Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Daniel Farré Serra

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: **Creació d'una Xarxa Neuronal Convolucional] diagnosi de pacients amb pneumònies**

Director/a: **Jordi Planes Cid**

Presentació

Mes: Juny

Any: 2022

Índex

1. Introducció

En l'última dècada l'avenç en el camp de la intel·ligència artificial és realment notable, tant és així que està present en el nostre dia a dia, ja sigui en els nostres telèfons intel·ligents capaços de reconèixer els nostres gustos per a recomanar-nos productes personalitzats, dispositius amb assistents de veu amb els que podem interaccionar per facilitar-nos la nostra vida personal, i inclòs en els últims models de vehicles que ja comencen a conduir de forma autònoma.

Tot això pel que fa al gran públic, però existeix una gran part d'investigació i indústria on s'estan invertint milions d'euros en aquest sector, tant empreses privades amb els seus propis productes, com investigacions públiques amb finançament. Nosaltres ens fixarem en productes enfocats en la medicina, ja que s'estan creant fites amb molt potencial que estan revolucionant les eines que disposen el personal sanitari.

Per donar-ne un exemple, actualment un producte el qual està donant molt de què parlar és "Alpha Fold" el qual és un programa d'intel·ligència artificial dissenyat per DeepMind, aquest és capaç de predir, a partir d'una cadena d'aminoàcids, les estructures tridimensionals de les proteïnes amb una precisió molt elevada. Aquest problema, dibuixar tridimensionalment l'estructura d'una proteïna, fins ara era un problema molt complicat el qual requeria un cost i temps computacionals molt elevats amb els mètodes tradicionals, però "Alpha Fold" és capaç de fer-ho en temps rècords, donant la possibilitat de poder crear nous fàrmacs.

Aquest treball ve motivat per aquest producte de DeepMind i molts altres productes d'altres empreses que intenten produir noves tecnologies mèdiques potenciades amb intel·ligència artificial.

1.1. Objectius

- Estudiar i comprendre la teoria que hi ha al darrere de les xarxes neuronals.
- Estudiar i comprendre la teoria que hi ha al darrere de les xarxes neuronals convolucionals.
- Estudiar les pneumònies i veure com identificar-les en radiografies.
- Fer el disseny de diferents xarxes neuronals convolucionals, amb l'objectiu d'identificar pacients amb pneumònia, rebent com a entrada imatges de radiografies.
- Fer la implementació d'aquestes xarxes neuronals, experimentant amb les diferents implementacions i arquitectures.
- Entrenar les diferents xarxes neuronals amb radiografies etiquetades.
- Buscar la millor arquitectura i implementació, i aconseguir tasses d'encert i precisió de les decisions, tan elevats com sigui possible.
- Analitzar, estudiar i comprendre els resultats obtinguts de les diferents implementacions.

Tots els experiments es troben en el repositori públic del projecte [1].

2. Marc teòric

2.1. Xarxes neuronals

Una xarxa neuronal artificial [2] és un model computacional d'aprenentatge automàtic, el qual té la capacitat d'aprendre i millorar el seu resultat analitzant una gran quantitat d'exemples d'entrenament. L'objectiu d'aquest tipus de models és intentar simular el sistema nerviós humà.

El nostre sistema nerviós conté unes 100 mil milions de neurones, això ens ha permès evolucionar com a espècie i desenvolupar la nostra intel·ligència amb la qual som capaços de conviure en grans ciutats, fer grans descobriments científics, tecnològics i fins i tot hem pogut escapar de la Terra. Si ens fixem en el sistema nerviós d'altres animals, com el ximpanzé i el d'un gat, que tenen respectivament 6 mil milions i 300 milions de neurones, podem pensar que al tenir un major nombre de neurones, nosaltres som capaços de resoldre problemes més complicats i tenir una major intel·ligència.

La idea de les xarxes neuronals és recrear tecnològicament aquest mateix concepte, un sistema nerviós que conté grans quantitats de neurones amb les quals és capaç de resoldre problemes molt complexos. Posem com a exemple una xarxa neuronal que ha de trobar patrons en una imatge, una amb un nombre reduït de neurones podrà aprendre a trobar patrons simples que una persona qualsevol podria trobar sense cap dificultat, ara bé, si construïm una xarxa neuronal que disposa d'una gran quantitat de neurones, aquesta serà capaç d'aprendre d'una forma molt més profunda i de distingir patrons molt complexos, patrons que una persona no podria veure a simple vista.

En la [Figura 1] hi ha una representació gràfica d'una xarxa neuronal artificial. En aquest apartat veurem al detall com funciona i cada una de les seves parts, des de les diferents capes que la componen, com funciona una neurona individualment i com és capaç d'aprendre una xarxa neuronal.

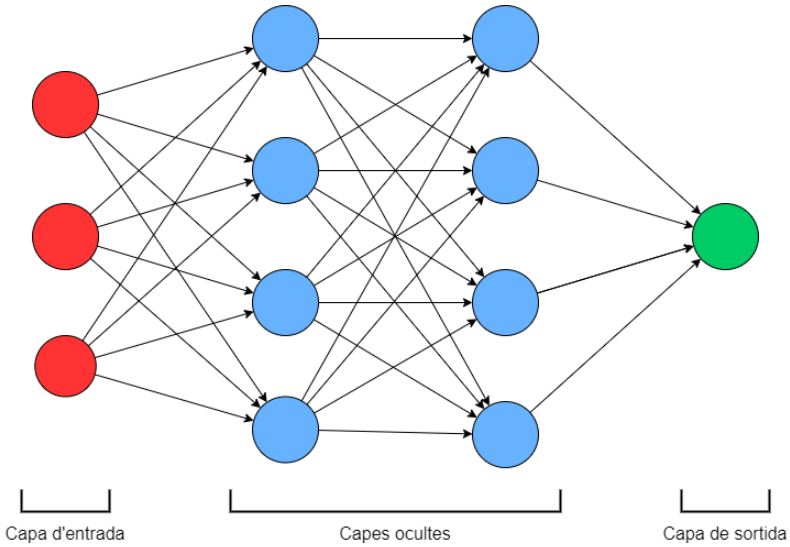


Figura 1: Esquema bàsic d'una xarxa neuronal

2.1.1. Capes

Una xarxa neuronal està formada per capes: capa d'entrada, capes ocultes i capa de sortida. En la [Figura 2] hi ha un esquema on es mostra com estan distribuïdes. És important entendre les parts que constitueixen una xarxa neuronal i les responsabilitats que té cada una d'elles.

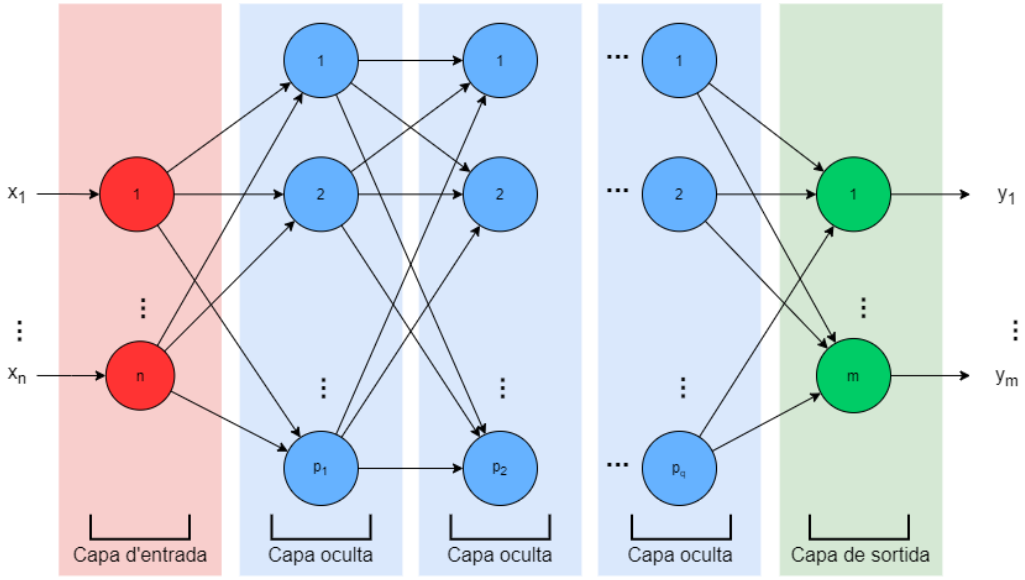


Figura 2: Capes d'una xarxa neuronal

Anem a veure una a una cada capa:

- **Capa d'entrada**

  Aquesta primera capa representa els paràmetres d'entrada, que poden ser d'1 a n. No existeix cap mena de càlcul, solament són els inputs que passaran a una capa oculta.

