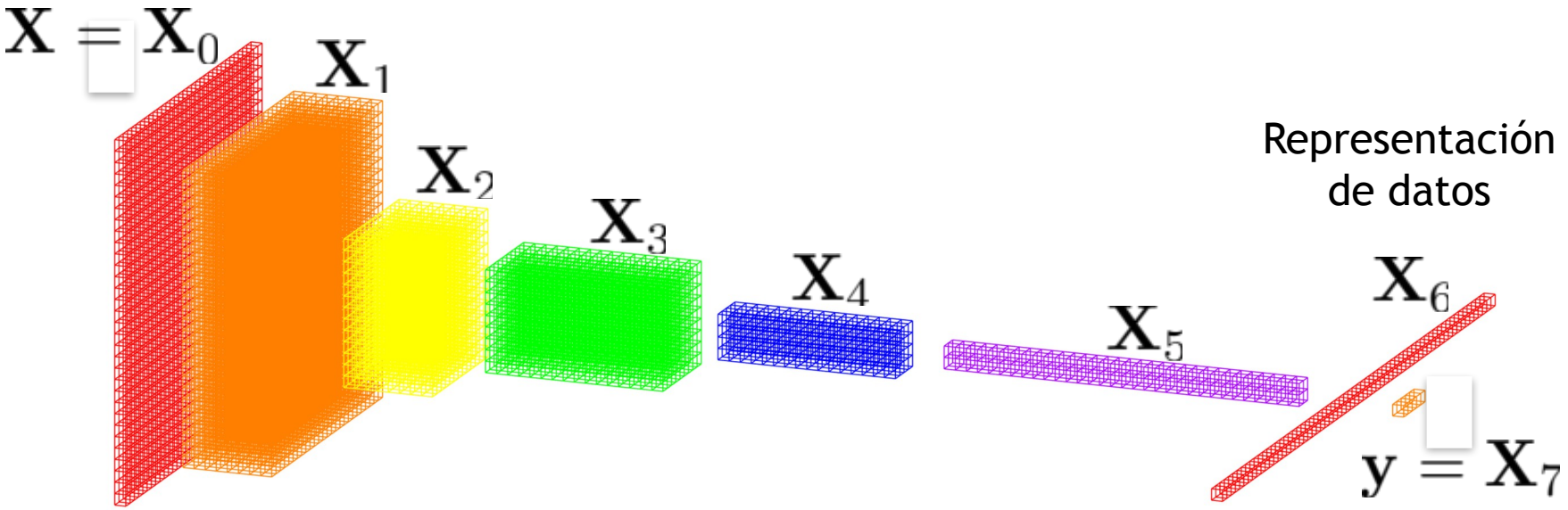
¿Cuántas neuronas y pesos?



Básicos

DEEP LEARNING BASICS (CNN)

Redes neuronales convolucionales



UPC
Universidad Peruana de ciencias Aplicadas

DEEP LEARNING BASICS (CNN)

Redes neuronales convolucionales

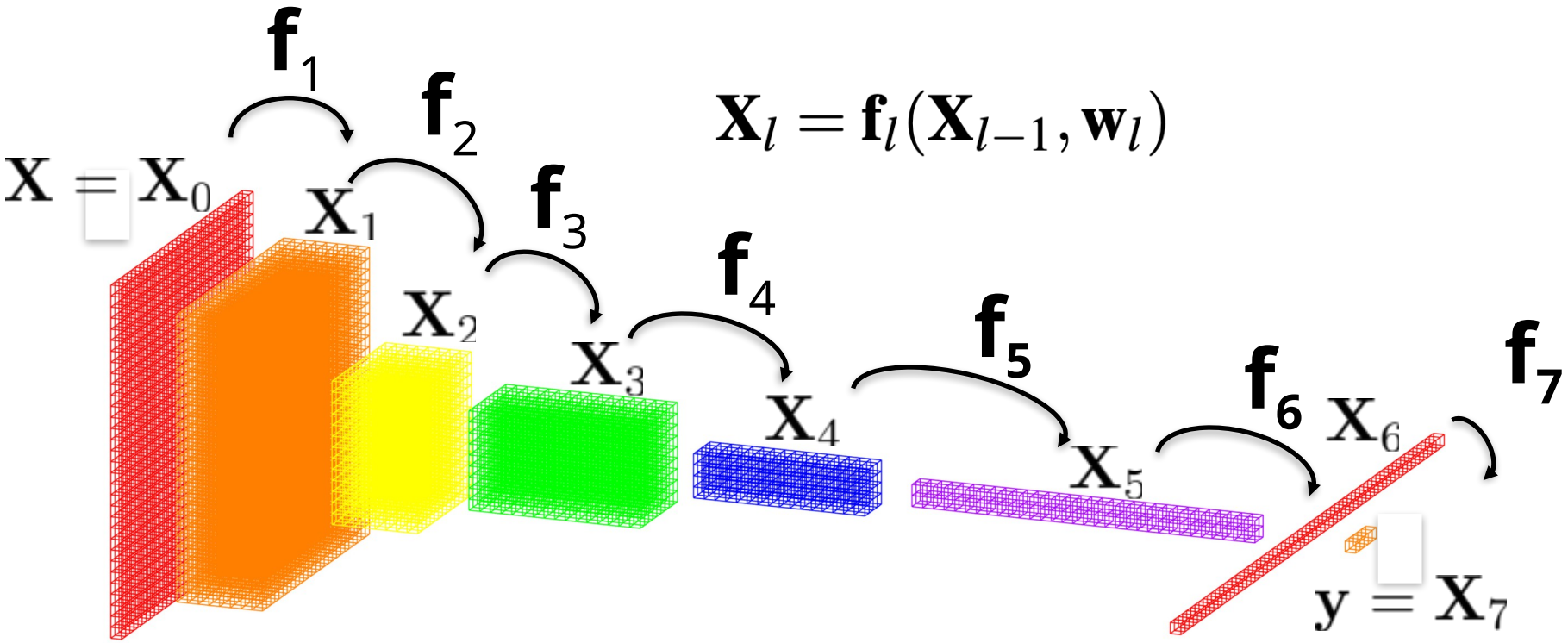


Imagen de entrada

Capas

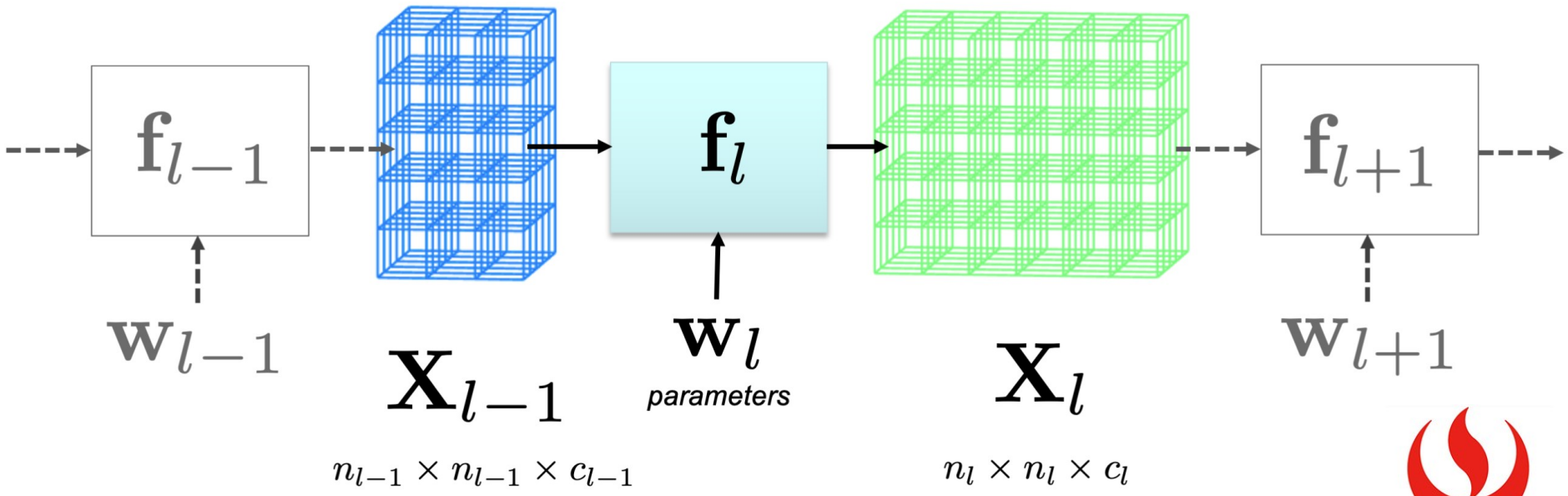
Clasificación

UPC
Universidad Peruana de ciencias Aplicadas

DEEP LEARNING BASICS

La red neuronal convolucional, puede entenderse como un conjunto de capas, en el que cada capa procesa una imagen de entrada **X** para obtener una salida imagen **Y**:

$$\mathbf{X}_l = \mathbf{f}_l(\mathbf{X}_{l-1}, \mathbf{w}_l)$$



DEEP LEARNING: Capas típicas

- Capa de convolución
- Capa de agrupación
- Unidad Lineal Rectificada
- Totalmente conectado y SoftMax

