

# TFG – Arquitectura de computadors i sistemes operatius

## FITA#05: Disseny de l'arquitectura de xarxes neuronals

### ***Gestió del projecte a Github:***

*Branca del repositori:*

<https://github.com/UOC-Assignments/uoc.tfg.jbericat/tree/FITA%2305>

*Dashboard de seguiment de les tasques associades a la fita:*

<https://github.com/UOC-Assignments/uoc.tfg.jbericat/projects/5>

**Estudiant:** Jordi Bericat Ruz

**Professor col·laborador:** Daniel Rivas Barragan

**Semestre:** Tardor 2021/22 (Aula 1)

**Versió:** ESBORRANY\_v5

# Índex

5 - Disseny de l'arquitectura de xarxes neuronals.....	1
5.1 - Tasques d'investigació i recerca .....	1
5.2 - Estructuració de l'algorisme de DL i configuració d'hyperparàmetres .....	2
5.2.1 – Identificació / definició de les mètriques de rendiment.....	2
5.2.2 - Establiment / disseny d'un model base (baseline model) .....	7
5.2.2.1 – AlexNet.....	14
5.2.2.2 – DenseNet121 .....	18
5.2.3 - Preparació de les dades per a l'entrenament del model.....	21
5.2.3.1 - Pre-processament de les imatges del dataset .....	21
5.2.3.2 - Divisió dels dataset en grups.....	27
5.3 - Avaluació del model i interpretació del seu rendiment .....	29
5.4 – Millores en el rendiment de la xarxa neuronal i reajustament d'hiper-paràmetres.....	30
5.4.x. DROPOUT REGULARIZATION .....	30
5.5 – Estudi del rati d'aprenentatge i optimització dels paràmetres .....	31
5.6 – <i>Wrapping-Up</i> : Adaptació i implementació de les arquitectures DCNN escollides.....	32

# 5 - Disseny de l'arquitectura de xarxes neuronals

## 5.1 - Tasques d'investigació i recerca

Donat que la implementació d'un model DCNN específic per a resoldre el problema proposat a la PdC d'aquest projecte no és una tasca gens trivial i que a més sobrepassa en complexitat els objectius marcats per al TFG, aleshores es decideix fer recerca de diferents models DCNN i metodologies d'entrenament i implementació ja existents per a avaluar si ens poden ser d'utilitat i així evitar haver de fer un disseny simplificat de xarxa neuronal profunda. En aquest sentit, s'ha trobat que els capítols 4 i 5 de la referència bibliogràfica [LLIBRE DL] ens poden servir de guió per a d'una banda preparar les dades (*datasets* imatges) per a ser processades pel model DCNN, i de l'altra per a avaluar i seleccionar una arquitectura DCNN ja existent per al seu ús en la PdC.

A més, s'ha complementat la informació obtinguda de la font anterior amb les que ofereix el següent enllaç:

<https://iq.opengenus.org/precision-recall-sensitivity-specificity/>

## 5.2 - Estructuració de l'algorisme de DL i configuració d'hyperparàmetres

### 5.2.1 – Identificació / definició de les mètriques de rendiment<sup>1</sup>

Definir les mètriques que s'utilitzaran per a mesurar el rendiment del model durant el seu entrenament és una tasca imprescindible per a poder monitoritzar si els canvis que es van realitzant en la configuració dels paràmetres a les diferents iteracions o *epoch* influeixen d'una manera o altra en la precisió (*accuracy*) de les prediccions realitzades pel model<sup>2</sup>.

Per a establir doncs un marc en el qual puguem definir aquestes mètriques es representarà el rendiment del model DCNN mitjançant una “**taula de confusió**” (mètode de representació de mesures d'encert en la predicció utilitzat habitualment durant l'entrenament de models de ML i DL).

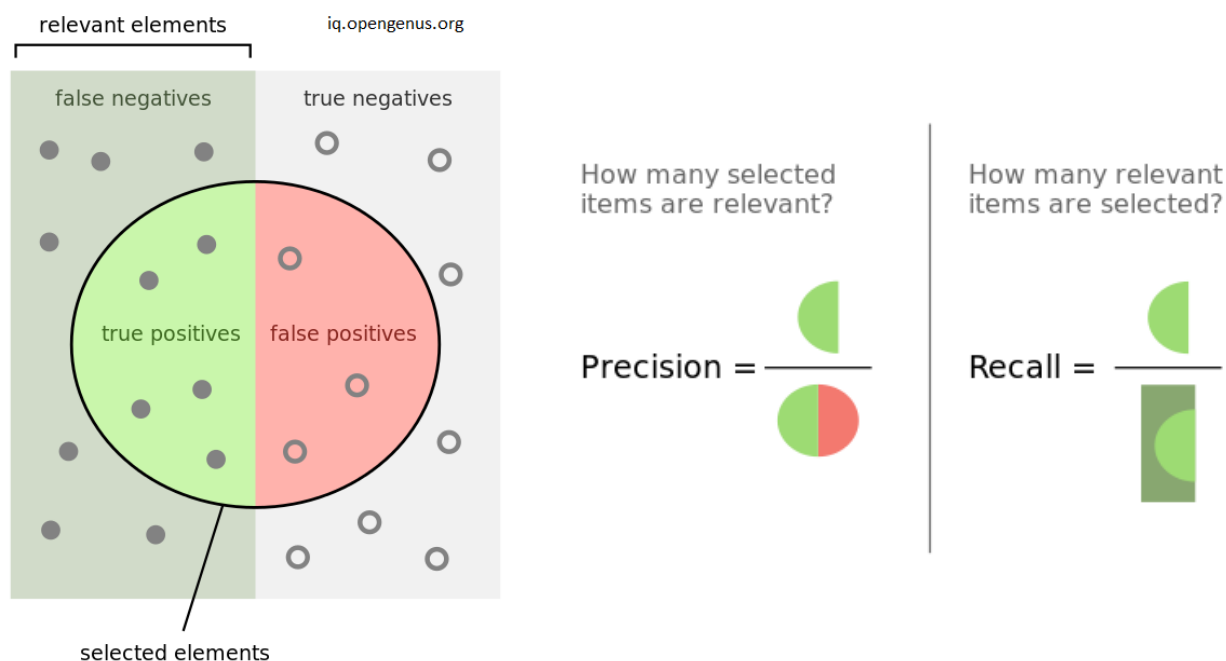
Simplificant, l'objectiu al representar el rendiment del model mitjançant d'una taula de confusió és el de poder centrar la mesura de l'encert de les prediccions en funció de, o bé la proporció de falsos positius (falses deteccions d'incendis) o bé de la de falsos negatius (no detecció d'un incendi). Així en funció de la importància de que es produeixi un o altre cas en el projecte de DL que s'estigui desenvolupant, caldrà tenir més cura de controlar els falsos positius (mesurats mitjançant la *precisió*), o bé els falsos negatius (mesurats mitjançant la *sensitivitat* o *recall*)

iq.opengenus.org		Predicted Class	
		NO	YES
Actual Class	NO	True Negative (TN)	False Positive (FP)
	YES	False Negative (FN)	True Positive (TP)

Imatge obtinguda de: <https://iq.opengenus.org/precision-recall-sensitivity-specificity/>

<sup>1</sup> <https://iq.opengenus.org/precision-recall-sensitivity-specificity/>

<sup>2</sup> Pàgina 148 del llibre de DL → “Defining the model evaluation metric is a necessary step because it will guide your approach to improving the System. Without clearly defined mètrics, it can be difficult to tell whether changed to a ML System result in progress or not”.



Imatges obtingudes de: <https://iq.opengenus.org/precision-recall-sensitivity-specificity/>

Així doncs, fetes les consideracions anteriors, en aquest punt cal decidir quina de les possibles falses prediccions pot implicar "pitjors" conseqüències a l'hora de predir si una imatge conté o no característiques pròpies d'un incendi forestal (objectiu central de la PdC d'aquest TFG).

Concretament, només es poden produir dos tipologies de falses prediccions:

- **Errada de tipus I: Fals positiu (*False Positive* o *FP*)**

Un "fals positiu" significa que s'ha reconegut / identificat un incendi a una imatge però en realitat no ho és (p.e. en el cas de la PdC, es podria donar el cas que un cos o objecte que emeti calor es detecti com la característica d'un incendi, per exemple, una pila de compost orgànic)

- **Errada de tipus II: Fals negatiu (*False Negative* o *FN*)**

Un fals negatiu significa que, havent-t'hi un incendi forestal a la imatge processada per l'algorisme d'aprenentatge profund, aquest no ha reconegut cap característica pròpia d'aquests.

Per tant, si tenim en compte que la detecció d'un incendi a les seves fases inicials és un factor crític que pot implicar la seva extinció abans que aquest causi danys a major escala, aleshores podem concloure que **tindrà pitjors conseqüències que es produeixi una predicció errònia de tipus II (FN)** que no pas de tipus I (FP). Podem determinar doncs que la mètrica de rendiment que ens permetrà identificar millor, o tenir una visió més precisa sobre la proporció d'incendis no detectats (FN) serà la basada en la **sensitivitat (*sensitivity / recall*)** o rati de positius veritaders (*True Positives o TP*). Dit d'una altra manera, la sensitivitat com a mètrica de rendiment ens permetrà saber amb millor exactitud que la precisió quantes vegades s'ha produït un FN durant cada iteració (*epoch*) d'entrenament del model, fet que ens ajudarà a fer un reajustament més precís dels paràmetres (pesos de les arestes del graf) i en conseqüència a obtenir un model final completament entrenat que mostri un rendiment més acurat (*model performance & accuracy*).