  Per posar un exemple, si una xarxa neuronal ha de calcular el preu de venda d'un pis, les entrades podrien ser: metres quadrats, distància al centre de la ciutat, zona, antiguitat, etc.

- **Capes ocultes**

  Les capes ocultes no tenen una connexió directa amb el món exterior i és on es duen a terme la major part dels càlculs computacionals.

  En una xarxa neuronal, poden existir cap o moltes capes ocultes, i cada una d'aquesta encapsula un nombre *n* de neurones. Aquest conjunt de capes representa el comportament de la xarxa neuronal.

  Relacionat amb les capes ocultes, diem que la profunditat d'una xarxa neuronal depèn directament del nombre de capes ocultes que té, una xarxa neuronal amb poques capes ocultes serà poc profunda i a l'inrevés serà molt profunda.

- **Capa de sortida**

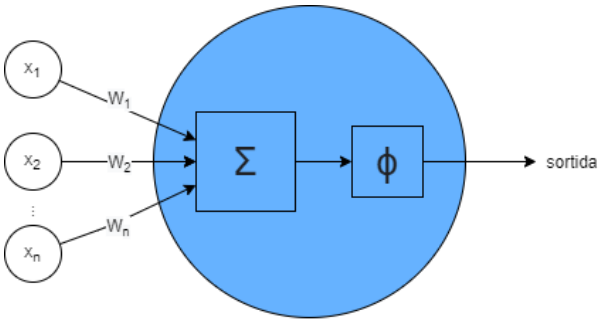
  La capa de sortida és molt similar a una capa oculta, ja que també està formada per neurones que fan càlculs computacionals, amb la diferència que el nombre de neurones que hi ha en aquesta, equival al nombre de sortides que té la xarxa neuronal perquè els resultats d'aquestes neurones seran els resultats finals.

  Seguint amb l'exemple anterior, la xarxa neuronal en l'última capa tindria una neurona que donaria com a sortida el preu de venda del pis.

2.1.2. Neurones

Com ja hem dit en la introducció de les xarxes neuronals, aquestes intenten imitar un cervell humà en l'àmbit neuronal. Una neurona d'una xarxa neuronal artificial [3] intenta modelar les

neurones que constitueixen el sistema nerviós. En la [Figura 3] tenim un esquema del funcionament individual d'una neurona.



x_i Valor de l'entrada i-èssima (input)
W_i Pes entre la connexió de la i-èssima entrada i la neurona
Σ Sumatori
ϕ Funció no lineal

Figura 3: Neurona de McCulloch-Pitts

La sortida d'una neurona és el resultat de realitzar una suma ponderada de les entrades, seguit de l'aplicació d'una funció no lineal.

La [Equació 1] representa el càlcul realitzat per una neurona:

$$\phi\left(\sum_{i=1}^{n} w_i x_i\right)$$

Equació 1: Càlcul d'una neurona

Explicat amb més paraules, a una neurona li arriben un nombre *n* de dades *x* a través d'unes connexions, cada una de les quals té un pes *w* i es fa un sumatori ponderat entre les entrades i els seus pesos. Com a petit avanç, la xarxa neuronal haurà d'anar ajustant aquests pesos per a poder aprendre i aconseguir resultats òptims. En tenir el resultat del sumatori s'aplica una funció no lineal o lineal, anomenades funcions d'activació, que veurem més en profunditat en el següent apartat, però com a petit resum, una funció d'activació equival a un filtre on depenent de les entrades la neurona s'activarà o no.

Aquest comportament és el mateix que el de les nostres neurones transportant impulsos elèctrics, nosaltres quan rebem un cop som capaços de distingir si ens produeix dolor o no, gràcies als receptors de dolor que filtren la sensació que ens produeix en funció de la força en la qual el rebem.

2.1.3. Funcions d'activació

En aquest apartat veurem que són les funcions d'activació [4], com funcionen i les situacions en les quals s'utilitzen. Concretament, veurem les quatre funcions més conegudes i usades pels desenvolupadors de xarxes neuronals, però a part d'aquestes, n'existeixen molts més amb diferents propòsits.

- **Funció esglaó (threshold)**

  Aquesta primera funció d'activació és de les més bàsiques i rígides que existeixen, ja que, per a un valor x, la sortida serà 0 o 1 (passa o no passa el filtre).

  La funció esglaó s'utilitza en les neurones de la capa de sortida en problemes que requereixen una classificació molt estricta, per exemple determinar si en una imatge hi ha

un cotxe o no. Però es pot veure fàcilment que si el nostre objectiu és que la xarxa neuronal ens doni un percentatge que ens indiqui el segur que està que a la imatge apareix un cotxe o no, aquesta funció no és una bona opció.

En la [Figura 4] i la [Equació 1] podem veure la funció.

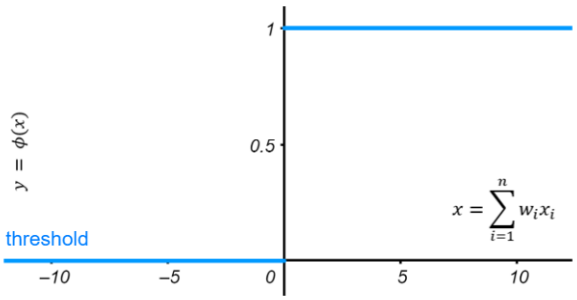


Figura 4: Funció esglaó

$$\phi(x) = \begin{cases} 0, si\ x < 0 \\ 1, si\ x \geq 0 \end{cases}$$

Equació 2: Funció esglaó

- **Funció sigmoide**

  La funció sigmoide és similar a la funció esglaó, però en aquest cas és una funció lineal la qual té una pujada gradual, en comptes de donar una sortida binària, aquesta donarà un valor d'entre el 0 i l'1.

  Aquesta funció és útil en la capa de sortida quan volem fer prediccions probabilístiques. També s'utilitza molt per a fer classificacions més precises, ja que la neurona pot fer una anàlisi més detallat a causa del rang.

  Podem veure aquesta funció en la [Figura 5] i la [Equació 3].