DEEP LEARNING: Capas típicas

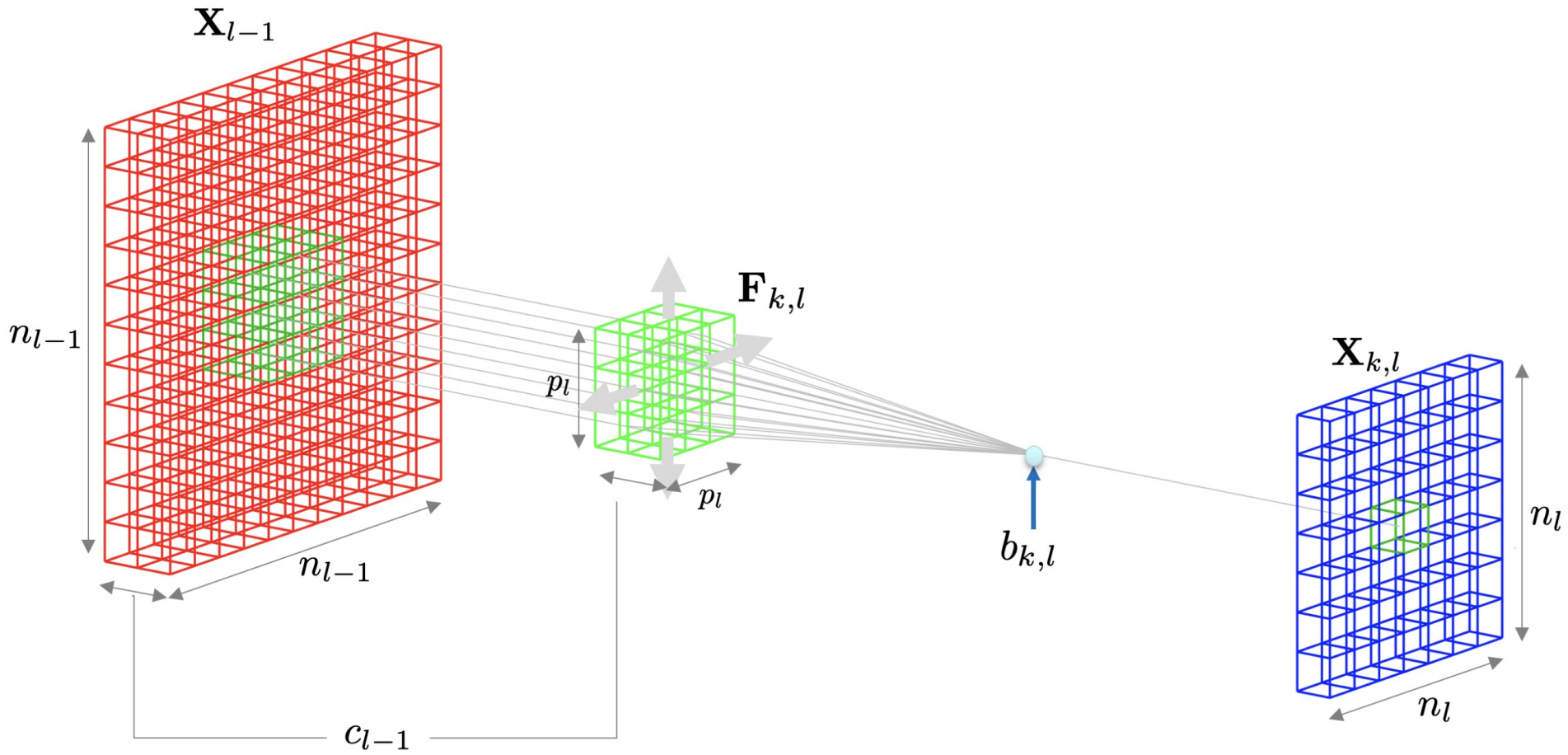
- Capa de convolución
- Capa de pooling
- Unidad Lineal Rectificada
- Totalmente conectado y SoftMax

DEEP LEARNING: Capa de convolución

$$\mathbf{X}_{k,l} = \mathbf{X}_{l-1} * \mathbf{F}_{k,l} + b_{k,l} \quad \text{for } k = 1 \ldots m_l$$

$$X_{k,l}(i,j) = b_{k,l} + \sum_{u=1}^{p_l} \sum_{v=1}^{p_l} \sum_{w=1}^{q_l} X_{w,l-1}(i+u, j+v) F_{k,l}(u,v,w)$$

DEEP LEARNING: Capa de convolución



UPC
Universidad Peruana de ciencias Aplicadas

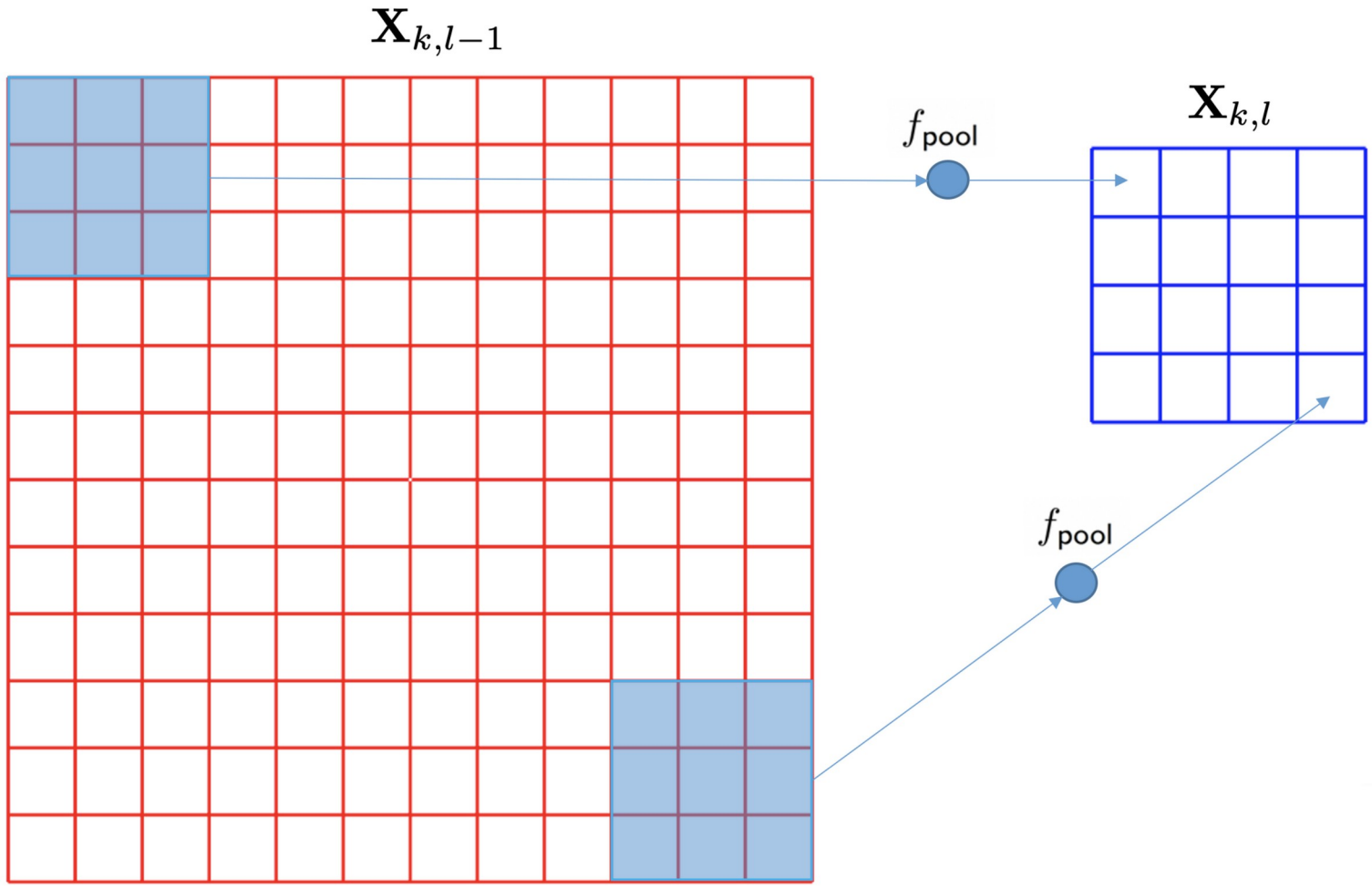
DEEP LEARNING: Capas típicas

- Capa de convolución
- Capa de agrupación
- Unidad Lineal Rectificada
- Totalmente conectado y SoftMax