Podem calcular la sensitivitat de les dades mesurades amb la següent expressió:

$$\text{sensitivitat} = \frac{TP}{TP + FN}$$

Un cop realitzat l'estudi anterior per a determinar la millor manera de mesurar el rendiment del model a l'hora de resoldre el problema binari de classificar imatges en funció de si contenen o no característiques d'incendis, passem a ara a estudiar el cas en el què el resultat de resoldre el problema de classificació ha de ser la categorització d'un incendi en funció de les seves característiques, tal com es defineix a la següent taula:

<b>Categoria d'incendi (Intensitat x Mida)</b>	Localitzat	Moderadament estès	Molt estès
Intensitat Baixa	A	B	C
Intensitat Moderada	D	E	F
Intensitat Alta	G	H	I

Abans però caldrà fer notar que, tenint en compte que s'està desenvolupant una PdC, per a simplificar considerarem les següents prioritats (essent "A" el nivell o categoria menys crítica i "I" el nivell o categoria més crítica) sense tenir en compte altres factors, com podria ser la proximitat a zones habitades:

$$A < B < C < D < E < F < G < H < I$$

En aquest nou escenari podem definir:

- **Errada de tipus I: Fals positiu (*False Positive* o *FP*)**

Ara, un “fals positiu” significa que havent processat una imatge que conté un incendi de categoria inferior (o bé sense cap incendi), el model n’ha identificat un, i de categoria superior si la imatge en contenia característiques.

- **Errada de tipus II: Fals negatiu (*False Negative* o *FN*)**

Ara, un fals negatiu significa que, havent processat una imatge que conté un incendi de categoria superior, el model n’ha identificat un de categoria inferior, o bé no n’ha identificat cap.

Observem que, de la mateixa manera que passava amb el problema de classificació binari, la mètrica més adequada serà la que ens permeti tenir una visió més acurada dels falsos negatius (FN) que es produeixin durant l’entrenament del model, és a dir, la **sensitivitat** de les mesures. Com a conseqüència o *side-effect* però, tindrem que amb aquesta mètrica no tindrem dades tant precises de la quantitat de falsos positius que es produeixin. Tanmateix, a la pràctica en el cas que ens ocupa això només podria suposar que en un moment donat algun dels drons de l’eixam intenti informar de / apagar un incendi inexistent, o bé que s’hi dediquin més recursos dels necessaris per a gestionar la situació, com a conseqüència d’un entrenament poc acurat del model. Així doncs, un cop realitzades les primeres mesures amb la mètrica seleccionada (sensitivitat o *recall*) contra el *dataset* d’explotació (veure [secció 5.3](#)), si s’observa un rendiment pobre en el cas de falsos positius el que caldrà fer serà tornar a fer l’entrenament del model utilitzant una mètrica menys conservadora, però que ens doni una visió més precisa dels falsos positius (sota risc de què es comencin a detectar casos de falsos negatius, cas en el que s’hauria d’avaluar si paga la pena el canvi de mesura de rendiment donades les fatals conseqüències que pot tenir **una sola** predicció errònia de tipus II en un incendi forestal). En aquest sentit, la mètrica basada en *F-Score*<sup>3</sup> és una possibilitat que cal tenir en compte i que s’estudiarà en funció dels resultats (*accuracy*) de les mesures basades en la sensitivitat (*recall*).

---

<sup>3</sup> Refs: Punt 4.1.4, pàg. 149 llibre DL

Podem calcular el *F-Score* de les dades mesurades de la següent manera:

$$r = \text{sensitivitat} = \frac{TP}{TP + FN}$$

$$p = \text{precisió} = \frac{TP}{TP + FP}$$

$$F - \text{Score} = \frac{2pr}{p + r}$$



## 5.2.2 - Establiment / disseny d'un model base (*baseline model*)

En primer lloc es procedeix a realitzar un estudi comparatiu superficial ([taula 5.X.X.X](#)) de les característiques d'aquelles arquitectures de xarxes neuronals profundes més conegudes<sup>4</sup> per tal d'escollir aquella que s'ajusti millor a les dimensions del problema que volem resoldre i a als recursos computacionals dels que es disposa<sup>5</sup> (estació de desenvolupament descrita a la [secció 1.X.X.X](#)). Com a criteris per a realitzar l'estudi comparatiu es tindran en compte 3 factors; d'una banda, la precisió assolida pel model mesurada en l'escala "Top-1". De l'altra, la complexitat computacional en termes d'instruccions per segons (en G-FLOPS) requerides per a entrenar el model, i finalment, la quantitat de nodes (paràmetres) de cada arquitectura. La [figura 5.x.x.x](#) cataloga segons aquests criteris les DCNN més destacades fins a 2018:

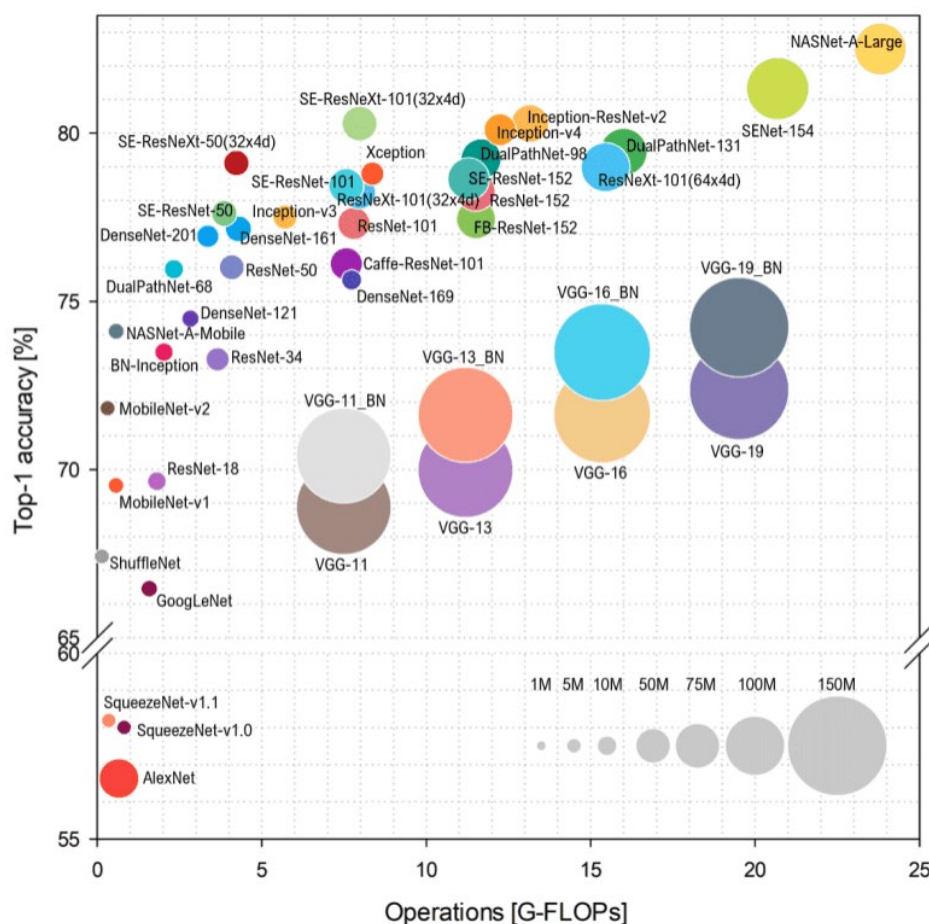


Figura 5.x.x.x – Visió general d'arquitectures DCNN fins a 2018 – Font: <https://theaisummer.com/cnn-architectures/>

<sup>4</sup> Referències bibliogràfiques: Capítol 5 Llibre DL, [pàgs: 199](#), <https://theaisummer.com/cnn-architectures/>

<sup>5</sup> Tanmateix, per a l'entrenament del model també es podria fer ús de plataformes tercers com per exemple "Google AI platform" → <https://cloud.google.com/ai-platform/training/docs/using-gpus>

Arquitectura	Característiques principals	#Paràmetres	Anàlisi de rendiment	Complexitat
DCNN				
LeNET	L'arquitectura està composta per 5 capes, 3 de convolucional que s'encarreguen de la extracció de característiques de les imatges, i dues de completes que implementen la classificació.	61K	Presenta un bon rendiment (fins a un 99%) amb datasets d'imatges en escala de grisos i aplicades a problemes de classificació que no superin les 10 classes <sup>6</sup> .	Baixa
AlexNET	Tot i oferir una arquitectura semblant a LeNET, AlexNET implementa noves característiques i una major quantitat de capes convolucional (5) <sup>7</sup> que li proporcionen una capacitat d'aprenentatge major que la primera en termes de complexitat de les característiques que pot arribar a reconèixer <sup>8</sup> .	60M	En problemes de classificació, el rendiment que ofereix AlexNet mesurat en escala Top-5 <sup>9</sup> és del 15,3%, valor que supera la precisió d'encert d'altres models posteriors al seu desenvolupament.	Moderada
VGGNet	Comparada amb les arquitectures vistes fins al moment, VGGNet destaca per la	Entre 138M i	Si utilitzem el ratio Top-5 amb el que s'ha mesurat el rendiment de	Alta