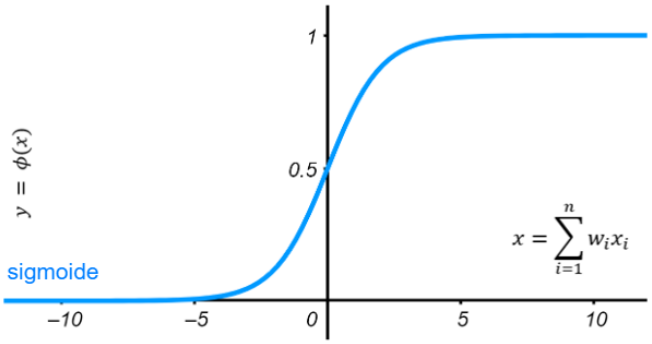


Figura 5: Funció sigmoide

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Equació 3: Funció sigmoide

- **Funció tangent hiperbòlica (tanh)**

  La funció tanh, és molt similar a la funció sigmoide, però aquesta compren el rang d'entre -1 a 1.

Aquesta funció s'utilitza de la mateixa forma que la funció sigmoide, ja que són molt similars, això no obstant, és prou diferent com per a utilitzar-la en certes situacions. L'avantatge d'aquesta funció és que no es perden tantes activacions a diferència de la sigmoide sinó que existeix un rang major, això ens permet fer classificacions molt precises. Ara bé, aquesta funció comporta un cost computacional més elevat que les altres funcions a causa del seu rang.

Podem veure aquesta funció en la [Figura 6] i la [Equació 4].

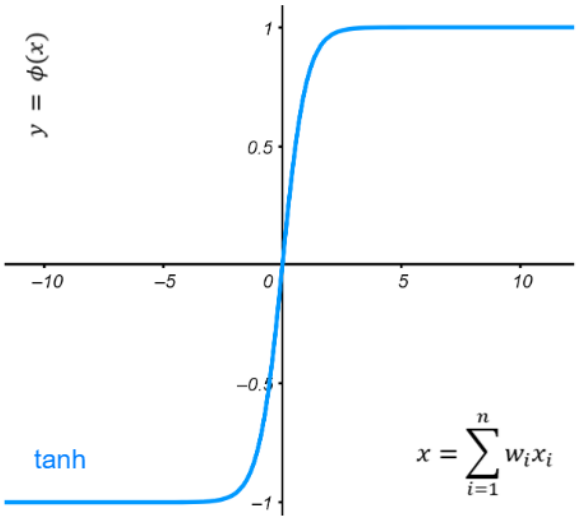


Figura 6: Funció tangent hiperbòlica

$$\phi(x) = \frac{2}{1 + e^{-2x}} - 1$$

Equació 4 : Funció tangent hiperbòlica

- **Funció rectificadora (ReLU)**

  La funció ReLU s'utilitza per a depreciar valors negatius, ja que donada una entrada x la seva sortida estarà compresa entre el rang de 0 a infinit.

  Aquesta funció d'activació és la més coneguda i usada en el camp de la intel·ligència artificial per la seva simplicitat i utilitat, sobretot en les xarxes neuronals per a la classificació d'imatges. El seu comportament és similar al de la funció sigmoide, però computacionalment té un cost més reduït, sobretot, en xarxes neuronals molt profundes que fan servir aquesta funció en diferents capes. Aquest baix cost computacional és degut al fet que la funció fa una discriminació molt agressiva que causa un nombre més reduït d'activacions en la xarxa neuronal, això es tradueix en el fet que la xarxa neuronal duu a terme menys càlculs que podrien ser innecessaris.

  Podem veure aquesta funció en la [Figura 7] i la [Equació 5].

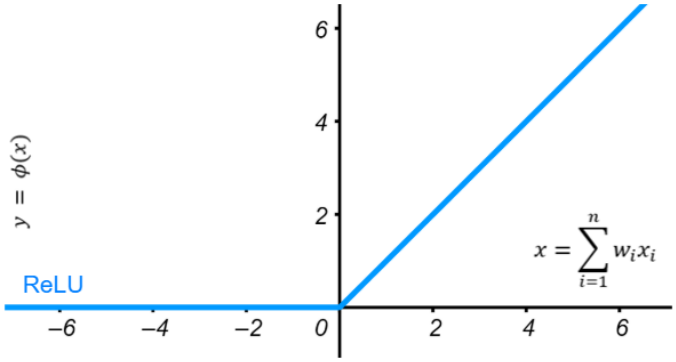


Figura 7: Funció rectificadora

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

Equació 5: Funció rectificadora

Ara que coneixem el funcionament de les funcions d'activació i que existeix d'un ampli ventall d'opcions, és normal fer-se les següents preguntes. Quina funció d'activació és millor? Quan he d'utilitzar una o una altra? Doncs la resposta a aquestes preguntes és que, depèn del problema que hagi de resoldre la xarxa neuronal, depèn de si prefereixes rendiment per davant de precisió o a l'inrevés, depèn dels exemples que empres per fer l'entrenament, etc. En definitiva, no hi ha una funció que sigui la millor, s'han de provar les opcions i seleccionar la que sigui millor per aquell problema.

2.1.4. Neurona de biaix

La neurona de biaix en una xarxa neuronal sol ser una part fonamental per aconseguir un model d'aprenentatge que doni bons resultats. Aquesta neurona permet desplaçar la funció d'activació, cap a l'esquerra o cap a la dreta de les neurones de la capa posterior. La xarxa neuronal ajustarà aquests valors de sortida de les neurones de biaix, que anomenarem valor de biaix, permetent que els resultats i l'aprenentatge de la xarxa neuronal siguin molt més flexibles.

En la [Figura 8] podem veure un exemple de com es desplacen les funcions d'activació.

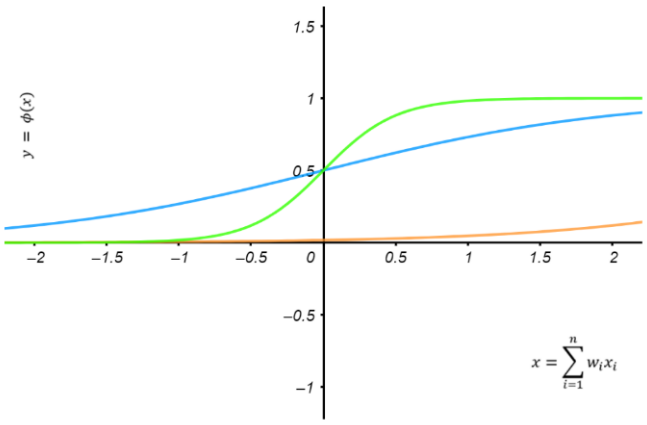


Figura 8: Funcions sigmoides amb biaix

L'objectiu d'aquest node és evitar un biaix en la xarxa neuronal, desplaçant les funcions d'activació permet que algunes neurones siguin capaces d'activar-se o donar resultats diferents. Normalment, aquests nodes es col·loquen un per cada capa oculta de la xarxa neuronal. Això no significa que sempre donin millors resultats, com passava amb les funcions d'activació, hi haurà ocasions que ajudaran a treure millors resultats i en altres no tant.

L'equació [Equació 1] que coneixíem per calcular l'activació d'una neurona ha de canviar afegint el valor de biaix, i a partir d'ara anomenarem *z* el sumatori ponderat amb biaix, [Equació 6]:

$$\phi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \phi(z)$$

Equació 6: Activació d'una neurona amb biaix

2.1.5. Algoritme de retropropagació

Per acabar amb les xarxes neuronals, anem a veure com aquestes tenen la capacitat d'aprendre amb l'algorisme de retropropagació [5], però abans, necessitem conèixer una mica la nomenclatura que utilitzarem per denotar les posicions de les neurones, els pesos i les sortides.

Per referir-nos a un pes entre la connexió d'una neurona en la posició *k* de la capa (*l* - 1) a la neurona en la posició *j* de la capa *l*, ho farem de la següent forma: w_{jk}^{l}, en la [Figura 9] podem veure un exemple del pes de la 4a neurona en la capa (*l* – 1) a la 2a neurona de la capa *l*.