DEEP LEARNING: Capa de pooling

$$X_{k,l}(i,j) = f_{\text{pool}}\{X_{k,l-1}(u,v) : (u,v) \in \Omega(i,j)\}$$

DEEP LEARNING: Capa de pooling



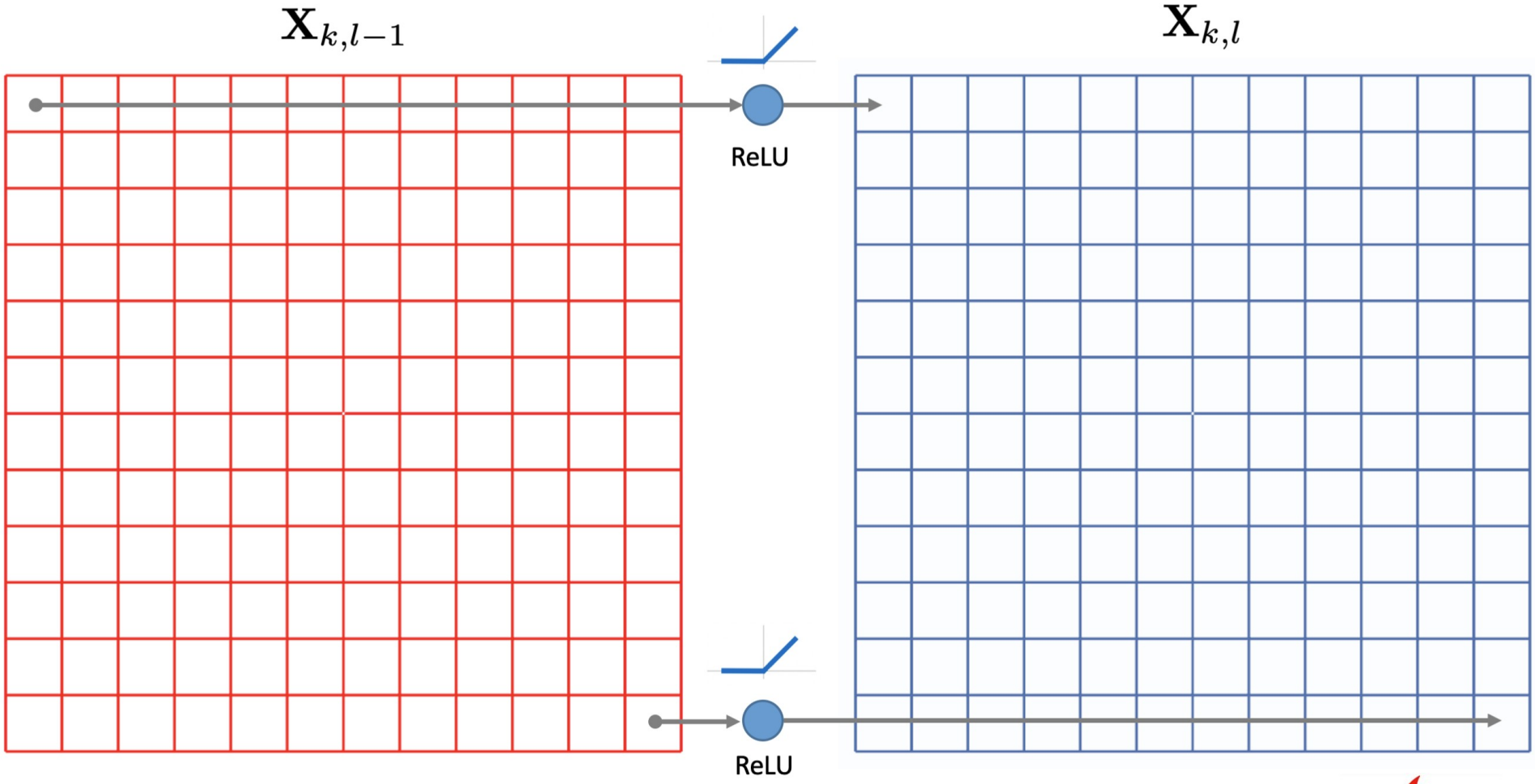
DEEP LEARNING: Capas típicas

- Capa de convolución
- Capa de pooling
- Unidad Lineal Rectificada
- Totalmente conectado y SoftMax

DEEP LEARNING: Unidad Lineal Rectificada(ReLU)

$$X_{k,l}(i,j) = \max\{0, X_{k,l-1}(i,j)\}$$

DEEP LEARNING: Unidad Lineal Rectificada (ReLU)