<sup>6</sup> Ref. P203 Llibre DL → "LeNet performance on the MNIST dataset: When you train LeNet-5 on the MNIST dataset, you will get above 99% accuracy"

<sup>7</sup> Ref. P205 Llibre DL → "AlexNet consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully connected layers with a final 1000-way softmax"

<sup>8</sup> Ref. P204 Llibre DL → "AlexNet has about 60 million parameters and 650,000 neurons, which gives it a larger learning capacity to understand more complex features. This allowed AlexNet to achieve remarkable performance"

<sup>9</sup> El rati Top-1 i Top-5 són utilitzats de manera habitual per a descriure la precisió d'un algorisme de classificació, de manera que el rati Top-1 representa la proporció de vegades en funció del temps que el classificador no ha proporcionat la classe que s'esperava, mentre que el rati Top-5 indica el percentatge vegades que la resposta correcta no estava entre les 5 primeres propostes de predicció del model. Refs: P211 Llibre DL

	seva arquitectura uniforme i quantitat mes reduïda de paràmetres per capa <sup>10</sup> , que la fan més fàcil d'entendre i gestionar <sup>11</sup> . Tot i aquesta uniformitat, els components que formen l'arquitectura són els mateixos que els vistos a les anteriors, però en aquest cas trobem una major profunditat de la xarxa: més capes convolucional (13) i de completes (3)		l'arquitectura AlexNet, podem destacar una <b>important millora del rendiment</b> de VGG16 respecte del primer, ja que utilitzant el mateix dataset d'imatges aconsegueix una ratio del 8.1%.	
GoogLeNet	Aquesta arquitectura es caracteritza per ser de les que ofereix una major profunditat amb 22 capes convolucional. Tanmateix, tot i estar basada en les arquitectures vistes anteriorment (AlexNet i VGGNet), la quantitat de paràmetres que requereix es molt inferior.	13M	Seguint el mateix criteri que amb les arquitectures anteriors, podem dir que GoogLeNet és la DCNN que ha aconseguit un millor rati Top-5, assolint valors molt baixos de fins el 6.67%, que denota un rendiment del model aproximat al que ofereix el cervell humà (pel que fa a resoldre problemes de classificació).	Molt alta
ResNet	L'arquitectura de xarxes neuronals residuals desenvolupada per Microsoft		ResNet és capaç d'assolir un ratio Top-5 mínim de 3,57% en les	Moderada

<sup>10</sup> **Ref. P214 llibre DL** → "VGG16 yields ~138 million parameters; VGG19, which is a deeper version of VGGNet, has more than 144 million parameters. VGG16 is more commonly used because it performs almost as well as VGG19 but with fewer parameters. VGGNet, has more than 144 million parameters. VGG16 is more commonly used because it performs almost as well as VGG19 but with fewer parameters."

<sup>11</sup> **P210 llibre DL** → "Both LeNet and AlexNet have many hyperparameters to tune. The authors of those networks had to go through many experiments to set the kernel size, strides, and padding for each layer, which makes the networks harder to understand and manage. VGGNet (explained next) solves this problem with a very simple, uniform architecture".

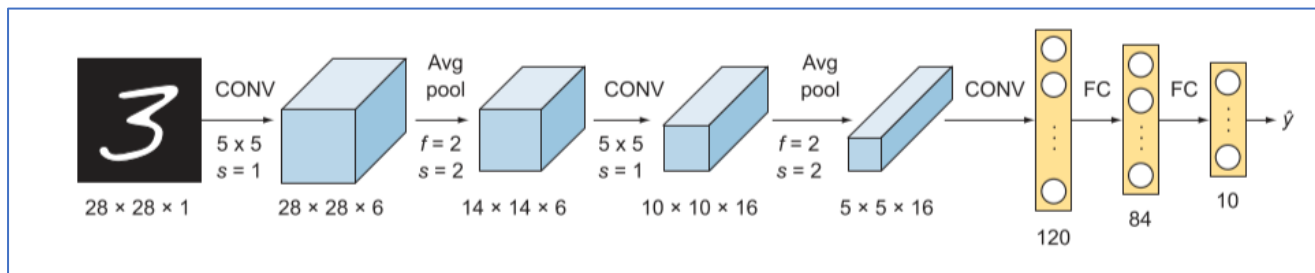
	ResNet es caracteritza per introduir una tècnica de normalització per lots que li permet augmentar la profunditat de la xarxa (fins a 152 capes a les versions més profundes) tot reduint a l'hora la seva complexitat computacional i augmentant el seu rendiment <sup>12</sup> .		mateixes condicions que les altres arquitectures vistes.	
DenseNet	Treballs posteriors al desenvolupament de ResNet <sup>13</sup> han sigut capaços de demostrar que habilitant la interconnexió de les diferents capes del model entre elles (especialment cap aquelles més properes a la entrada i sortida de l'arquitectura) és possible augmentar la profunditat de la xarxa (des de 121 fins a 201) i en conseqüència la precisió de les prediccions sense que això impliqui un increment de la quantitat de paràmetres i de la complexitat.	???		

**Taula 5.X.X.X:** Comparativa d'algunes de les arquitectures DCNN avançades més destacades desenvolupades des de 1998 fins a 2018

<sup>12</sup> **Refs. P230 llibre DL** → “The Residual Neural Network (ResNet) was developed in 2015 by a group from the Microsoft Research team.5 They introduced a novel residual module architecture with skip connections. The network also features heavy batch normalization for the hidden layers. This technique allowed the team to train very deep neural networks with 50, 101, and 152 weight layers while still having lower complexity than smaller networks like VGGNet (19 layers). ResNet was able to achieve a top-5 error rate of 3.57% in the ILSVRC 2015 competition, which beat the performance of all prior ConvNets”

<sup>13</sup> <https://arxiv.org/abs/1608.06993> → “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.”

A continuació es mostra el diagrama de cadascuna de les architectures estudiades a la taula 5.x.x.x. Cal destacar que amb l'ajuda d'aquests, a simple vista és possible arribar a fer-se una idea de la profunditat (i per tant de la precisió) de cada model:



Imatge obtinguda de [LLIBRE DL, Pàg.]

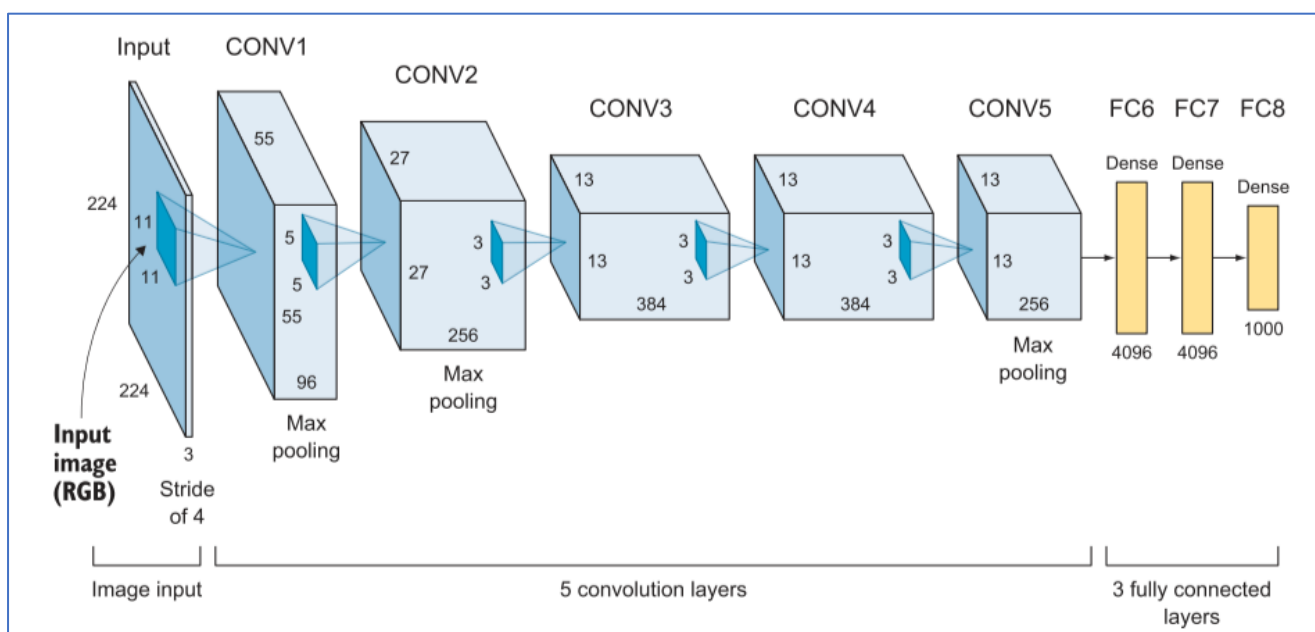


Figura 5.xxx – Arquitectura AlexNet – Font: [LLIBRE DL, Pàg.]