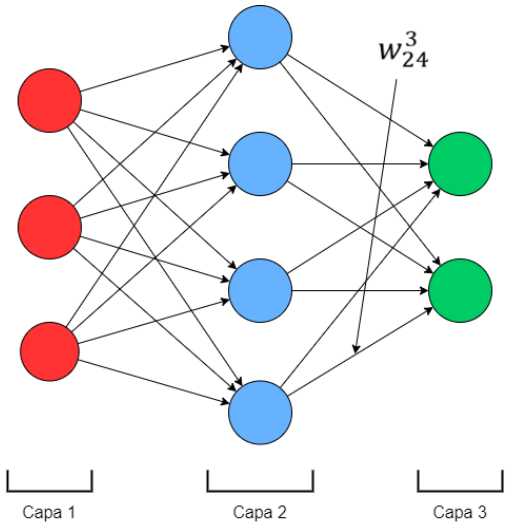


Figura 9: Pes entre la connexió de dues neurones

Per a les activacions i el biaix, utilitzarem una notació molt similar: utilitzarem b_j^l per indicar el biaix de la neurona *j* de la capa *l*, i de la mateixa forma, utilitzarem a_j^l per indicar l'activació de la neurona *j* de la capa *l*. En la [Figura 10] podem veure un exemple.

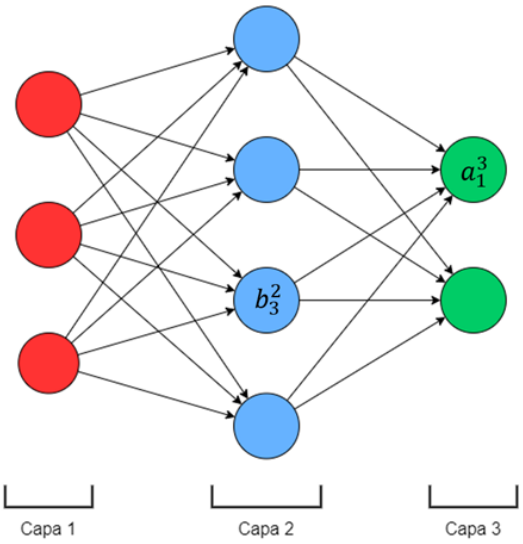


Figura 10: Anotació de biaix i activació

Amb aquesta notació, podem reescriure la fórmula d'activació, que ja coneixíem, [Equació 6], d'una neurona en la posició *j* de la capa *l,* [Equació 7]:

$$a_j^l = \phi\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Equació 7: Equació d'activació

2.1.5.1. Càlcul de la sortida d'una xarxa neuronal basat en matrius

Tenint l'equació [Equació 7], l'objectiu és convertir-la en una altra expressió equivalent simplificada que representarà operacions amb matrius.

Per fer-ho definirem una matriu de pes w^l, aquesta matriu representa els pesos que es connecten a la capa *l*, és a dir, la posició de la fila *j* i columna *k*, és w_{jk}^l. De la mateixa forma, per a cada capa definim un vector de biaix b^l, que funciona de la mateixa forma, cada posició *j* representa b_j^l. I per últim, també definirem un vector d'activació a^l, que funcionarà igual al vector de biaix a_j^l.

L'últim que necessitem per reescriure l'equació [Equació 7] en forma matricial, és la idea de la vectorització de funcions, que, resumidament, consisteix a aplicar una funció per cada un dels elements d'un vector.

Tenint tots els ingredients necessaris, ja podem construir l'equació [Equació 8] en forma vectoritzada:

$$a^l = \phi(w^l a^{l-i} + b^l)$$

Equació 8: Equació d'activació en forma matricial

Si mirem amb deteniment aquesta funció, tenim que, la matriu d'activacions en la capa *l* a^l, és el resultat d'aplicar la funció d'activació sobre la matriu resultant de multiplicar la matriu de pesos de la capa *l* w^lper la matriu d'activacions en la capa *l – 1* a^{l-i}, sumant el vector de biaixos de la capa *l* b^l.

Aquesta expressió ens dona una visió més general de com les activacions d'una capa es relacionen amb la capa anterior aplicant la matriu de pesos a les activacions, després afegim el vector de biaix i en acabar apliquem la funció d'activació.

Cal destacar que aquesta equació és més important del que sembla a primera vista, a part que queda molt simplificada sense *j* i *k.* El fet de poder fer inferència i entrenar a les xarxes neuronals fent operacions amb matrius és una molt bona notícia, perquè les API's que s'utilitzen per implementar algorismes d'intel·ligència artificial estan preparades per treballar amb matrius. Això és degut al fet que des de fa un temps s'estan emprant targetes gràfiques per entrenar models d'intel·ligència artificial, perquè tenen un gran poder computacional fent operacions amb matrius i permeten entrenar xarxes neuronals molt profundes en un temps raonable.

2.1.5.2. Funció de cost

La funció de cost ens permet comparar el resultat de sortida de la xarxa neuronal amb el resultat que nosaltres esperem. El resultat d'aplicar aquesta funció és la diferència que hi ha entre el valor obtingut i l'esperat, i amb aquest resultat podem fer correccions a la xarxa neuronal per aconseguir un resultat més proper a l'esperat.

L'objectiu de l'algorisme de la retropropagació, és veure com canvia el resultat de la funció de cost, respecte al canvi de qualsevol pes o biaix. L'exemple més conegut que s'utilitza com a funció de cost, és la funció de cost quadràtic, [Equació 9].

$$C = \frac{1}{2}\sum_{x}(y(x) - a^{L}(x))^{2}$$

Equació 9: Funció de cost quadràtic

On, la suma *x* és sobre els exemples d'entrenament individual, *y(x)* és el vector de sortida desitjada i $a^{L}(x)$ és el vector de sortida d'activacions de la capa *L*.

2.1.5.3. Algorisme del gradient descendent

Ara que ja sabem com comparar el resultat obtingut amb l'esperat utilitzant la funció de cost, la pregunta és, com puc aconseguir reduir el resultat d'aquesta funció el màxim possible? Doncs una de les solucions que existeixen, és l'algorisme del gradient descendent que el podem definir com un algorisme iteratiu d'optimització de primer ordre per a trobar el mínim d'una funció.

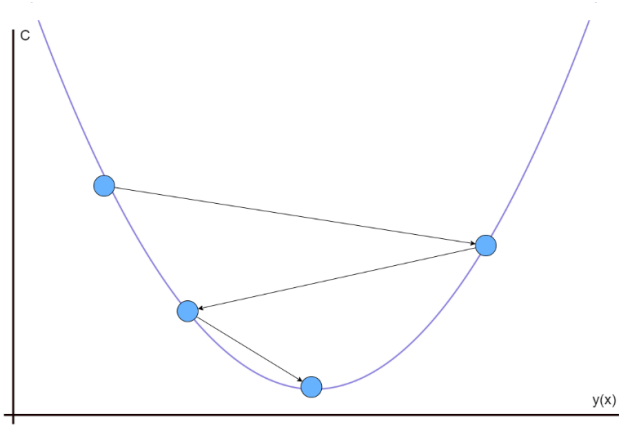


Figura 11: Algorisme del gradient descendent

En la figura [Figura 11] podem veure un petit exemple de com seria el funcionament d'aquest algoritme amb la funció de cost quadràtic:

- L'algoritme comença en un punt qualsevol de la funció *C* i es calcula el pendent de la recta tangent en el punt (gradient).
- Si el pendent és positiva, s'ha d'anar a la dreta i si és negativa s'ha d'anar a l'esquerra com es veu en la figura, i es torna a calcular el pendent en el nou punt.
- Repetir el pas anterior fins que es trobi el punt òptim de la funció *C.*

Tenint la funció de cost quadràtic, podríem intentar trobar el punt mínim de la funció utilitzant força bruta com en la figura [Figura 11], però en treballar amb xarxes neuronals, la funció que hi ha en la figura no és tan senzilla. Les funcions que s'usen en xarxes neuronals són funcions multidimensionals, això és causat per la gran quantitat de connexions que hi ha entre les neurones, també se sumen les funcions d'activació el qual fa que les funcions de cost acabin tenint formes irregulars, tot això fa que computacionalment sigui molt difícil aconseguir el punt òptim en xarxes neuronals molt grans. L'exemple d'una funció de cost que ja comença a ser més complicat de trobar el punt mínim, podria ser la figura [Figura 12].