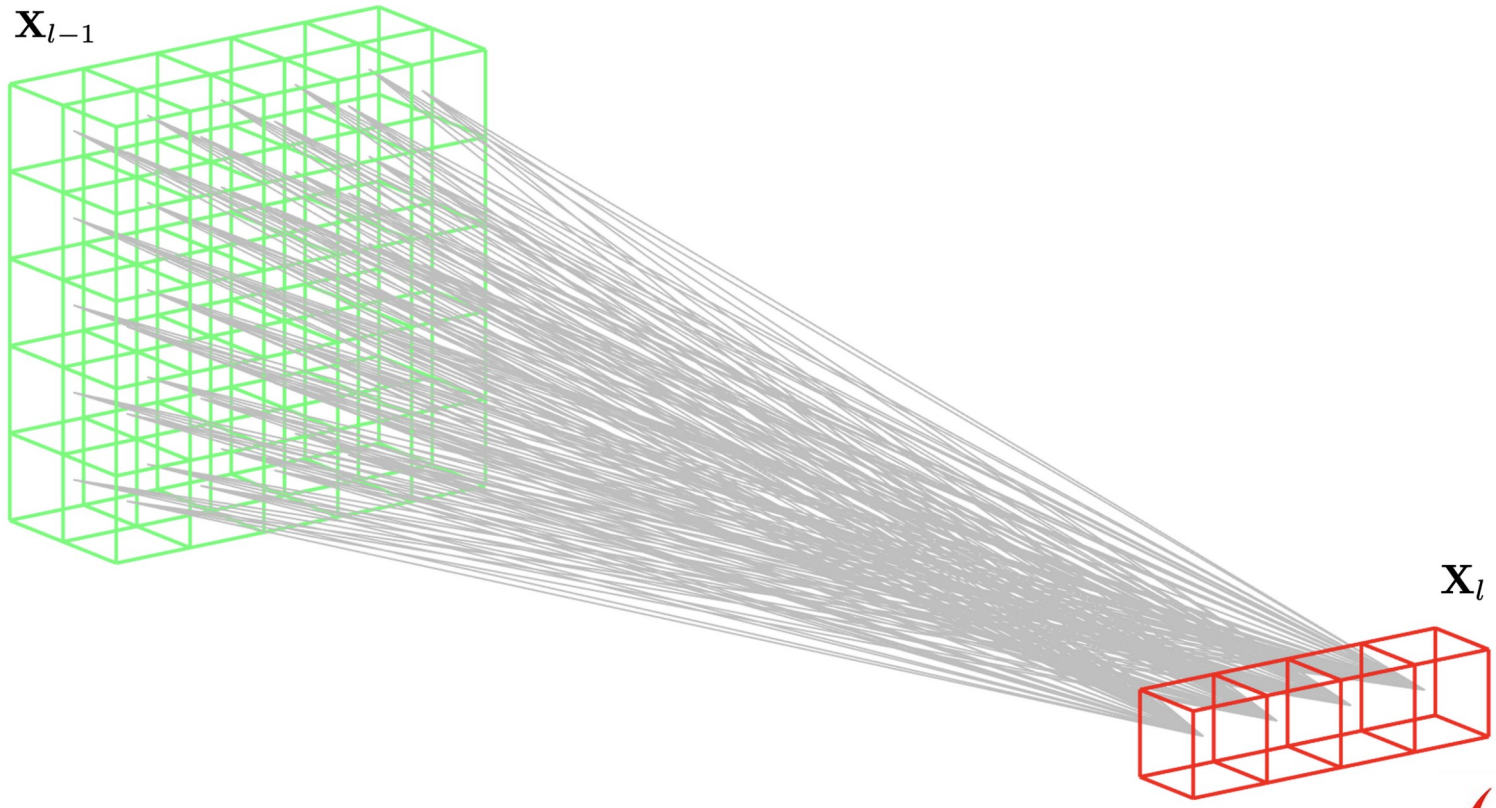
PhD. Carlos Fernando Montoya Cubas

UPC
Universidad Peruana de ciencias Aplicadas

DEEP LEARNING: Capas típicas

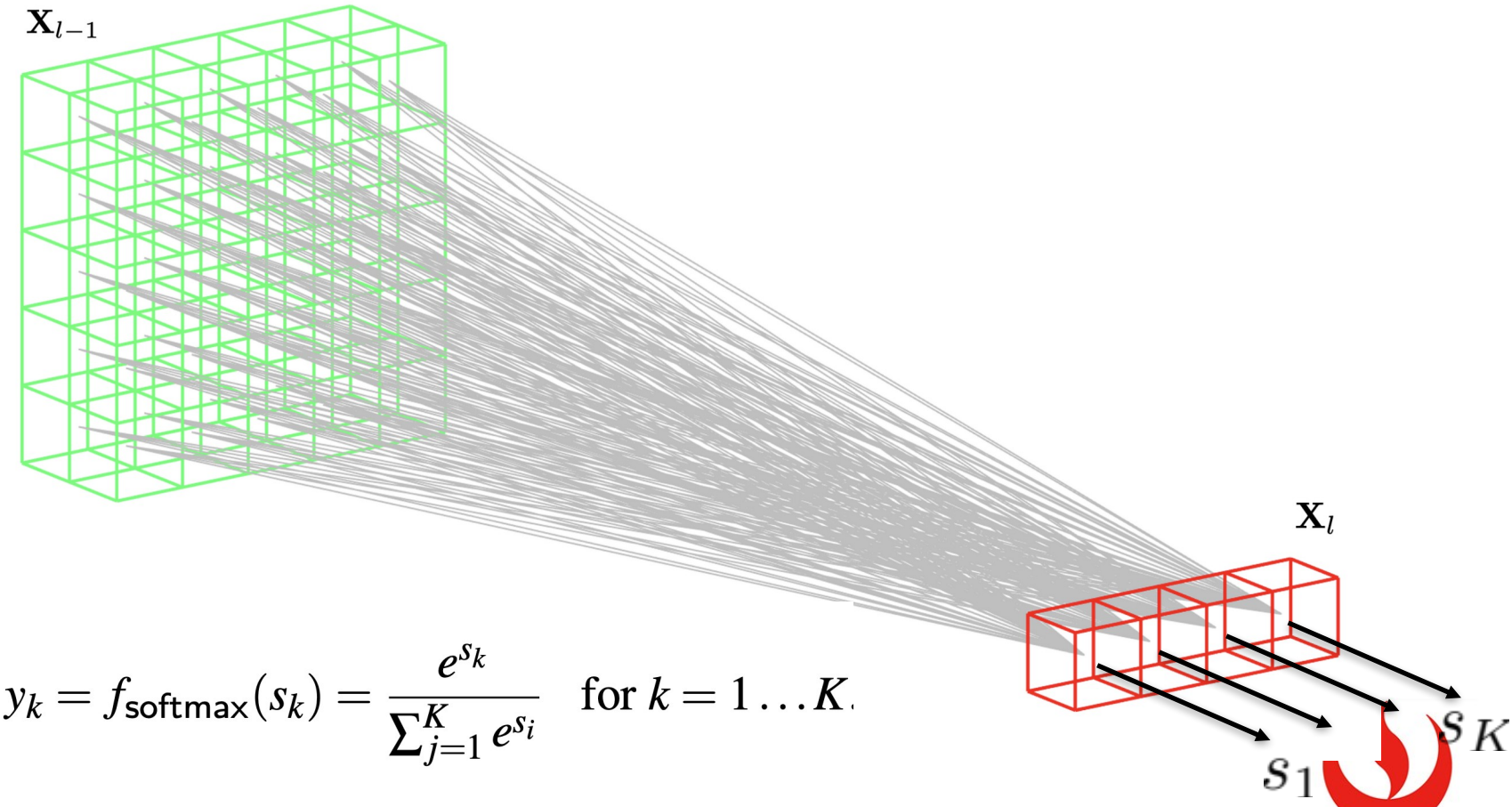
- Capa de convolución
- Capa de pooling
- Unidad Lineal Rectificada
- Totalmente conectado y SoftMax

DEEP LEARNING: Completamente conectado



UPC
Universidad Peruana de ciencias Aplicadas

DEEP LEARNING: Completamente conectado + SoftMax



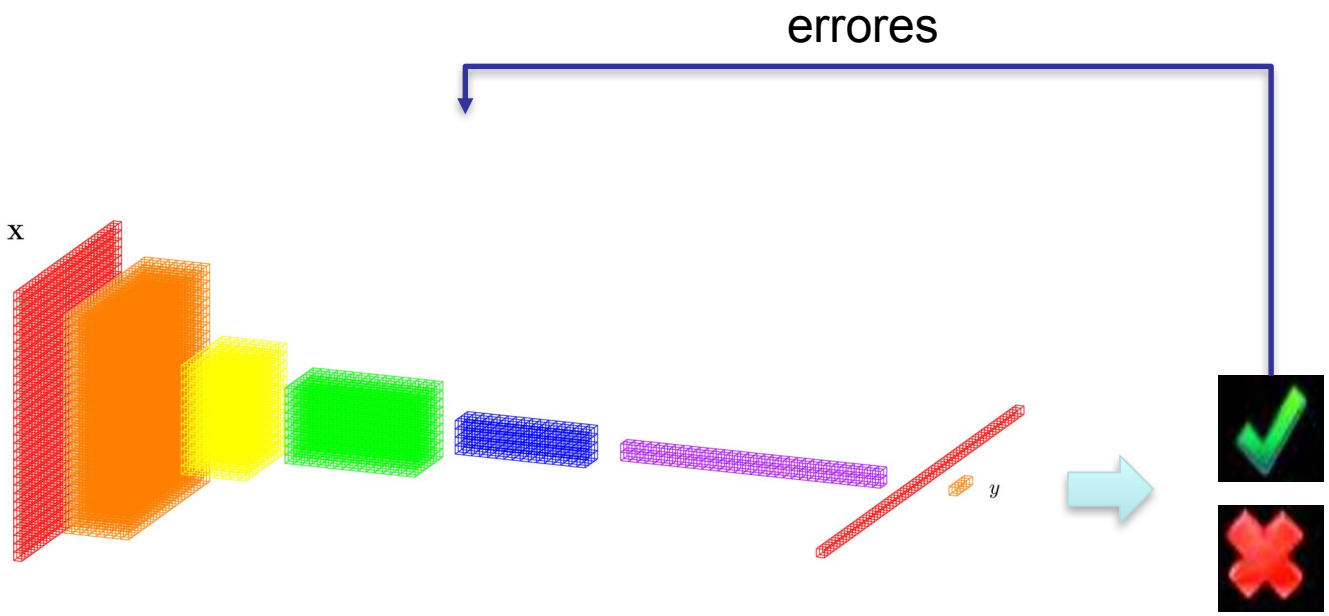
$$y_k = f_{\text{softmax}}(s_k) = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_i}} \quad \text{for } k = 1 \ldots K.$$