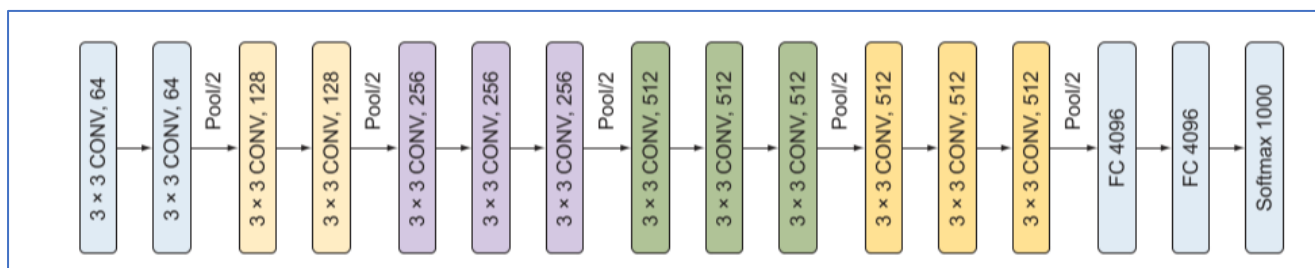


Figura 5.xxx – Arquitectura VGGNet16 – Font: [LLIBRE DL, Pàg.]

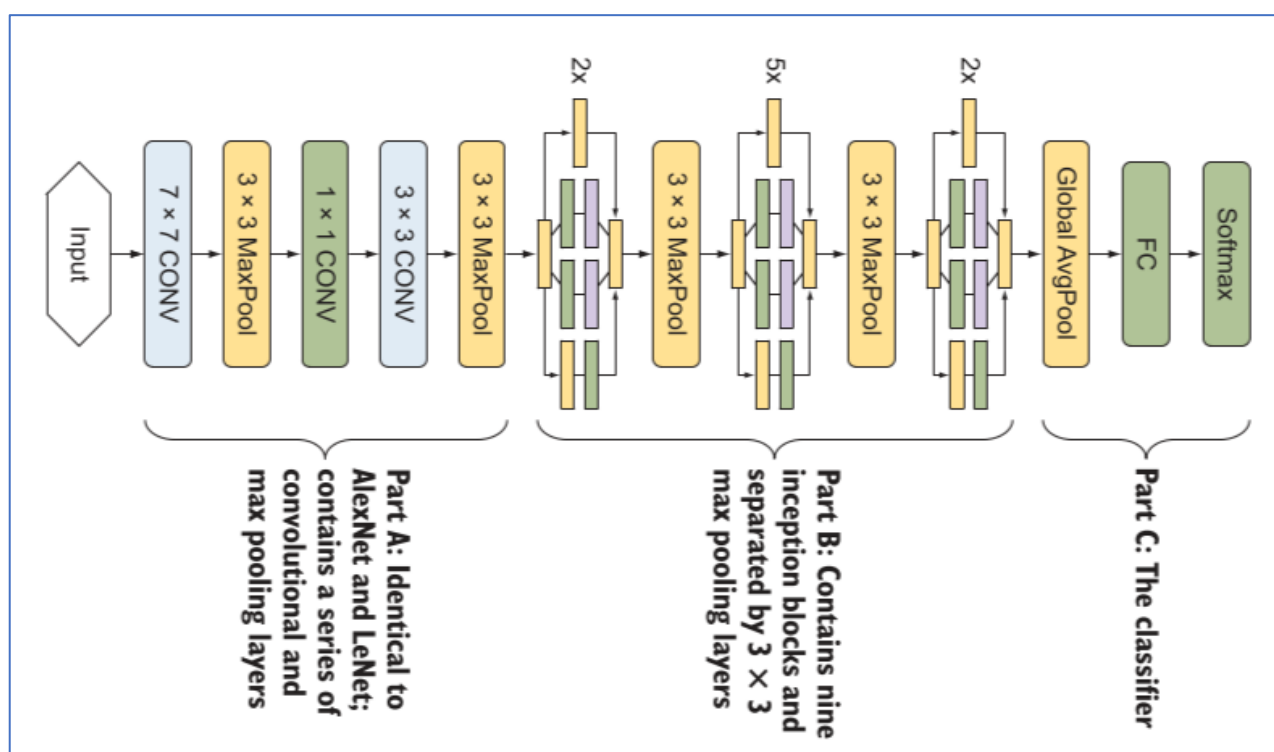
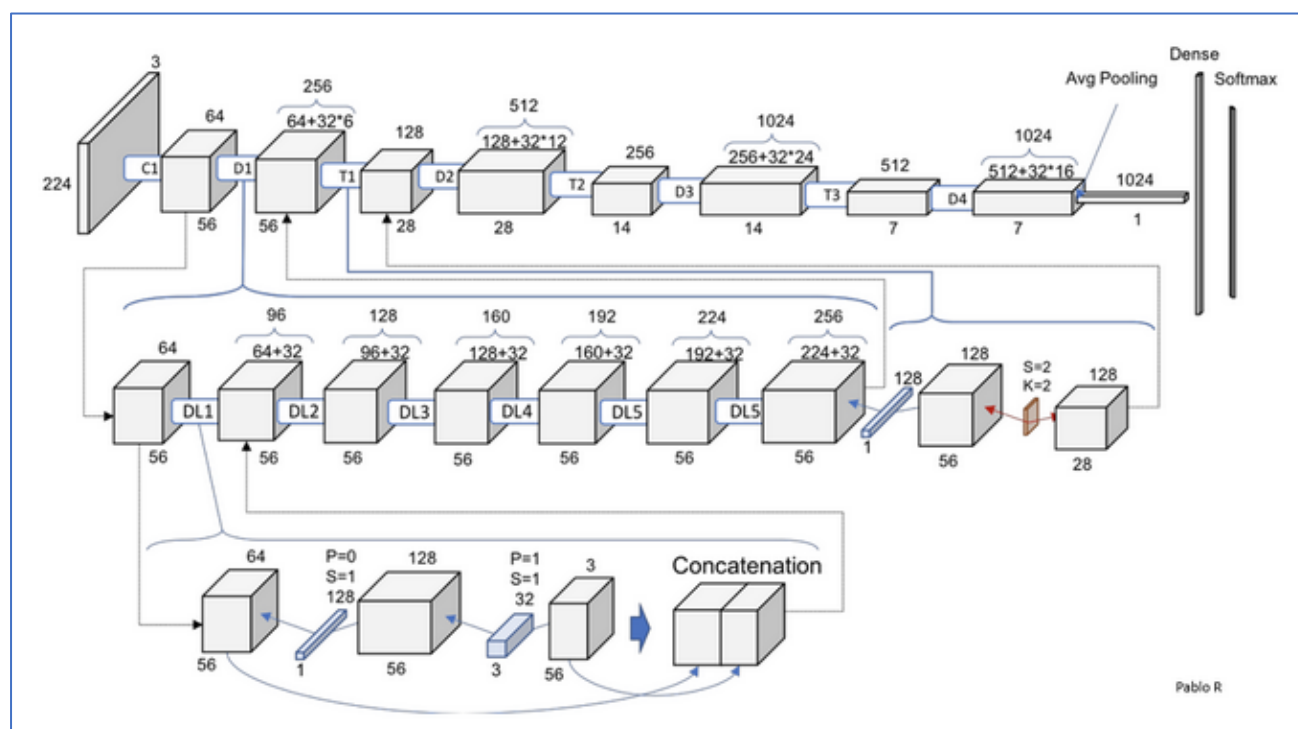


Figura 5.xxx – Arquitectura GoogLeNet – Font: [LLIBRE DL, Pàg. ]



**Figura 5.xxx** – Arquitetura DenseNet - Font: *Pablo R.*

Un cop analitzades les opcions anteriors, i realitzat un breu exercici de recerca inicial<sup>14</sup>, de partida es descarten aquelles que presenten una complexitat d'implementació molt elevada (GoogLeNet i ResNet queden descartades en aquest sentit). D'altra banda, degut a la seva obsolescència (data del 1998), també es descarta LeNet per a la realització d'aquesta PdC, tot i que els estudis de rendiment realitzats ofereixen uns resultats acceptables en les condicions de classificació d'aquesta (menys de 10 classes i imatges en escala de grisos). Així doncs, passarem a realitzar la implementació de dos dels tres models restants (AlexNet i DenseNet) mitjançant la llibreria d'alt nivell "Keras" i el framework "TensorFlow" per a Python. Pel que fa VCC16, la descartem degut a què com s'ha vist a la **figura 5.x.x.x**, la precisió que ofereix el model en relació a la seva complexitat és relativament pobre si la comparem amb el model DenseNet, que proporciona millors resultats tot i requerir la configuració de molts menys paràmetres que VCC16. Pel que fa AlexNet, tot i ser una model obsolet, es realitzarà també la seva implementació per qüestions didàctiques, ja que és un bon punt de partida per a entendre el funcionament d'arquitectures més avançades, com és el cas de DenseNet. D'aquesta manera també podrem realitzar un anàlisi comparatiu de com ha sigut la evolució de les DCNN durant la darrera dècada. En aquest sentit, més endavant estudiarem (**secció 6**), per a cadascun dels models implementats, el nivell de complexitat de configuració dels hyperparàmetres vs. complexitat computacional necessària per a processar els dataset vs. rendiment obtingut, tot fent ús de les mateixes mètriques que s'han establert a la [secció 5.2.1](#)). Un cop s'hagin obtinguts les conclusions pertinents a la **secció 6**, es prendrà la decisió de quin model s'acaba utilitzant per a desenvolupar la resta d'aquesta PdC.