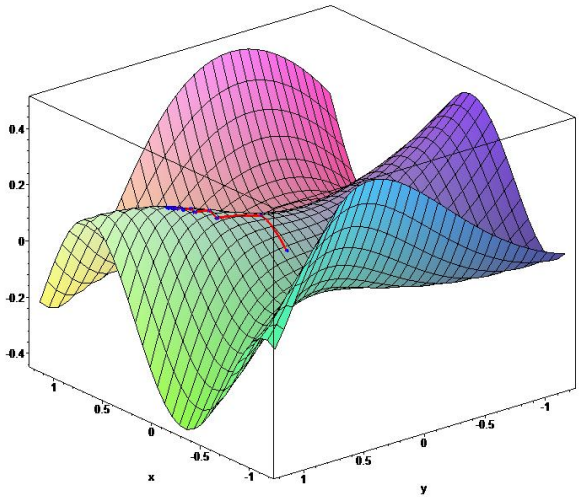


Figura 12: Pla tridimensional

La solució a aquest problema està a aplicar l'algoritme del descens del gradient unit amb l'algorisme de la retropropagació, el qual ens donarà com a resultat el gradient de la funció de cost, simplificant aquesta multidimensionalitat que genera funcions realment complexes.

2.1.5.4. Les quatre equacions fonamentals

Com hem vist fins ara, la retropropagació busca modificar els pesos i els biaixos amb l'objectiu de minimitzar l'error que ens dona la funció de cost.

Per poder saber com varia la funció de cost *C* respecte als pesos *w* i als biaixos *b* ho fem mitjançant variables parcials, en el cas de la variació amb els pesos $\partial C/\partial w$ i per als biaixos $\partial C/\partial b$. Sabent això, podem definir l'error d'una neurona *j* de la capa *l* com δ_j^l, [Equació 10].

$$\delta_j^L = \frac{\partial C}{\partial z_j^l}$$

Equació 10: Error d'una neurona

Aquest valor ens indica la implicació que ha tingut la neurona *j* de la capa *l* en l'error del resultat final.

Però per aplicar l'algorisme de retropropagació no és tan senzill com anar calculant les variables parcials de cada pes i biaix, ja que d'alguna forma hem de propagar els errors de les neurones entre les capes. Si no ho fem no estaríem tenint en compte que l'error que produeix una neurona és per culpa d'una neurona anterior que li ha passat aquest error. Per propagar aquest error d'una forma eficient, necessitarem quatre fórmules fonamentals:

1. Error en la capa de sortida

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\phi'(z_j^L)$$

Equació 11: Error en la capa de sortida

Aquesta equació [Equació 11], ens dona com a resultat l'error de la neurona *j* en l'última capa. El primer terme $\partial C / \partial a_j^L$ representa la velocitat en la qual canvia la funció de cost amb l'activació de la neurona *j* de la sortida, el segon terme $\phi'(z_j^L)$ mesura la velocitat en la qual canvia la funció d'activació en z_j^L.

Aquesta equació [Equació 12] en forma matricial quedaria de la següent forma:

$$\delta^L = \nabla_a C \odot \phi'(z^L)$$

Equació 12: Error en la capa de sortida en forma matricial

2. Error $\boldsymbol{\delta^l}$ en termes d'error en la següent capa $\boldsymbol{\delta^{l+1}}$

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \phi'(z^l)$$

Equació 13: Error propagat

Aquesta equació [Equació 13] ens dona l'error en la capa *l*, propagat de la capa *l + 1*. Obtenim aquest resultat ja que tenim l'error δ^{l+1} i apliquem la transposició $(w^{l+1})^T$, això es pot veure com si desplacéssim l'error a través de la xarxa. Després tenim el producte de Hadamard $\odot\ \phi'(z^l)$, que mou l'error enrere a traves de la funció d'activació en la capa *l*.

3. Taxa de canvi del cost, respecte a qualsevol biaix de la xarxa [Equació 14]

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Equació 14: Taxa de canvi del cost respecte a qualsevol biaix de la xarxa

L'error δ_j^l és exactament igual a la taxa de canvi $\partial C / \partial b_j^l$.

4. Taxa de canvi del cost, respecte a qualsevol pes de la xarxa

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

Equació 15: Taxa de canvi del cost respecte a qualsevol pes de la xarxa

Per últim aquesta equació [Equació 15] ens diu com calcular les derivades parcials $\partial C / \partial w_{jk}^l$ en termes de δ^l i a^{l-1}que ja sabem calcular.

2.1.5.5. Algorisme de retropropagació

Finalment, ara que ja coneixem com fer els càlculs en formes matricials i sabem les quatre equacions fonamentals per aplicar l'algorisme de retropropagació, anem a veure els passos a seguir per aplicar aquest algorisme:

1. **Entrada x:** establir l'activació corresponent a a^1en la capa d'entrada.
2. **Feedforward:** Per cada $\mathrm{l} = 2,3,\ldots,\mathrm{L}$ calcular $z^l = w^l a^{l-1} + b^l$ i $a^l = \phi(z^l)$.
3. **Error de sortida $\boldsymbol{\delta^L}$:** calcular el vector $\delta^L = \nabla_a C \odot \phi'(z^L)$.
4. **Retropropagar l'error:** per cada $l = L-1, L-2, \ldots, 2$ calcular $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \phi'(z^l)$
5. **Sortida:** el gradient de la funció de cost està donat per $\frac{\partial C}{\partial b_j^l} = \delta_j^l$ i $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$.

Anem a veure amb més calma aquest algorisme punt per punt:

1. El primer pas lògic és donar una entrada a la xarxa neuronal per a començar l'algorisme.
2. Es van calculant les matrius de pesos, biaixos i activacions per a cada una de les capes fins a l'última d'elles.
3. Un cop tenim el resultat de sortida i hem calculat l'error amb la funció de cost, calcularem l'error de la capa de sortida δ^L.
4. Retropropaguem l'error per a cada capa, anem calculant els vectors d'error δ^l fins a arribar a la penúltima capa.
5. En arribar a la penúltima capa, podem calcular les variables parcials de tots els pesos i els biaixos per saber com canvia la funció de cost en funció d'aquests. Tenint aquests gradients, aplicant l'algorisme del gradient descendent, s'ajustarien els pesos i els biaixos per a arribar al punt més baix de la funció.

2.1.5.6. Altres implementacions

Tot el que hem explicat sobre com aprèn una xarxa neuronal, l'algorisme de retropropagació i l'algorisme del gradient descendent, són solament una introducció a les implementacions de xarxes neuronals reals. Existeixen millores d'aquests algorismes, per exemple el gradient descendent estocàstic o alternatives a algorismes d'optimització de funcions com "AdaGrad" o "Adam", que és un dels més estesos [6]. Pel que fa a la retropropagació també existeixen millores a l'algorisme que hem mostrat, però si contéssim tot el que existeix no acabaríem mai i la complexitat augmenta.