Training

DEEP LEARNING: Training

perros

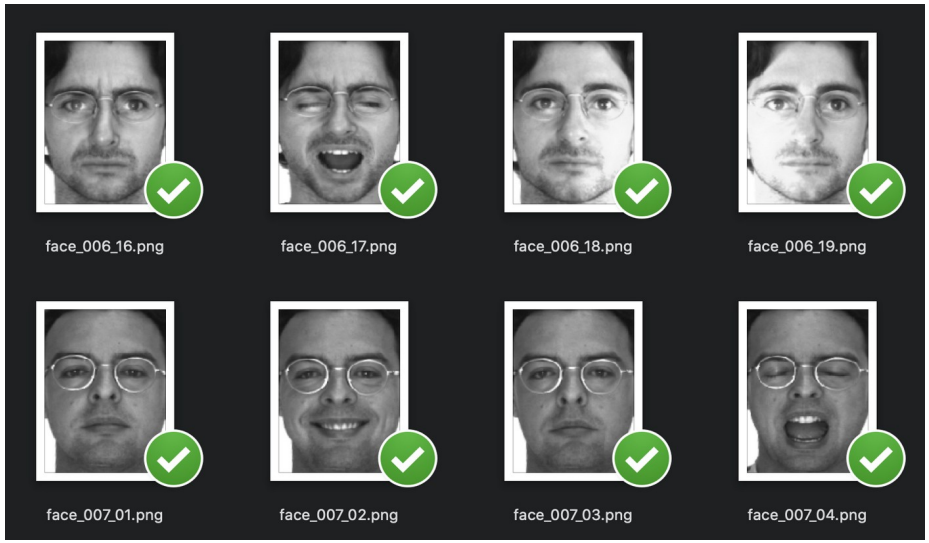
gatos

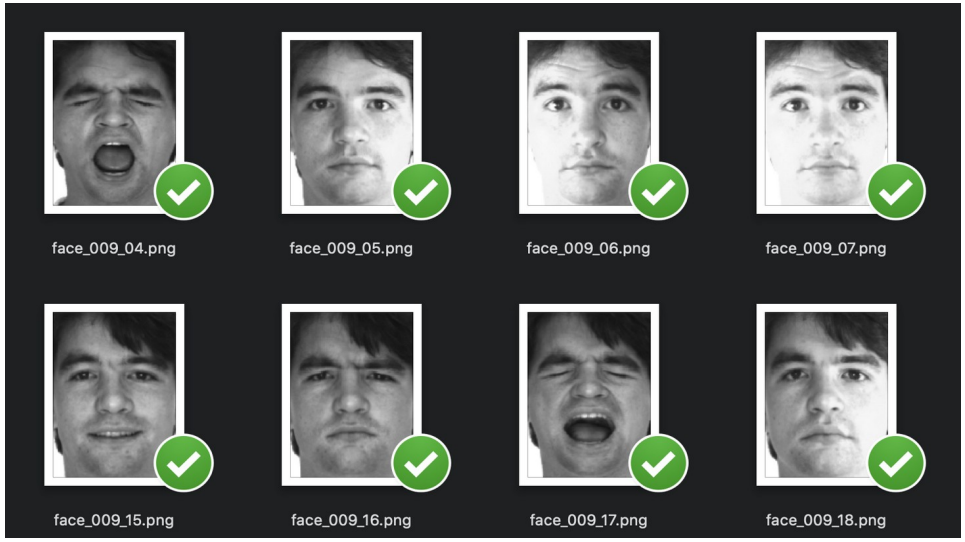


Retro-propagación

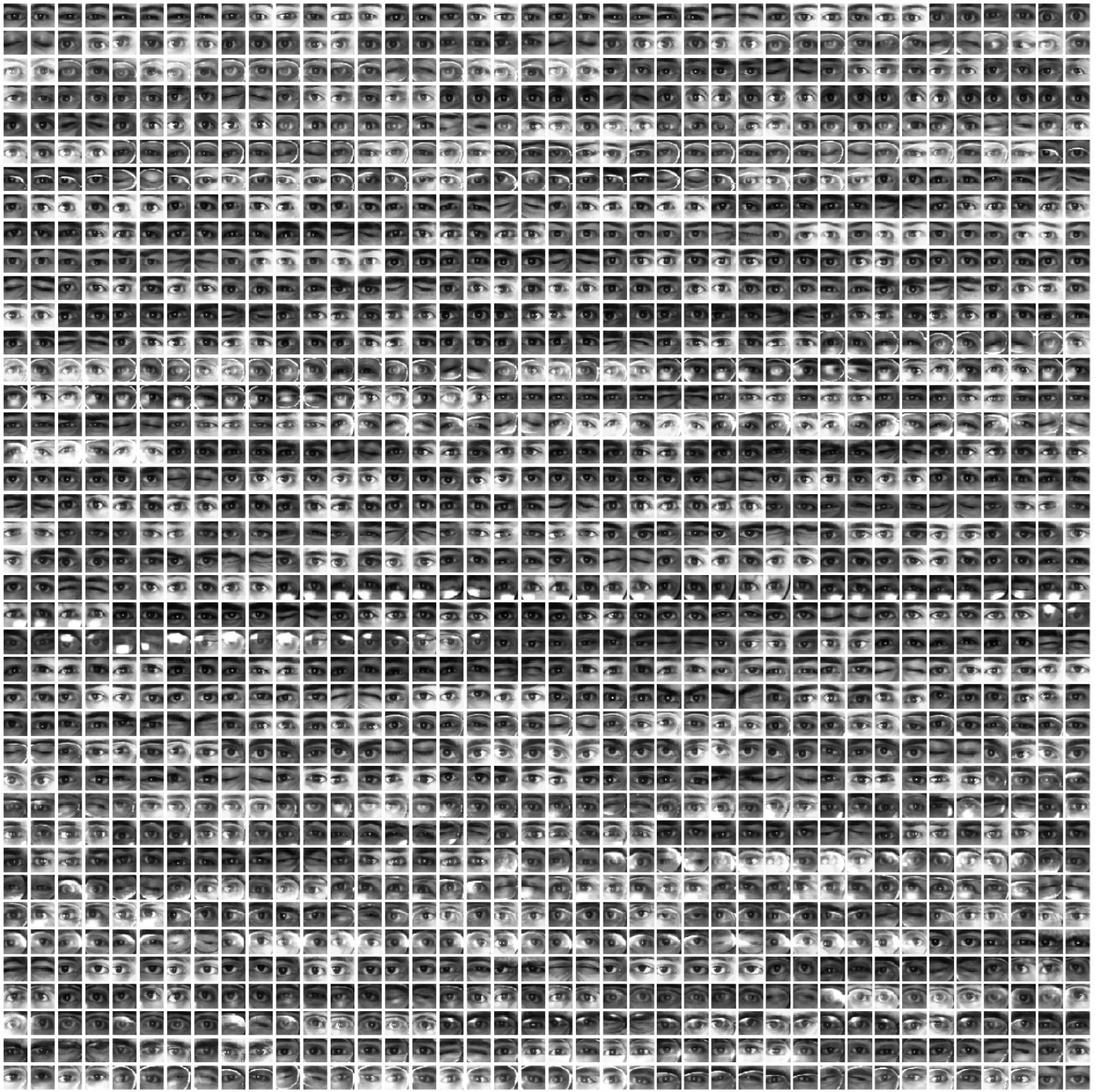
Ejemplo

Reconocedor de ojos y nariz

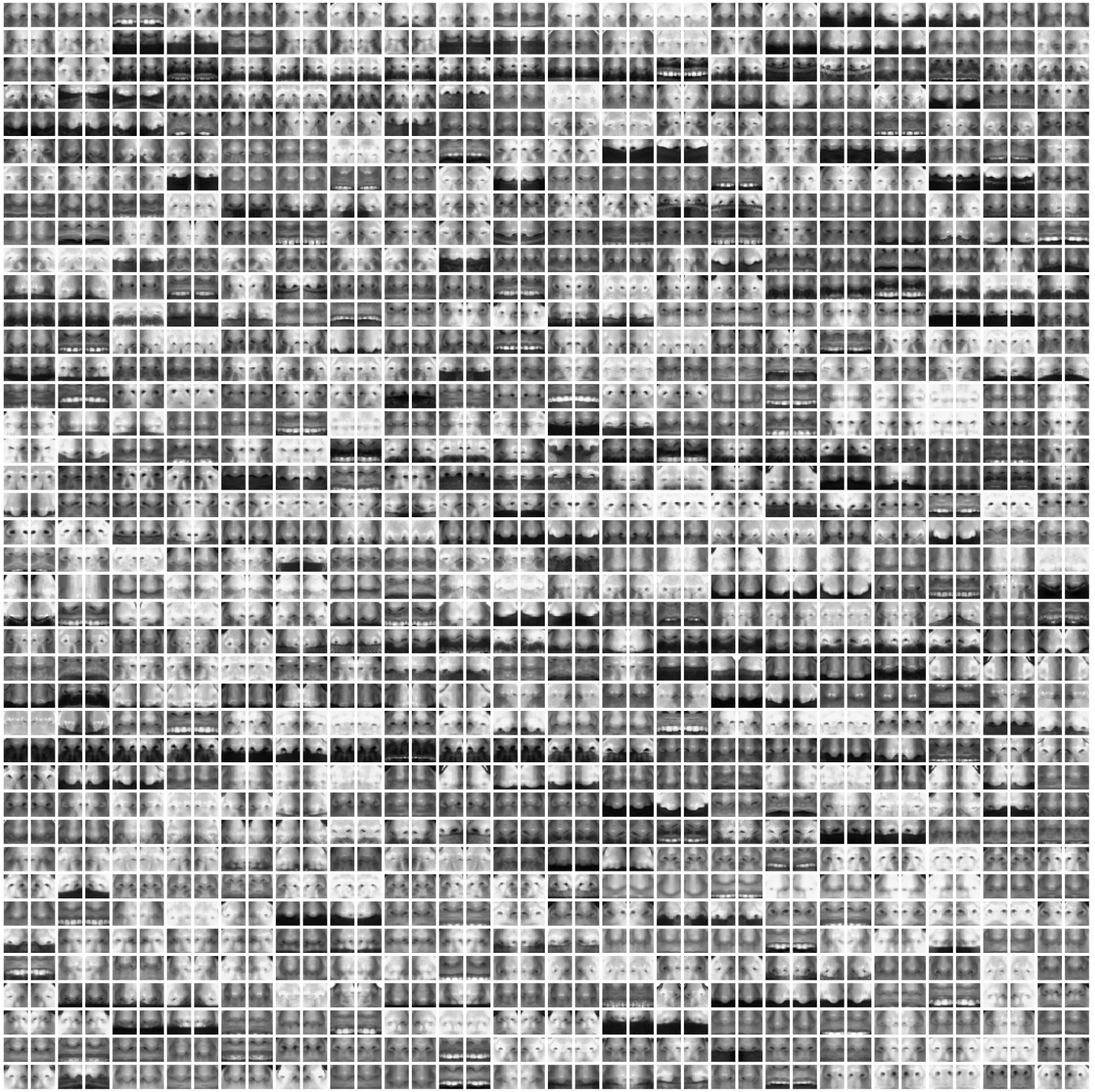

face_006_16.png
face_006_17.png
face_006_18.png
face_006_19.png
face_007_01.png
face_007_02.png
face_007_03.png
face_007_04.png