<sup>14</sup> Es pot consultar el procediment d'implementació de googLeNet i ResNet amb TensorFlow + Keras als següents enllaços, respectivament → <https://medium.com/mlearning-ai/implementation-of-googlenet-on-keras-d9873aeed83c> i <https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/>



### 5.2.2.1 – AlexNet

El model AlexNet, amb 60 milions de paràmetres i al voltant de 650.000 neurones és una de les arquitectures que en el moment del seu desenvolupament va causar un impacte més significatiu, degut a que va permetre demostrar que la utilització de xarxes neuronals per a resoldre problemes de classificació d'una certa complexitat aplicats a visió per computador<sup>15</sup>. Concretament, AlexNet està capacitat per a classificar una imatge d'entrada d'entre fins a 1000 classes diferents.

#### Arquitectura:

AlexNet implementa una arquitectura de xarxa neuronal formada per 5 capes convolucionals per a la extracció de característiques (mitjançant una sèrie de Kernels de mida determinada, en funció de la capa) i 3 de completes, que tenen com a sortida la funció de classificació softmax. D'altra banda, AlexNet també incorpora un seguit de capes de “pooling màxim d'encavalcament” (Overlapping Max Pooling) que ajuden a reduir lleugerament el coeficients Top-1 i Top-5 de predicció.

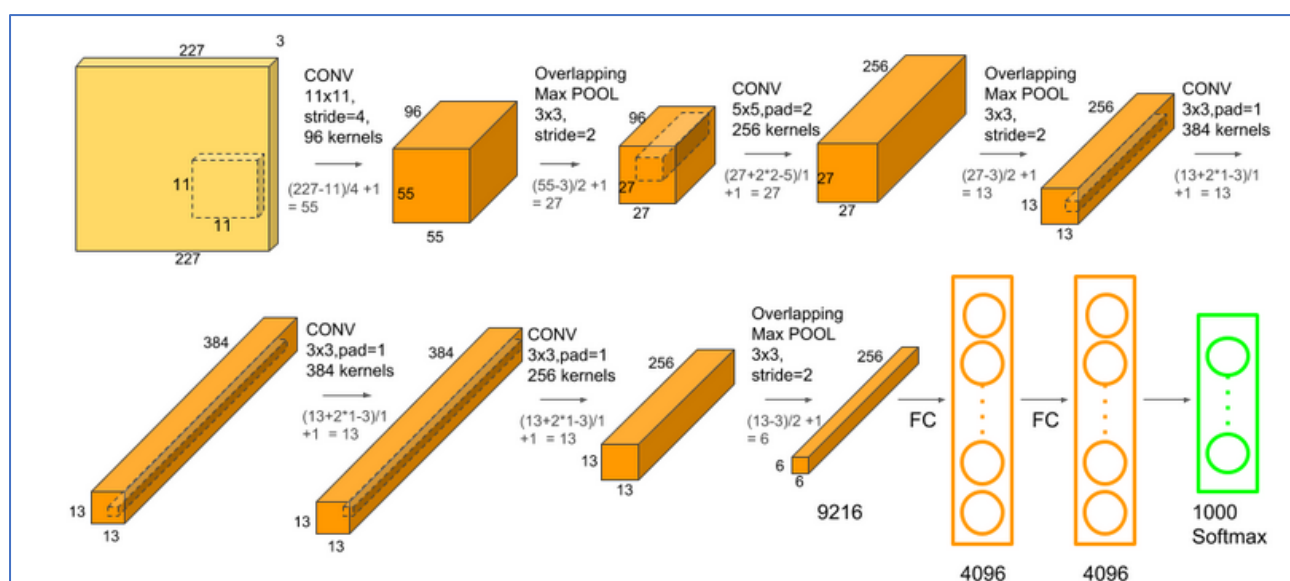


Figura 5.x.x.x – Arquitectura del model AlexNet – Font: <https://learnopencv.com/understanding-alexnet/>

<sup>15</sup> <https://learnopencv.com/understanding-alexnet/> → [...] with the publication of the paper, “ImageNet Classification with Deep Convolutional Neural Networks” by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, he and a handful of researchers were proven right. It was a seismic shift that broke the Richter scale! The paper forged a new landscape in Computer Vision by demolishing old ideas in one masterful stroke.



### Característiques destacades:

Un dels trets diferencials del model AlexNet respecte dels seus predecessors l'any 2012 i que va permetre accelerar l'entrenament de xarxes relativament profundes va ser la utilització d'una funció d'activació<sup>16</sup> *ReLU* (*Rectified Linear unit*), que permet una convergència de la xarxa molt més ràpida que altres funcions d'activació utilitzades amb anterioritat, com *tanh* o *sigmoid*. Podem representar la funció d'activació ReLu de la següent manera<sup>17</sup>:

$$f(x) = \max(0, x)$$

On es defineix que  $f(x)$  tindrà com a sortida el major valor entre 0 i  $x$ . Ho podem veure clar si ho expressem com una funció definida a trossos:

$$f(x) = \begin{cases} 0, & \text{si } x < 0 \\ x, & \text{si } x \geq 0 \end{cases}$$

Observem que la sortida (imatge de  $x$ ) serà igual a zero (FALSE) sempre i quan la entrada ( $x$ ) sigui un valor negatiu. En cas contrari, tindrem que la sortida serà la mateixa entrada, és a dir " $x$ ". Si ens fixem en la gràfica de les funcions ReLu i *tanh* podem veure que a la segona, el pendent de la recta per a valors en l'interval  $(-\infty, -2) \cup (2, +\infty)$  és molt pròxim a zero, factor que pot decelerar el procés de descens gradual (*gradient descent*) per l'arquitectura de capes.

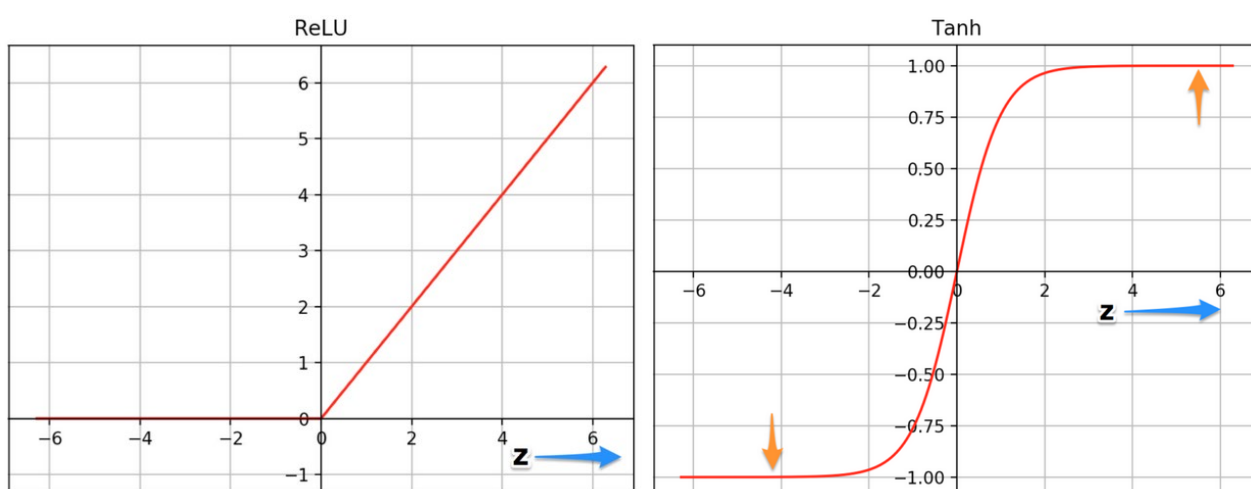


Figura 5.xxx - Imatges obtingudes de <https://learnopencv.com/understanding-alexnet/>

<sup>16</sup> Recordem que les funcions d'activació s'encarreguen de determinar si s'ha reconegut o no una característica (resultat binari) **DESENVOLUPAR I CERCAR REFS.**

<sup>17</sup> <https://deeptai.org/machine-learning-glossary-and-terms/relu>

### Altres característiques rellevants del model:

Hi ha una sèrie de factors que hem de tenir en consideració a l'hora d'adaptar les imatges del dataset d'aquesta PdC i poder utilitzar-lo per a l'entrenament del model AlexNet:

- a) **Mida** → Les imatges del dataset han de ser convertides a mida 256x256 mitjançant la funció de pre-processament corresponent (veure [secció 5.2.3.1](#))
- b) **Color** → Les imatges han de ser en RGB (3 canals), però si s'alimenta el model amb imatges en escala de grisos (com es el cas), aleshores la mateixa implementació de l'arquitectura s'encarrega de replicar el canal únic de l'escala de grisos a tres matrius exactament iguals.