El que es pretén mostrar és que al darrere de l'aprenentatge de les xarxes neuronals no existeix cap màgia, sinó que hi ha algorismes més o menys complexos que són capaços d'aprendre analitzant exemples utilitzant estadística.

2.2. Xarxes neuronals convolucionals

Fins ara hem vist que és una xarxa neuronal artificial i com aquesta és capaç d'aprendre i millorar a resoldre problemes a mesura que li anem introduint nous exemples, però i si posem un cas pràctic? Suposem que ens demanen que dissenyem un model el qual ha de classificar imatges d'animals, la idea és que el model aprengui a distingir diferents tipus d'animals mentre que li anem passant imatges classificades. La forma habitual de resoldre aquest problema és fer el disseny de forma que la seva capa d'entrada estigui formada per tots i cada un dels píxels de les imatges, les capes ocultes i la capa de sortida quedaria a elecció del dissenyador.

Un cop tenim el nostre model implementat i entrenat, fem proves amb les imatges que tinguem per testejar i veiem que la precisió del model, a l'hora d'encertar els animals, és molt baixa.

Veient aquesta situació canviem el model afegint més capes i provant altres funcions d'activació, això no obstant, el model continua donant resultats dolents. Què està passant? Doncs que una xarxa neuronal convencional no analitza les imatges de la mateixa forma que ho faria una persona, nosaltres quan veiem una imatge, per exemple la d'un gos, sabem que ho és perquè distingim els seus trets característics com el musell, orelles, cua, forma, etc. Per a una xarxa neuronal convencional no hi hauria cap diferència entre una imatge normal i la mateixa, però amb els píxels desordenats com en la figura [Figura 13], on ràpidament veiem que és impossible distingir-hi cap cosa.

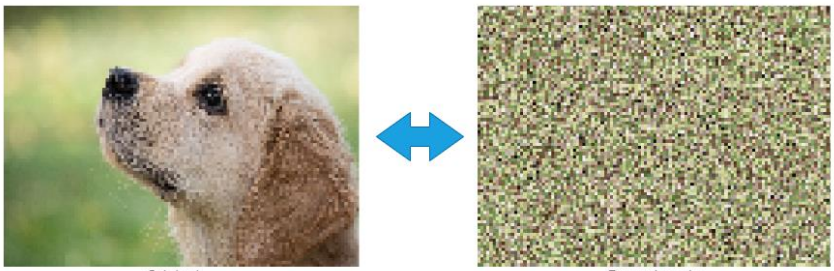


Figura 13: Entrada d'una imatge en cada model

El fet d'introduir en la capa d'entrada tots els píxels d'una imatge comporta dos inconvenients:

- Hem vist que per al model no importa si la imatge és original o està desordenada, això es tradueix en el fet que no es fa cap anàlisi de píxels veïns i el model no és capaç de trobar cap patró en la imatge amb el que pugui distingir un animal d'un altre que provoca que la precisió de les prediccions sigui baixa.
- Donar com a entrada una imatge completa al model és molt ineficient en termes de computacionals i de memòria. Imaginem que volem fer l'entrenament amb 50.000 imatges amb una resolució de 516 per 516 a color, aquesta quantitat d'imatges i amb aquesta resolució ocuparia una gran quantitat de memòria i la capa d'entrada hauria de ser de la mida de $516 * 516 * 3 = 798.768$, la qual implica una quantitat molt elevada de càlculs.

Amb el que hem vist de la classificació d'imatges utilitzant un model de xarxa neuronal convencional, podem determinar que aquest és ineficient computacionalment i ineficaç en els resultats que dona, per això existeix l'evolució natural i un subtipus de les xarxes neuronals artificials, les xarxes neuronals convolucionals, aquesta implementació està motivada per la forma en què les persones analitzem imatges i el cervell busca trets característics dels elements, i no la imatge en general.

En poques paraules, una xarxa neuronal convolucional [7] afegeix a un model tradicional una sèrie de passos previs a la capa d'entrada, amb la intenció de què el model sigui capaç de distingir patrons complexos. Esquemàticament, es veuria com en la figura [Figura 14].

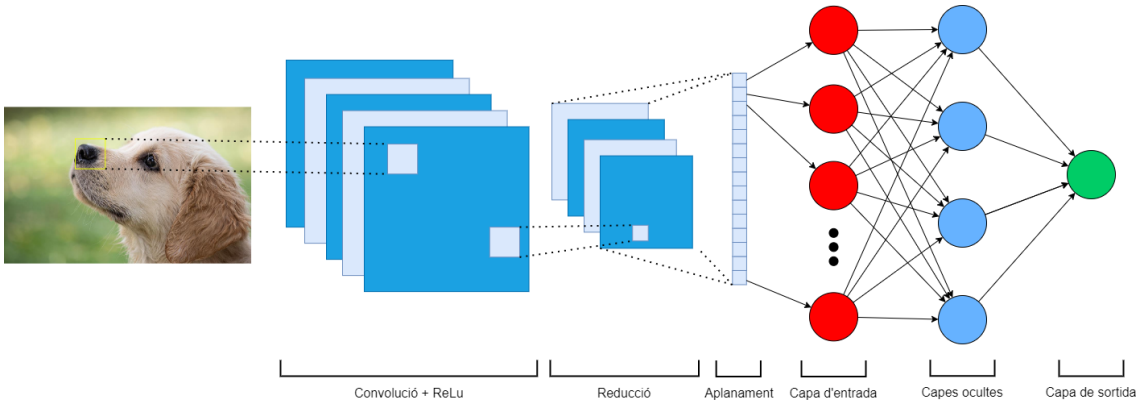


Figura 14: Capes d'una xarxa neuronal convolucional

2.2.1. Capes

En la figura [Figura 14] tenim les noves capes que apareixen en el model, a més de les que ja havíem vist fins ara, és important saber les característiques generals d'aquesta, els processos que duen a terme i l'impacte que tenen de cara al tractament d'imatges per a poder entendre com el nou model aprèn a classificar imatges.

Les capes que s'incorporen al model són: la capa de convolució, capa de reducció i la capa d'aplanament, on aquesta última estarà connectada directament amb la capa d'entrada. A diferència de les capes d'una xarxa neuronal convencional que segueixen una seqüència lògica: capa d'entrada, capes ocultes i capa de sortida, ara tenim la capa de convolució i la capa de reducció les quals no tenen cap ordre definit i es pot concatenar un gran nombre d'aquestes amb diferents configuracions. En general, se solen fer combinacions de dos en dos: capa de convolució més capa de reducció *n* vegades.

Seguint amb les capes de convolució i reducció, ja que són una part essencial perquè els resultats del model sigui els més precisos possibles, el nombre d'aquestes determinarà la profunditat, la qual serà determinant perquè el model sigui capaç de trobar bons patrons identificatius de les imatges. Per comprendre millor com aquestes actuen, veurem un exemple d'una xarxa neuronal convolucional que classifica números de l'1 al 9 a partir d'un dibuix, va ser desenvolupada a la Universitat de Ryerston [8] i consisteix en una aplicació web on es pot veure de forma visual que fa cada capa d'una xarxa neuronal convolucional:

En la figura [Figura 15] tenim una captura d'aquesta eina on hi han indicades les capes, primer tenim l'entrada on hem dibuixat el número cinc, després dues configuracions de convolució més reducció i per últim l'aplanament.