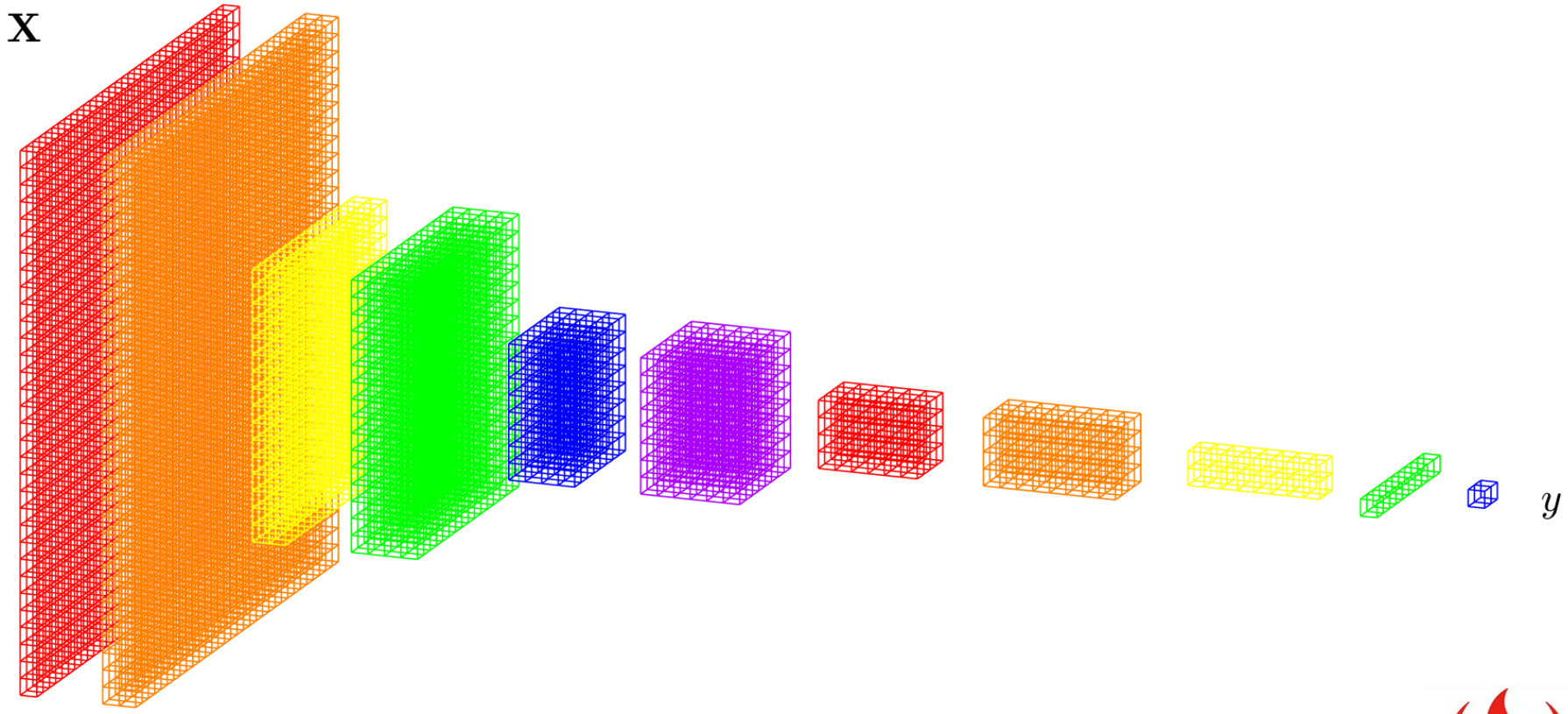

face_009_04.png
face_009_05.png
face_009_06.png
face_009_07.png
face_009_15.png
face_009_16.png
face_009_17.png
face_009_18.png

Ojos



Nariz




x
y

Implementation in PyTorch

```
class CNN_Classification(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(

            nn.Conv2d(in_channels=3, out_channels=4, kernel_size = 5, stride = 1, padding = 0),
            nn.ReLU(),
            nn.Conv2d(in_channels=4, out_channels=8, kernel_size = 5, stride = 6, padding = 0),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Flatten(),
            nn.Linear(32,2)
        )
```

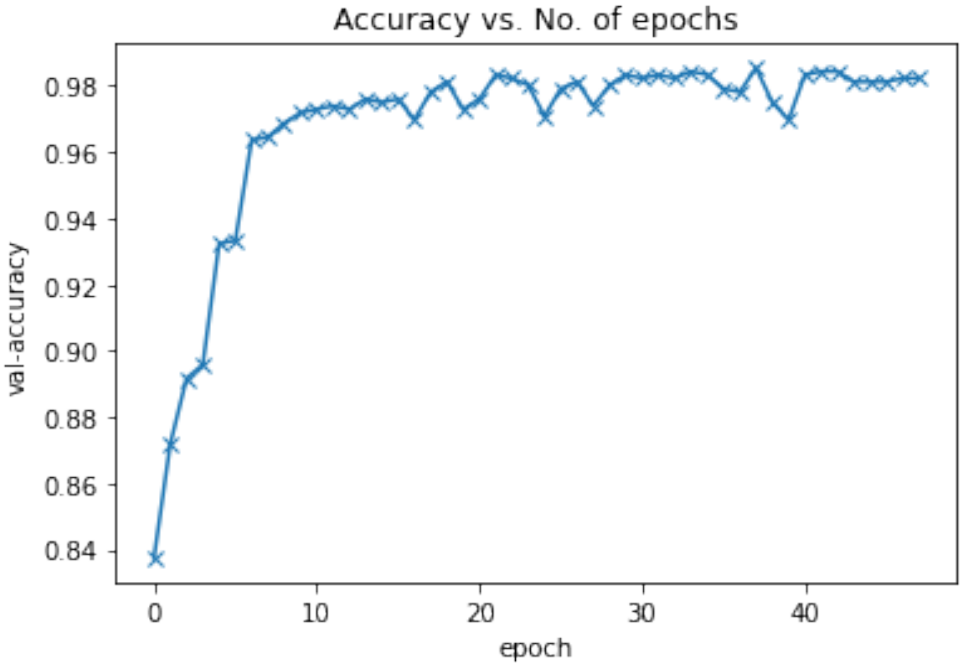
```
CNN_Classification(
  (network): Sequential(
    (0): Conv2d(3, 4, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(4, 8, kernel_size=(5, 5), stride=(6, 6))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Flatten(start_dim=1, end_dim=-1)
    (6): Linear(in_features=32, out_features=2, bias=True)
  )
)
```

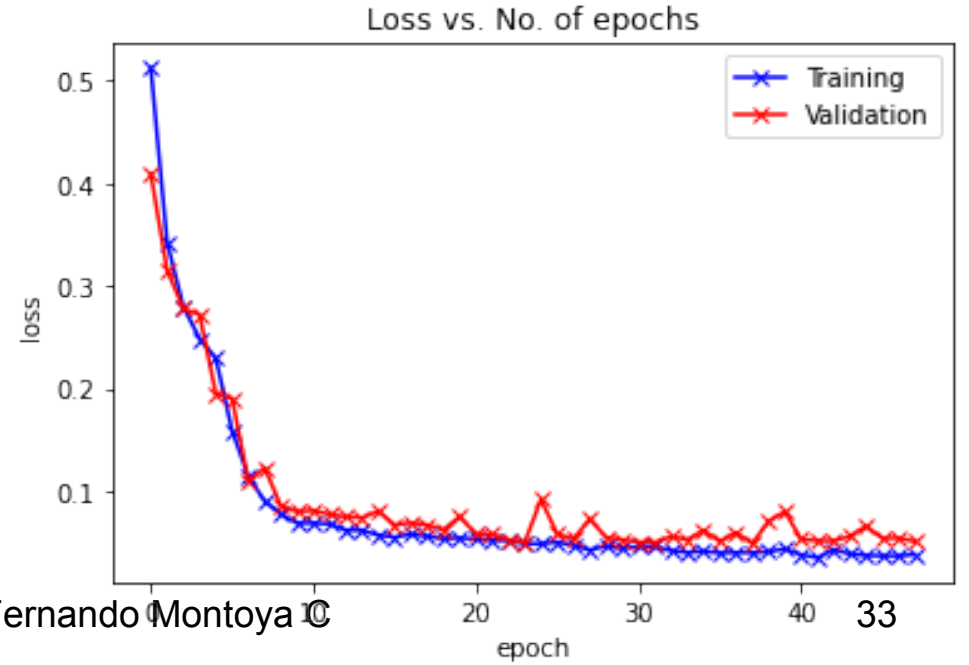
Epoch	Train-Loss	Val-Loss	Val-Acc	Best	Time [sec]
0	0.5126	0.4093	0.8375	***	1.8
1	0.3427	0.3156	0.8719	***	1.8
2	0.2797	0.2781	0.8917	***	1.8
3	0.2486	0.2719	0.8958	***	1.8
4	0.2295	0.1950	0.9323	***	1.8
5	0.1588	0.1904	0.9333	***	1.8
6	0.1145	0.1096	0.9635	***	1.8
7	0.0904	0.1229	0.9646	***	1.8
8	0.0776	0.0855	0.9687	***	1.8
9	0.0688	0.0801	0.9719	***	1.8
10	0.0689	0.0811	0.9729	***	1.8
11	0.0681	0.0774	0.9740	***	1.8
12	0.0620	0.0750	0.9729		1.8
13	0.0617	0.0742	0.9760	***	1.8
14	0.0572	0.0806	0.9750		1.8
15	0.0551	0.0670	0.9760		1.8
16	0.0580	0.0692	0.9698		1.8
17	0.0560	0.0671	0.9781	***	1.8
18	0.0538	0.0624	0.9813	***	1.8
19	0.0544	0.0752	0.9729		1.8
20	0.0533	0.0582	0.9760		1.8
21	0.0522	0.0578	0.9833	***	1.9
22	0.0519	0.0526	0.9823		1.8
23	0.0489	0.0503	0.9802		1.8
24	0.0488	0.0918	0.9708		1.8
25	0.0504	0.0582	0.9792		1.8
26	0.0470	0.0535	0.9813		1.8
27	0.0428	0.0728	0.9740		1.8
28	0.0463	0.0551	0.9802		1.8
29	0.0443	0.0527	0.9833	***	1.8
30	0.0459	0.0504	0.9823		1.8
31	0.0461	0.0501	0.9833		1.8
32	0.0420	0.0557	0.9823		1.8
33	0.0402	0.0528	0.9844	***	1.8
34	0.0414	0.0619	0.9833		1.8
35	0.0397	0.0510	0.9792		1.9
36	0.0403	0.0597	0.9781		1.8
37	0.0400	0.0491	0.9854	***	1.8
38	0.0410	0.0704	0.9750		1.8
39	0.0438	0.0802	0.9698		1.8
40	0.0382	0.0532	0.9833		1.9
41	0.0350	0.0517	0.9844		1.8
42	0.0432	0.0516	0.9844		1.8
43	0.0387	0.0561	0.9813		1.8
44	0.0374	0.0660	0.9813		1.8
45	0.0370	0.0540	0.9813		1.8
46	0.0369	0.0544	0.9823		1.8
47	0.0383	0.0509	0.9823		1.8