### Implementació

A continuació es detalla la implementació amb TensorFlow i Python del model AlexNet. Cal fer notar que a la [secció 5.6](#) es detallarà la seqüència completa de construcció del model utilitzant a les mostres dels dataset generats a la [secció 4](#), que són específiques per a la realització d'aquesta PdC.

## # 1. Instal·lació de dependències

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import os
import time
```

## # 2. Construcció de l'arquitectura

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=(227,227,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy',
optimizer=tf.optimizers.SGD(lr=0.001), metrics=['accuracy'])
model.summary()
```

**Snippet 5.x.x.x** – Font: <https://valueml.com/alexnet-implementation-in-tensorflow-using-python/>

The diagram illustrates the DenseNet architecture. It starts with an 'Input'  $x_0$  (represented by three red feature maps). This input is processed by a 'BN-ReLU-Conv' layer to produce  $H_1$  (three green feature maps).  $H_1$  is then processed by another 'BN-ReLU-Conv' layer to produce  $x_1$  (three purple feature maps). This pattern continues:  $x_1$  is processed by a 'BN-ReLU-Conv' layer to produce  $H_2$  (three yellow feature maps), and  $H_2$  is processed by a 'BN-ReLU-Conv' layer to produce  $x_2$  (three orange feature maps). Finally,  $x_2$  is processed by a 'BN-ReLU-Conv' layer to produce  $H_3$  (three brown feature maps), which is then passed through a 'Transition Layer' to produce the final output  $x_3$ . The diagram uses curved arrows to show the flow of data from the input through each stage of convolutional layers and transition layers.

The diagram illustrates a ResNet-like architecture for multi-class classification. The main flow is as follows:

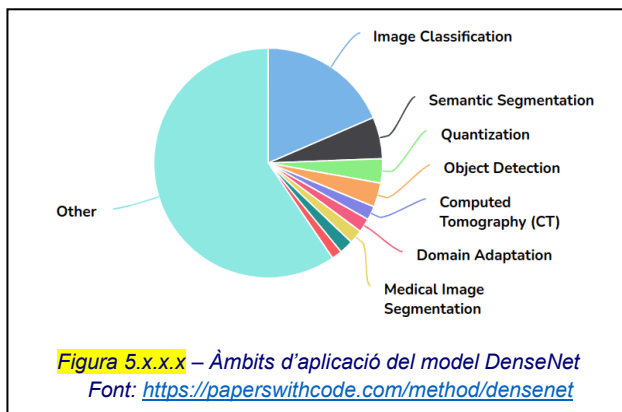
- Input** (red box) is processed by **Conv 7x7** (grey box) and **Pool 3x3** (grey box).
- The output is fed into **Dense Block 1** (grey box, containing 6 sub-blocks).
- A **Trans. 1** (teal box) residual connection bypasses Dense Block 1 and is added to its output.
- The result is fed into **Dense Block 2** (grey box, containing 12 sub-blocks).
- A **Trans. 2** (teal box) residual connection bypasses Dense Block 2 and is added to its output.
- The result is fed into **Dense Block 3** (grey box, containing 24 sub-blocks).
- A **Trans. 3** (teal box) residual connection bypasses Dense Block 3 and is added to its output.
- The result is fed into **Dense Block 4** (grey box, containing 16 sub-blocks).
- A **Trans. 4** (teal box) residual connection bypasses Dense Block 4 and is added to its output.
- The final output of the dense blocks is processed by **Pool 7x7** (grey box), **FC** (green box), and **Output** (yellow box).

The diagram also shows the internal structure of a dense block (bottom section):

- Batch Norm** (red box) is applied to the input.
- The output is passed through **ReLU** (grey box), **Conv 3x3** (grey box), **Drop Out** (grey box), and **Pool 2x2** (grey box) layers.
- The output of the dense block is then added to the input of the dense block via a residual connection (indicated by a plus sign).

G. Huang

<sup>20</sup> FER RECERCA AVIAM A QUE ES REFEREIX AIXÒ



Pel que fa les aplicacions del model, s'ha pogut demostrar que la característica diferencial de **reusabilitat de les característiques (features) de les dades** que aquesta arquitectura ofereix possibilita la seva aplicació en un ventall ampli d'àmbits que va més enllà de la classificació d'imatges<sup>21</sup>, com es pot veure a la **figura** de la esquerra.

## Implementació

```
# 1. En primer lloc cal importar les llibreries apropiades del framework
# TensorFlow per a Python: "tensorflow.keras.applications" per a importar
# el model DenseNet121, i "" per a importar altres capes que s'utilitzaran
# per a construir la xarxa (p.e. funcions de pre-processament d'imatges).
import tensorflow
import pandas as pd
import numpy as np
import os
import keras
import random
import cv2
import math
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from tensorflow.keras.layers import
Dense, GlobalAveragePooling2D, Convolution2D, BatchNormalization
from tensorflow.keras.layers import Flatten, MaxPooling2D, Dropout

from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input

from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array

from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

import warnings
warnings.filterwarnings("ignore")
```

<sup>21</sup> Font: <https://theaisummer.com/cnn-architectures/> → "Even though DenseNet was proposed for image classification, it has been used in various applications in domains where feature reusability is more crucial (i.e. segmentation and medical imaging application)."

```
# 2. Amb el bloc de codi següent implementem l'arquitectura del model DenseNet

model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(128, 128, 3))

x=model_d.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(8,activation='softmax')(x) #FC-Layer

# 3. De partida evitem entrenar tota la xarxa per a evitar "overfitting", tot #
ajustant la variable layer.trainable=False de manera que es desactiven
# totes les capes de l'arquitectura excepte les 8 darreres (que són les que
# permeten la detecció de taques i vores de les imatges).

for layer in model.layers[:-8]:
    layer.trainable=False

# 4. Un cop s'obtinguin resultats de predicció acceptables amb el model,
# aquest es pot acabar d'ajustar si tornem a activar la resta de capes per # a
l'entrenament amb layer.trainable=True.

"""
for layer in model.layers[:-8]:
    layer.trainable=True
"""

# 5. Si després d'aquest canvi compilem l'arquitectura i generem un resum,
# observarem una reducció notable de la quantitat de paràmetres que s'han
# de configurar a la xarxa:

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

A la [secció 5.6](#) es detalla la seqüència de construcció del model junt a les mostres dels dataset generats a la [secció 4](#) específics per a la realització d'aquesta PdC.

### 5.2.3 - Preparació de les dades per a l'entrenament del model

Abans de procedir a realitzar l'entrenament del model escollit a la secció anterior ([5.2.2 – Disseny d'un model base](#)), s'ha de:

- a) Pre-processar els dataset d'imatges “en cru o *raw*” que s'han generat com a resultat de les accions realitzades a la secció 4 d'aquest document
- b) Dividir els mateixos dataset en 3 grups: Entrenament, validació i explotació

Tot seguit es detalla en què consisteixen les accions de preparació anterior, així com la manera com s'aplicaran a les dades de què disposem per a la PdC.

#### 5.2.3.1 - Pre-processament de les imatges del dataset

El pre-processament i “neteja” de les dades que alimentaran el model de xarxes neuronals es un pas previ prescindible quan es tracta d'entrenar models DL, però important si es tracta de refinar al màxim la seva velocitat d'aprenentatge i rendiment. Per tant aplicarem les estratègies més habituals de preprocessament a les imatges dels dataset de la PdC.

##### **a) Imatges en color vs. escala de grisos**

Coneixent que per a representar una imatge en color RGB amb una estructura de dades calen tres matrius (una per a cada color base), mentre que una imatge en escala de grisos la podem representar amb una sola matriu, aleshores podem concloure que utilitzant imatges en escala de grisos “estalviarem” complexitat computacional a l'hora d'entrenar el model, ja que la quantitat de paràmetres que haurem de configurar serà molt menor (això és, els pesos de les d'arestes del graf que formen els nodes de les diferents capes ocultes de la xarxa neuronal profunda). En el cas d'aquesta PdC, com que ja estem utilitzant imatges que simulen visió IR tèrmica ([veure secció 3.x.x.x](#)) i aquestes estan convertides al B/N per a aconseguir una simulació més fidel, aleshores no haurem de prendre cap acció i simplement considerarem que les imatges ja han sigut pre-processades en aquest sentit. Cal notar que la utilització d'imatges en escala de grisos sempre serà convenient si es poden reconèixer les seves característiques utilitzant la visió humana, per tant en el cas dels dataset d'aquesta PdC clarament ens podrem beneficiar de l'estalvi en complexitat.

## b) Redimensionament de les imatges

Quan estem entrenant xarxes convolucionals profundes (DCNN), igual com passa amb les xarxes de perceptrons multicapa (MLP), hem de tenir cura de que totes les imatges dels dataset tinguin la mateixa mida, ja que la capa d'entrada (input Layer) de la xarxa neuronal sempre contindrà tants nodes com píxels conté la imatge. En el nostre cas, estem utilitzant imatges amb una resolució relativament alta (512x512), fet pel qual s'haurà de preprocessar cada imatge abans de ser utilitzada per a entrenar el model. Aquesta tasca es pot fer en temps real (*on-the-fly*) en el moment de carregar el dataset al model, mitjançant la llibreria Keras. Tanmateix, hem de tenir en compte que tant AlexNet com DenseNet tenen requeriments diferents de mida d'imatges (256x256 i 224x224<sup>22</sup> respectivament). Podem realitzar la implementació de cadascun dels casos de la següent manera:

### AlexNet