*** Early stop after 10 epochs with no improvement

Best model saved best_model.pt (val_acc = 0.9854 in epoch = 37)
Last model saved last_model.pt (val_acc = 0.9823 in epoch = 47)

Training Time: 85.82 sec





Epoch	Train-Loss	Val-Loss	Val-Acc	Best	Time [sec]
0	0.5126	0.4093	0.8375	***	1.8
1	0.3427	0.3156	0.8719	***	1.8
2	0.2797	0.2781	0.8917	***	1.8
3	0.2486	0.2719	0.8958	***	1.8
4	0.2295	0.1950	0.9323	***	1.8
5	0.1588	0.1904	0.9333	***	1.8
6	0.1145	0.1096	0.9635	***	1.8
7	0.0904	0.1229	0.9646	***	1.8
8	0.0776	0.0855	0.9687	***	1.8
9	0.0688	0.0801	0.9719	***	1.8
10	0.0689	0.0811	0.9729	***	1.8
11	0.0681	0.0774	0.9740	***	1.8
12	0.0620	0.0750	0.9729		1.8
13	0.0617	0.0742	0.9760	***	1.8
14	0.0572	0.0806	0.9750		1.8
15	0.0551	0.0670	0.9760		1.8
16	0.0580	0.0692	0.9698		1.8
17	0.0560	0.0671	0.9781	***	1.8
18	0.0538	0.0624	0.9813	***	1.8
19	0.0544	0.0752	0.9729		1.8
20	0.0533	0.0582	0.9760		1.8
21	0.0522	0.0578	0.9833	***	1.9
22	0.0519	0.0526	0.9823		1.8
23	0.0489	0.0503	0.9802		1.8
24	0.0488	0.0918	0.9708		1.8
25	0.0504	0.0582	0.9792		1.8
26	0.0470	0.0535	0.9813		1.8
27	0.0428	0.0728	0.9740		1.8
28	0.0463	0.0551	0.9802		1.8
29	0.0443	0.0527	0.9833	***	1.8
30	0.0459	0.0504	0.9823		1.8
31	0.0461	0.0501	0.9833		1.8
32	0.0420	0.0557	0.9823		1.8
33	0.0402	0.0528	0.9844	***	1.8
34	0.0414	0.0619	0.9833		1.8
35	0.0397	0.0510	0.9792		1.9
36	0.0403	0.0597	0.9781		1.8
37	0.0400	0.0491	0.9854	***	1.8
38	0.0410	0.0704	0.9750		1.8
39	0.0438	0.0802	0.9698		1.8
40	0.0382	0.0532	0.9833		1.9
41	0.0350	0.0517	0.9844		1.8
42	0.0432	0.0516	0.9844		1.8
43	0.0387	0.0561	0.9813		1.8
44	0.0374	0.0660	0.9813		1.8
45	0.0370	0.0540	0.9813		1.8
46	0.0369	0.0544	0.9823		1.8
47	0.0383	0.0509	0.9823		1.8

*** Early stop after 10 epochs with no improvement

Best model saved best_model.pt (val_acc = 0.9854 in epoch = 37)
Last model saved last_model.pt (val_acc = 0.9823 in epoch = 47)

Training Time: 85.82 sec

Training Confusion Matrix =
[[3616 45]
[36 3663]]

Training Accuracy = 0.9890

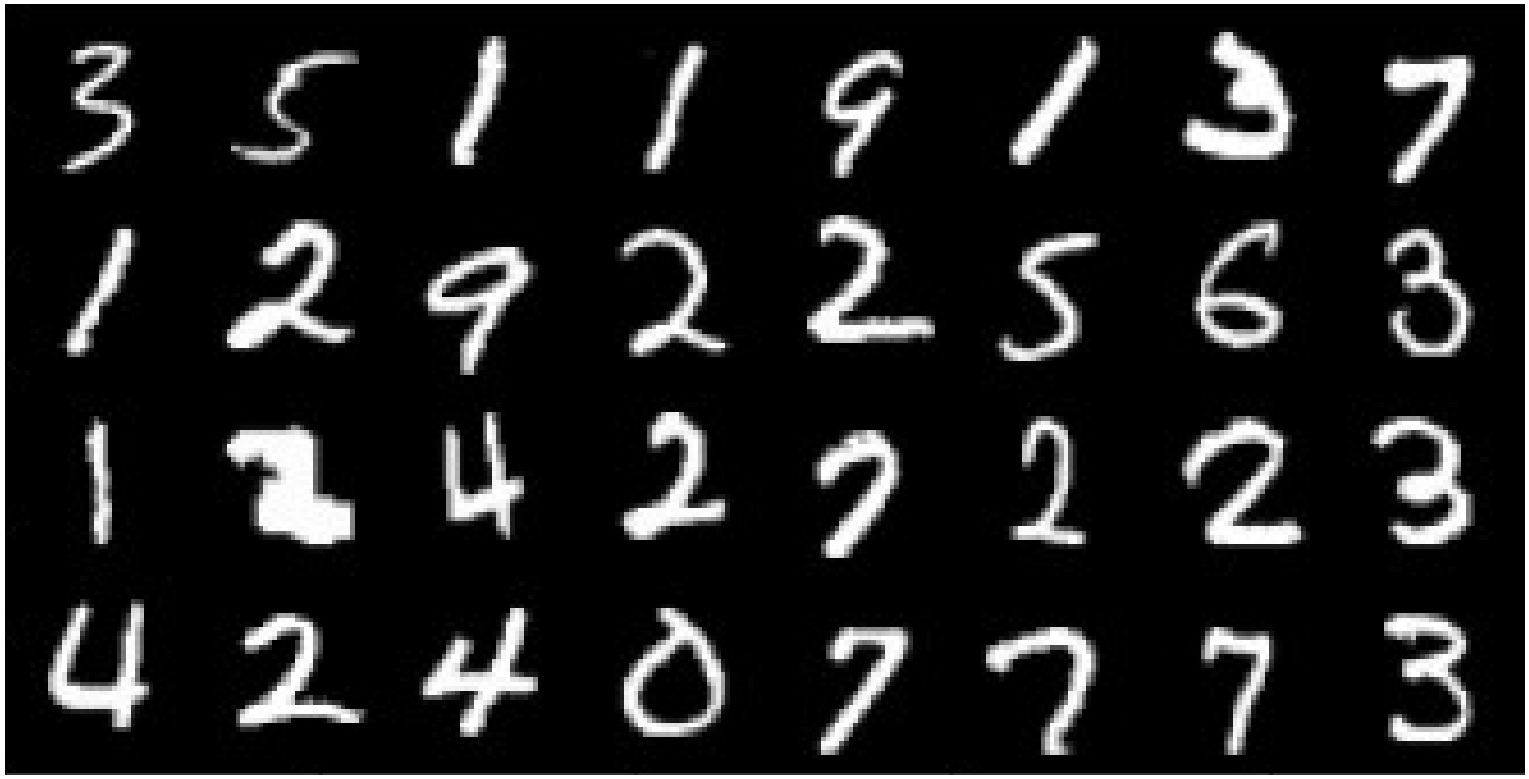
Validation Confusion Matrix =
[[490 9]
[5 456]]

Validation Accuracy = 0.9854

Testing Confusion Matrix =
[[1027 13]
[6 1034]]

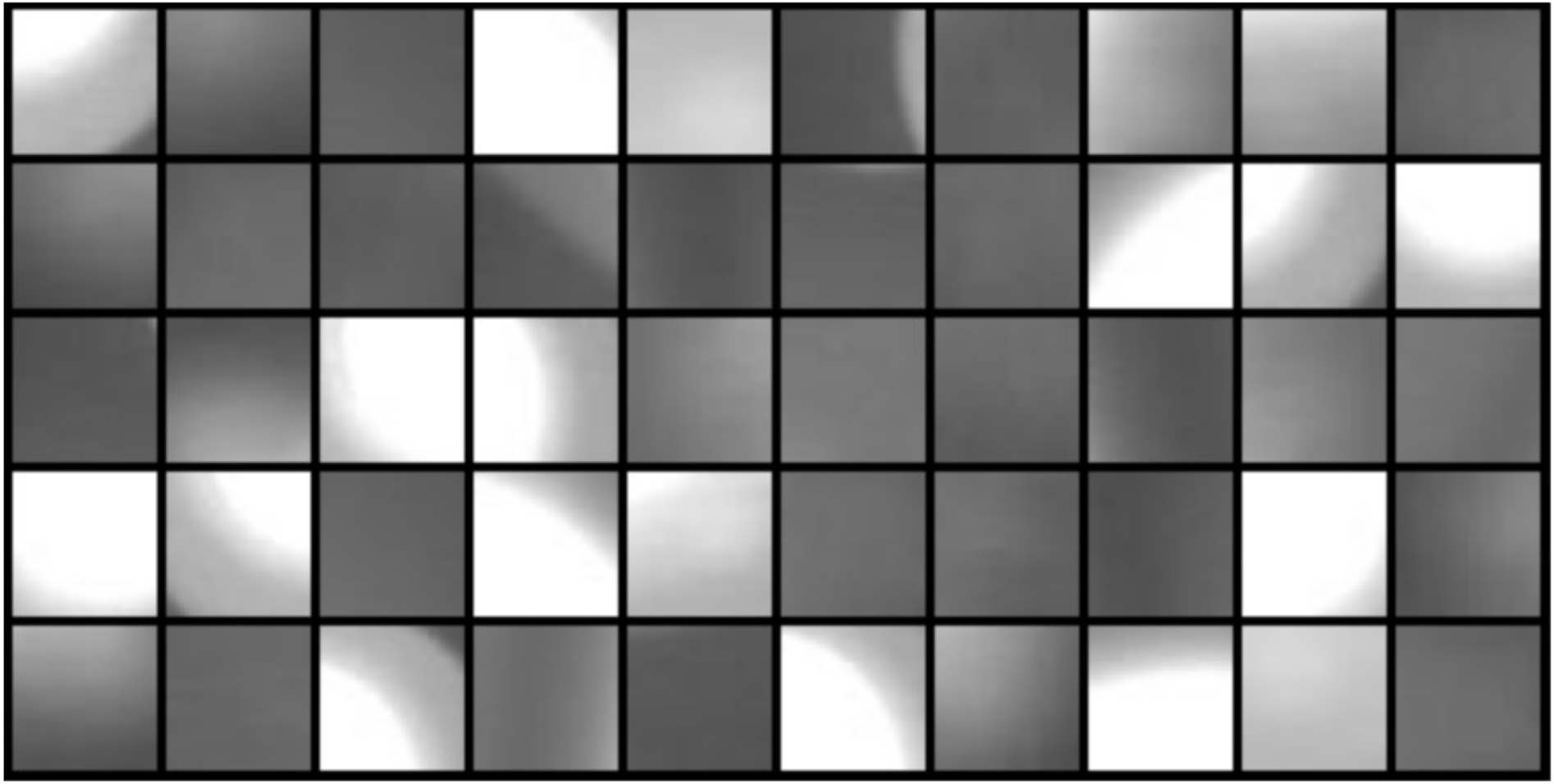
Testing Accuracy = 0.9909

Mas Ejemplos: MNIST



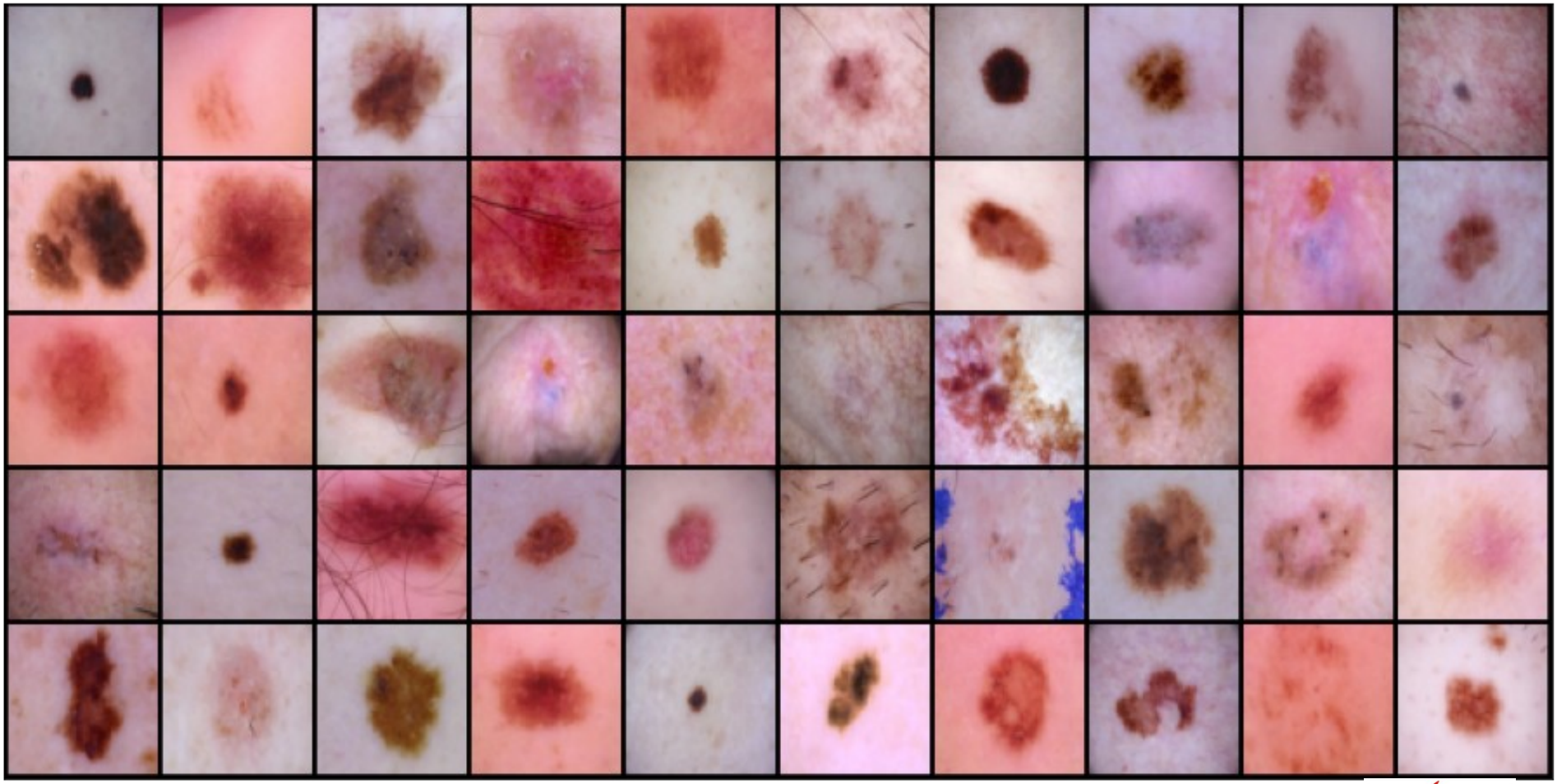
UPC
Universidad Peruana de ciencias Aplicadas

Mas Ejemplos: Defectos



Mas Ejemplos: Gatos vs Perros

Mas Ejemplos: Cáncer de piel 2



UPC
Universidad Peruana de ciencias Aplicadas

Method	Implementation	Features	Gun		Shuriken		Blade		Total
			Pr	Re	Pr	Re	Pr	Re	η
ResNet50	2	2048	1.00	0.90	1.00	1.00	0.35	0.55	0.80
VGG16	1	1000	0.95	0.86	0.56	1.00	0.36	0.84	0.73
VGG19	2	4096	0.78	0.72	0.27	1.00	0.18	0.58	0.53
AlexNet	2	4096	0.80	0.96	0.84	1.00	0.62	0.54	0.79
DenseNet121	1	1000	0.99	0.82	0.99	0.98	0.34	0.61	0.78
GoogleNet	2	1024	0.89	0.98	0.74	1.00	0.23	0.20	0.67
InceptionResNetV2	0	1536	0.90	0.60	0.82	0.99	0.55	0.60	0.74
InceptionV3	0	2048	0.85	0.69	0.76	1.00	0.47	0.40	0.69
MobileNet	2	1280	0.98	0.84	0.94	1.00	0.35	0.95	0.82
RCNN-ILSVRC13	2	4096	0.75	1.00	0.69	1.00	0.37	0.15	0.64
ShuffleNet	2	544	0.97	0.78	0.86	1.00	0.20	0.95	0.74
SqueezeNet	2	1000	0.96	0.38	0.27	1.00	0.06	0.35	0.42
Xception	0	2048	0.54	0.12	0.75	0.78	0.06	0.21	0.38
ZfNet512	2	1024	0.87	0.84	0.28	1.00	0.04	0.10	0.48

GRACIAS