```
#Preprocessing: Image resizing function

def process_images(image, label):
    image = tf.image.per_image_standardization(image)
    image = tf.image.resize(image, (256,256))
    return image, label
```

**Snippet 5.x.x.x** – Pre-procés de la mida de les imatges (Implementació AlexNet) - Font: <https://learnopencv.com/understanding-alexnet/>

### DenseNet121

Per a la implementació del redimensionament d'imatges a DenseNet121 s'utilitzarà una estratègia diferent. Concretament, s'afegirà una nova capa "Lambda layer" per davant de la capa d'entrada o *input Layer*, la qual s'encarregarà de canviar la mida de cada imatge en temps real a en el moment d'entrenar el model:

```
# Preprocessing: Adding Lambda Layer that scales up the data to the correct size

model.add(K.layers.Lambda(lambda x:K.backend.resize_images(x,
height_factor=7,width_factor=7,data_format='channels_last')))
```

**Snippet 5.x.x.x** – Pre-procés de la mida de les imatges (Implementació DenseNet) - Font: <https://bouzouitina-hamdi.medium.com/transfer-learning-with-keras-using-densenet121-fffc6bb0c233>

<sup>22</sup> Font: <https://bouzouitina-hamdi.medium.com/transfer-learning-with-keras-using-densenet121-fffc6bb0c233>



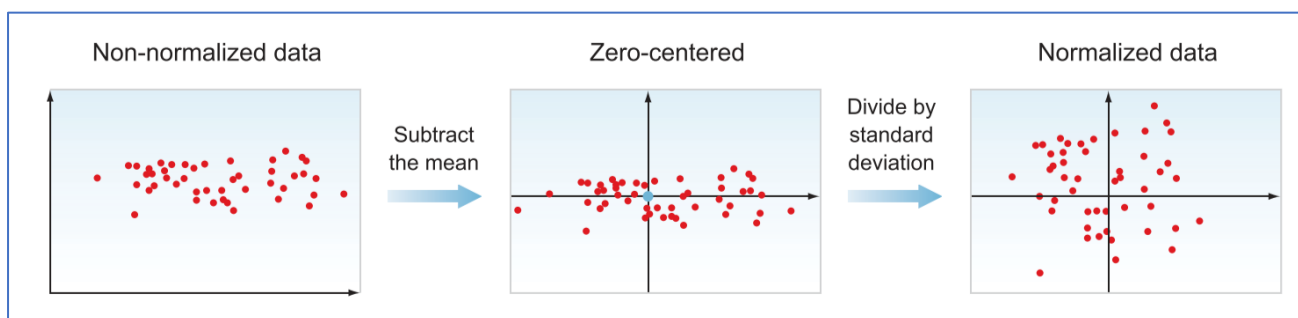
### c) Estandardització de les dades

La estandardització o normalització de dades en termes estadístics permet, donat un conjunt de mostres (en el cas que ens ocupa, entenem com a mostres els píxels d'una imatge que podem representar en una variable amb valors  $x_1, x_2, \dots, x_n$ ), localitzar amb la menor complexitat computacional temporal possible una dada en relació amb la seva mitjana, així com comparar els valors de les observacions de conjunts diferents, o dit d'una altra manera, comparar el valor dels píxels corresponents a imatges diferents<sup>23</sup>.

Per a calcular el valor estandarditzat o Z-Score d'una observació (píxel) donada la variable  $x = x_1, x_2, \dots, x_n$ , amb mitjana  $\bar{x}$  i desviació típica  $s_x$ , només cal restar la mitjana al valor de la observació i dividir-lo per la desviació típica, tal que:

$$x_i \xrightarrow{\text{Estandarditzar}} z_i = \frac{x_i - \bar{x}}{s_x}$$

D'aquesta manera, si estandarditzem les dades obtindrem com a resultat que el valor de tots els píxels de cada imatge es trobarà en el rang  $[0, 1]$  de valors, que a efectes pràctics ens permetrà accelerar el procés d'aprenentatge de la xarxa neuronal<sup>24</sup>. La següent figura mostra la distribució uniforme de les dades aconseguida un cop realitzat el procés d'estandardització:



*Imatge obtinguda de [Ref. Llibre DL], pàg. 155*

<sup>23</sup> Referències bibliogràfiques: Pàgs. 44-46, materials de l'assignatura "Estadística", mòdul "Estadística descriptiva: Introducció a l'anàlisi de dades", Àngel J. Gil Estallo (UOC)

<sup>24</sup> P. 154 Llibre DL → "Although not required, it is preferred to normalize the pixel values to the range of 0 to 1 to boost learning performance and make the network converge faster."

### Implementació (AlexNet i DenseNet)<sup>25</sup>.

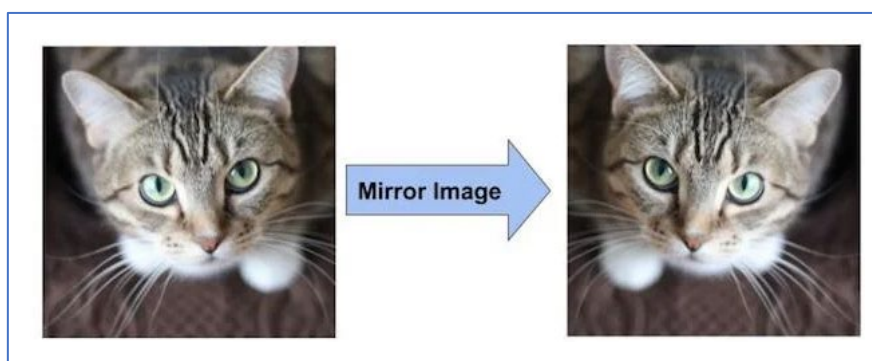
Podem implementar la normalització dels dataset de la PdC de manera conjunta per lots mitjançant la llibreria de funcions d'estandarització que implementa el framework de ML *Keras* per a Python+TensorFlow:

```
from keras.layers.normalization import BatchNormalization  
  
[...]  
  
model.add(BatchNormalization())
```

Concretament, caldrà afegir una capa d'estandarització per lots justament **després** de cadascuna de les capes de la xarxa neuronal, de manera que la següent capa podrà rebre d'entrada una imatge ja estandaritzada.

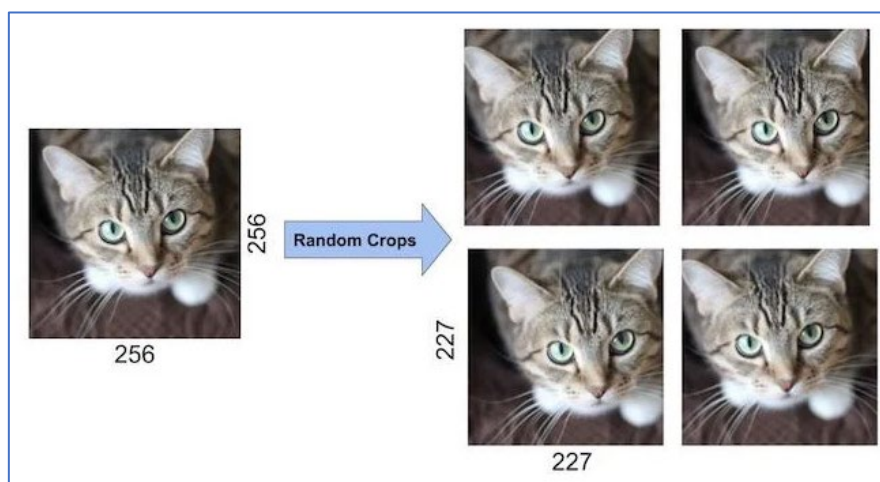
#### **d) Augmentació de dades (data augmentation)**

Es tracta d'un conjunt de tècniques que permet ampliar la quantitat d'imatges dels dataset (i per tant incrementar el rati d'aprenentatge del model) simplement aplicant-hi certes modificacions per a generar diferents variacions. A més, està demostrat que ampliant el catàleg d'imatges amb còpies lleugerament modificades s'augmenta la precisió de les prediccions realitzades pel model i es redueix la probabilitat de que es produeixi *overfitting*<sup>26</sup>, és a dir, que s'acabi "sobre-entrenant" el model fent que aquest perdi la seva capacitat de generalització amb imatges que no formin part del dataset d'entrenament. Alguns exemples de tècniques utilitzades per a entrenar alguns models coneguts (p.e. AlexNet) són el "mirroring", o el "random cropping", tal i com es mostra a continuació:



<sup>25</sup> Informació obtinguda de [P184 Llibre DL]

<sup>26</sup> <https://www.unite.ai/what-is-overfitting/>

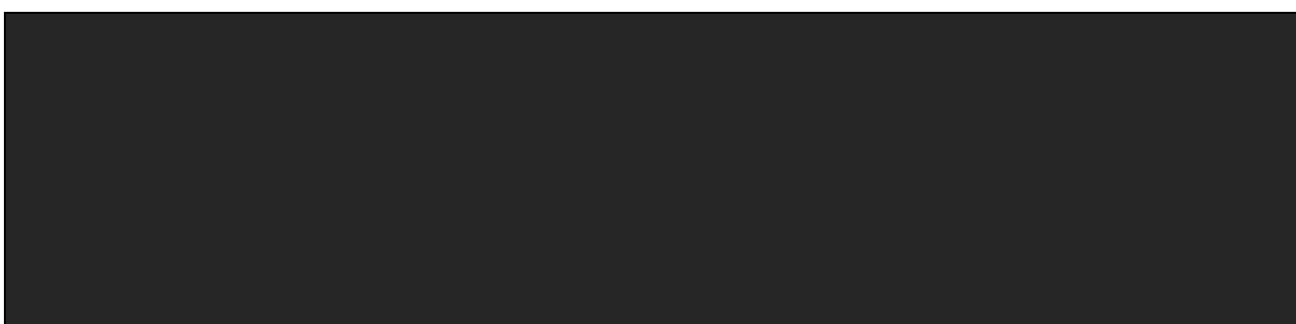


Imatges obtingudes de <https://learnopencv.com/understanding-alexnet/>

Cal fer notar que existeixen altres tècniques semblants, com la rotació d'imatges, l'apropament o "zoom", etc, i que a més és possible utilitzar combinacions de diferents tècniques per a ampliar encara més els dataset d'imatges.

Per a aplicar les tècniques esmentades d'augmentació d'imatges als dataset d'aquesta PdC podem utilitzar diferents llibreries, per exemple, la implementació que proporciona Keras per a TensorFlow, o bé la llibreria *Augmentor*<sup>27</sup> per a Python. Escollirem la primera degut a que permet realitzar l'augmentació de la col·lecció d'imatges en temps real en el moment d'entrenar el model<sup>28</sup>.

### Implementació (AlexNet):



Snippet 5.x.x.x – Font:

<sup>27</sup> <https://augmentor.readthedocs.io/en/master/>

<sup>28</sup> Pel que fa les tasques de pre-processament, normalment és habitual realitzar-les o bé a la entrada del model com en el cas del redimensionament, o bé en etapes intermèdies, com passa amb la estandardització

### Implementació (DenseNet):



**Snippet 5.x.x.x – Font:**

### 5.2.3.2 - Divisió dels dataset en grups

Un cop s'hagin aplicat les diferents tècniques de pre-processament d'imatges als dataset per tal d'adaptar-los als requeriments de l'arquitectura DCNN, el que haurem de fer és dividir cada dataset en tres grups o subconjunts diferents;

- a) Grup d'entrenament (*training dataset*) → 75% - 80%
- b) Grup de validació (*validation dataset*) → 15% - 20%
- c) Grup d'explotació (*test dataset*) → 15% - 20%

**Bloc XX:** Divisió dels dataset i la seva proporció – Font: [LLIBRE DL P152]

De manera que utilitzarem el subconjunt d'entrenament a) per a entrenar el model tot ajustant el pes de les arestes que connecten les capes ocultes (hidden layers) de la DCNN; el subconjunt de validació b) per a contrastar el rendiment del model **durant l'entrenament** (de manera periòdica, cada certes *epoch* o iteracions d'entrenament) contra un subconjunt de mostres “desconegut” per al model per a comprovar el nivell de generalització del coneixement adquirit pel model durant la etapa d'entrenament i realitzar els ajustos corresponents als pesos de les arestes per a millorar el seu rendiment; i finalment, el subconjunt d'explotació c) ens servirà com a “prova de camp” per a avaluar el rendiment final del model **un cop completat l'entrenament**.

Pel que fa la proporció de mostres dels dataset que s'assignen a cada subconjunt tindrem en compte que, per norma general, si el dataset està comprés per una quantitat relativament gran de mostres (de l'ordre dels milions) aleshores amb un 1% de les mostres per als subconjunts de validació i d'explotació ja n'hi haurà prou, de manera que reservarem tota la resta per a efectuar l'entrenament i així aconseguir una major precisió de les prediccions del model. En el cas de la PdC que estem desenvolupant però, els dataset contindran una quantitat d'imatges molt inferior (de l'ordre dels centenars o milers), de manera que els ratis habitualment utilitzats per a l'entrenament de models ML seran més convenients (veure **bloc XX**).

Implementació (AlexNet i DenseNet):

Snippet 5.x.x.x – Font: <https://valueml.com/alexnet-implementation-in-tensorflow-using-python/>

## **5.3 - Avaluació del model i interpretació del seu rendiment**

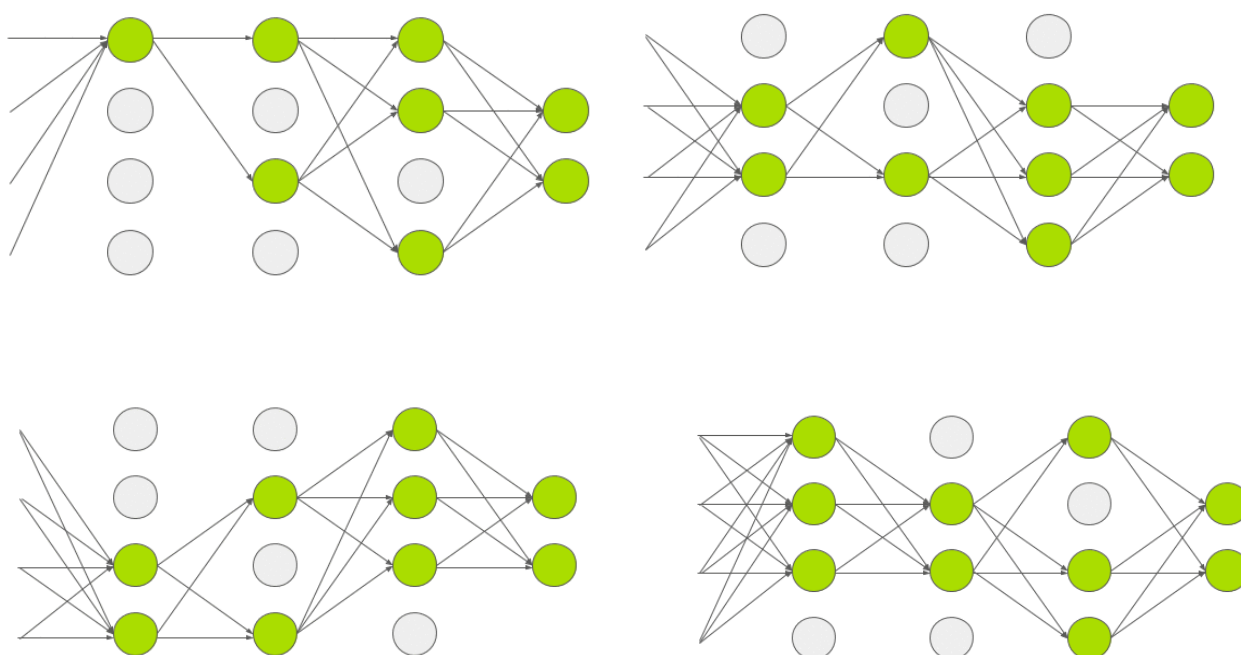
TO-DO

## 5.4 – Milllores en el rendiment de la xarxa neuronal i reajustament d'hiper-paràmetres

TO-DO

### 5.4.x. DROPOUT REGULARIZATION

<https://learnopencv.com/understanding-alexnet/>





## **5.5 – Estudi del rati d'aprenentatge i optimització dels paràmetres**

TO-DO

## 5.6 – Wrapping-Up: Adaptació i implementació de les architectures DCNN escollides

Un cop establerts a les seccions anteriors els paràmetres inicials amb els quals crearem les arquitectures de xarxa neuronal que utilitzarem en aquesta PdC, podem procedir a la seva implementació. Es seguiran els següents passos, tant en el cas de la implementació de AlexNet com en el de DenseNet:

- 1 Import the dependencies.
- 2 Get the data ready for training:
  - Download the data from the Keras library.
  - Split it into train, validate, and test datasets.
  - Normalize the data.
  - One-hot encode the labels.
- 3 Build the model architecture. ~~In addition to regular convolutional and pooling layers, as in chapter 3, we add the following layers to our architecture:~~
  - ~~– Deeper neural network to increase learning capacity~~
  - ~~– Dropout layers~~
  - ~~– L2 regularization to our convolutional layers~~
  - ~~– Batch normalization layers~~
- 4 Train the model.
- 5 Evaluate the model.
- 6 Plot the learning curve.

P185 LLIBRE DL

TO-DO: Fer implementacions a la repo del github i posar aquí l'enllaç i/o copiar/enganxar codi