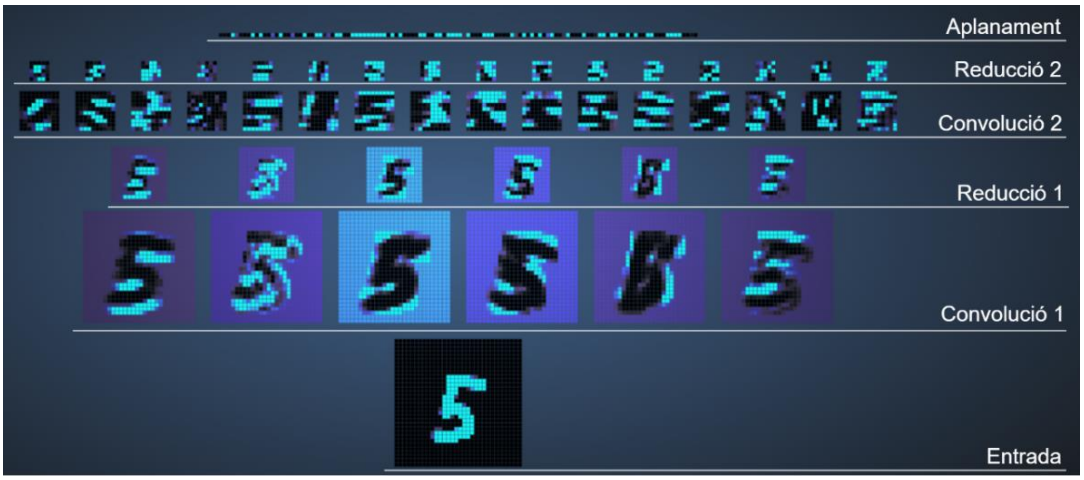


Figura 15: Exemple de funcionament

En les seccions posteriors analitzarem les operacions matemàtiques que hi ha al darrere de cada una de les parts, però ara ens centrarem en com va evolucionant la imatge a mesura que va passant per cada capa d'una forma més visual:

- **Entrada**

  En la capa d'entrada podem dibuixar un número de l'1 al 9 i en aquest cas hem dibuixat el número 5.

- **Convolució i reducció 1**

  Quan la imatge passa per la capa de convolució a primera vista veiem que aquesta ha passat per diferents filtres amb els quals s'han canviat els colors i les vores del número, i bàsicament això és el que fa l'operació de convolució, aplicar filtres sobre les imatges i reduir una mica la imatge. I pel que fa a la reducció, simplement agafa la imatge i redueix la seva mesura.

- **Convolució i reducció 2**

  Després que la imatge hagués passat per les dues capes anteriors encara era fàcil veure que el número era el cinc, però després d'aplicar aquestes, és molt difícil per nosaltres deduir que aquestes imatges provenien del cinc, però per a la xarxa neuronal convolucional aquestes imatges tan críptiques seran els patrons que buscarà en les imatges per determinar que la imatge inicial era un cinc.

- **Aplanament**

  En última posició, l'aplanament consisteix a agafar les últimes imatges que tenim, ajuntar-les i passar-les de dues dimensions a una dimensió, d'aquesta forma cada posició del vector serà una entrada per a la xarxa neuronal.

Havent vist com es tracten les imatges en una xarxa neuronal convolucional, el primer que podríem pensar és que a l'utilitzar un elevat nombre de capes podria implicar una perduda d'informació en les imatges, ja que estem reduint la mida d'aquestes, aquest seria un enfocament erroni, perquè no és que estiguem perdent informació, sinó que estem resumint la imatge fins al punt que l'únic que apareix són patrons profunds i característics.

Aquest és l'objectiu principal d'aquest tipus de xarxes neuronals, passar la imatge per diverses capes de convolució i reducció per obtenir patrons molt concrets, que com hem pogut veure

amb dues aplicacions d'aquestes capes hem obtingut uns patrons característics del número cinc, però aquesta tècnica la podem portar a imatges reals ja siguin d'objectes, localitzacions, formes, cares, etc. Un cop tenim aquests patrons la xarxa neuronal serà l'encarregada d'interpretar-los, determinar a quin objecte pertanyen i donar el resultat.

2.2.1.1. Capa de convolució

Aquesta capa rep el nom per l'operació matemàtica de convolució [9] i també dona nom al propi model de xarxa neuronal convolucional a causa de la gran importància d'aquesta capa, ja que és la que atorga la capacitat d'obtenir patrons a partir d'imatges.

Abans de passar a la definició de la convolució fem un petit recordatori del que és una imatge: una imatge és una matriu on cada posició representa un píxel, en el cas d'una imatge en escala de grisos, la seva dimensió és $n * n$ on els valors de la matriu tenen un rang de 0 a 255 que representa l'escala de grisos, i pel que fan les imatges en color (RGB Red-Green-Blue) són matrius de $n * n * 3$ dimensions, on aquestes 3 dimensions conformarien les capes de vermell, verd i blau, els valors també comprenen el rang de 0 a 255 per cada color.

Anant ja a la definició de l'operació de convolució, aquesta consisteix a recórrer la imatge d'entrada agafant grups de píxels i anar operant amb productes escalars contra una petita matriu anomenada "kernel" [10], la fórmula de la capa de convolució és la següent [Equació 16]:

$$Y_j = g(K_{ij} \otimes Y'_i)$$

Equació 16: Operació de convolució

On la sortida Y, sent j el número de kernel resultat d'aplicar una funció d'activació g (normalment ReLu) sobre la multiplicació del kernel K de la posició i de la matriu Y' sen aquesta la matriu d'entrada. Veiem l'exemple de la figura [Figura 16]:

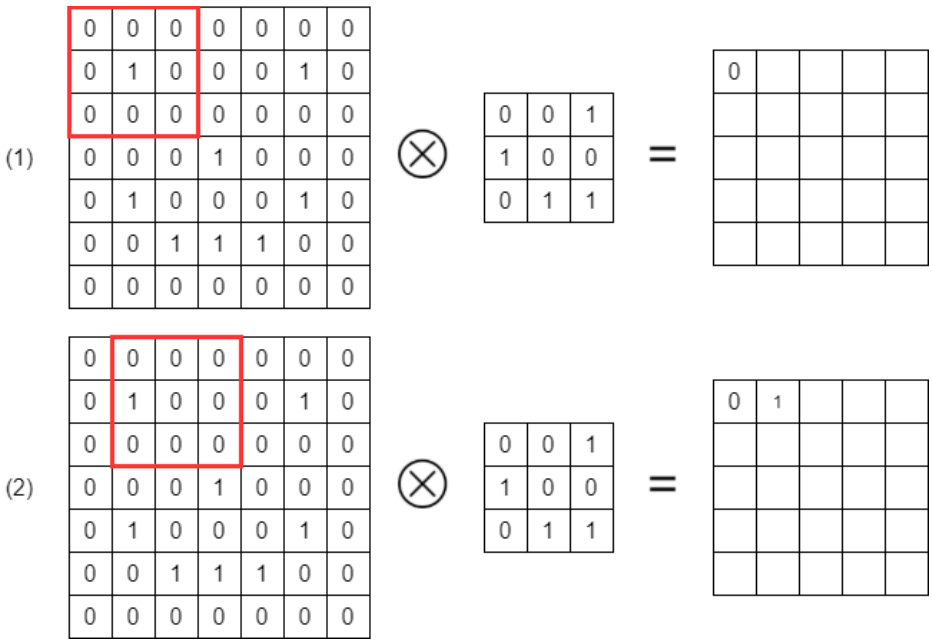


Figura 16: Exemple de convolució 1

En la figura [Figura 16] podem veure com ens anem desplaçant per la matriu d'entrada i anem operant amb el kernel donant com a resultat una matriu més petita, els càlculs són els següents:

$$(1)\ 0 * 0 + 0 * 0 + 0 * 1 + 0 * 1 + 1 * 0 + 0 * 0 + 0 * 0 + 0 * 1 + 0 * 1 = 0$$

$$(2)\ 0*0+0*0+0*1+1*1+0*0+0*0+0*0+0*1+0*1=1$$

Repetint aquest procés per tota la matriu d'entrada i obtenim el següent resultat, [Figura 17]:

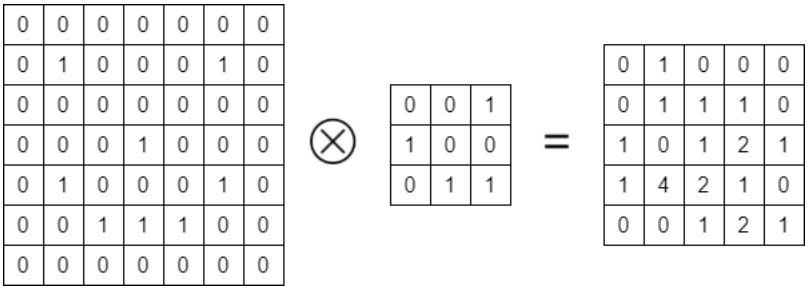


Figura 17: Exemple de convolució 2

Realment la convolució no té cap complicació, solament consisteix a fer sumes i multiplicacions, ara bé hem de pensar que aquesta operació es realitza un gran nombre de cops i normalment s'utilitzen varis kernels per a poder trobar més patrons, també és realitzem entrenaments amb milers d'imatges i sobretot afectarà la mida d'aquestes, tot això sumat que es posen diverses capes de convolució, implicarà un gran nombre de càlculs i haurem de tenir en compte aquestes condicions.

En la figura [Figura 18] hi ha alguns exemples de kernels aplicats a una imatge que serveixen per a detectar vores, d'aquesta forma ens podem fer una idea de com aquests filtres poden canviar una imatge per a destacar alguna característica:

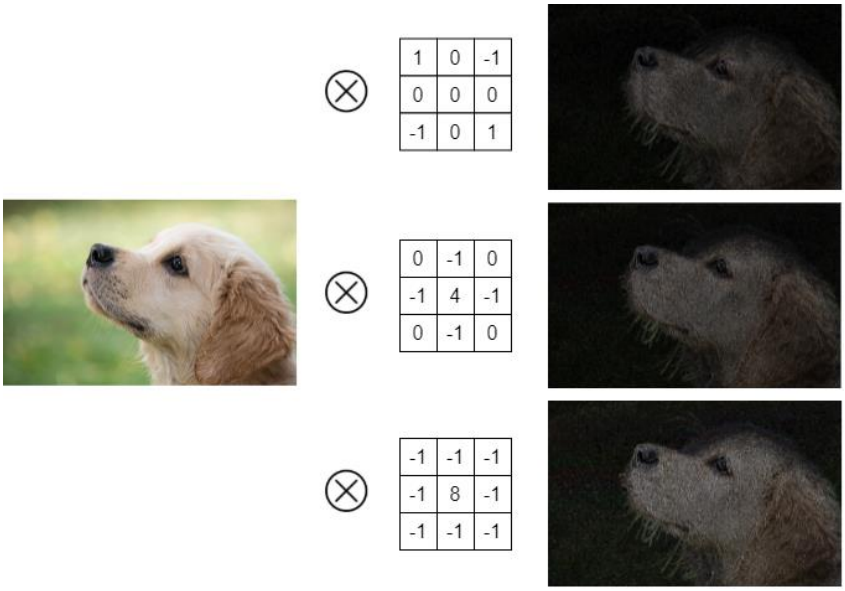


Figura 18: Aplicació de convolució en una imatge ral

Un cop vist com funciona la convolució en imatges ens pot sorgir el següent dubte, com definim els millors kernels per a detectar característiques en diferents tipus d'imatges? Doncs la gràcia de les xarxes neuronals convolucionals és que nosaltres ens abstraiem d'aquesta tasca i serà responsabilitat del model trobar-los.

2.2.1.2. Capa de reducció

La capa de reducció [11] consisteix a fer un mostreig de la imatge d'entrada per així reduir la seva mida. L'objectiu d'aquest mostreig és reduir la imatge conservant les seves característiques principals, ja que com hem vist en l'apartat anterior, la mida de les imatges augmenta els càlculs

que s'han de fer en les capes de convolució i aquesta capa ens permet reduir costos sense perdre informació de la imatge.

La tècnica que utilitzarem per a fer aquest mostreig s'anomena "max-pooling" i, semblant a com funciona la convolució, anirem recorrent la imatge d'entrada agafant grups de píxels, però en comptes d'operar amb altres matrius, d'aquest grup agafarem el píxel amb el valor màxim, d'aquesta manera aconseguirem el mostreig de la imatge. Exemple en la figura [Figura 19].

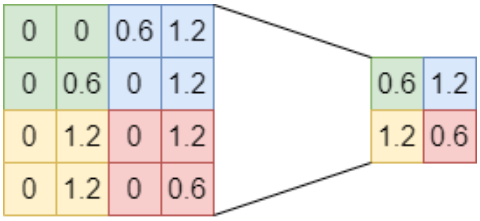


Figura 19: Exemple de reducció

2.2.1.3. Capa d'aplanament

Arribats a aquest punt, ja tenim unes imatges molt petites on hi ha representades les característiques principals que hagi pogut trobar el model, però aquestes característiques les hem de passar a una xarxa neuronal convencional perquè aprengui a interpretar-les.

Per a introduir aquestes imatges el que farem serà aplanar-les, és a dir, agafar els píxels que són matrius de 2*2 dimensions i convertir-lo en un vector pla concatenant les files de les matrius. Un cop tenim aquest vector, per introduir-lo a la xarxa neuronal, farem que cada una de les posicions sigui una entrada de la xarxa neuronal convencional.

2.2.2. Algorisme de retropropagació

Hem vist en què consisteix una xarxa neuronal convolucional i no és més que afegir una sèrie de capes que tenen com a objectiu aprendre a trobar característiques en les imatges i resumir-les per a posteriorment passar aquesta informació a una xarxa neuronal com la que hem vist en l'apartat anterior.

Sabent això ara la qüestió és: com aprenen aquestes noves capes? Doncs de la mateixa forma que coneixem, amb l'algorisme de retropropagació, això sí, amb certes modificacions. No entrarem al detall en les matemàtiques que hi ha al darrere d'aquesta variació de l'algorisme, l'explicació es pot resumir d'una forma ràpida.

De la mateixa forma que es calcula l'error de totes les neurones fent la retropropagació per tot el model en les xarxes neuronals convencionals, ara, passa el mateix en les capes convolucionals, concretament en els kernels els quals per cada un valor de les matrius, el podem veure com si fos un pes de les neurones que s'ha d'ajustar per disminuir l'error.

Així doncs, per a fer funcionar l'algorisme de retropropagació en les xarxes neuronals convolucionals simplement hem de veure els kernels de les capes convolucionals com si fossin neurones.

2.3. Entrenament de xarxes neuronals convolucionals

El correcte entrenament d'un model és tant o més important que el mateix algorisme en si, perquè, ja podem tenir el millor algorisme de xarxes neuronals mai fet, que si l'entrenament que fem és incorrecte, és impossible que obtinguem bons resultats.

Quan parlem d'un "bon entrenament" ens referim a diversos aspectes clau que hem de tenir en compte quan anem a entrenar un model, com per exemple: el set de dades, èpoques dels

entrenaments, augment del set de dades, rati d'aprenentatge del model, mètriques per avaluar l'entrenament, altres tecnologies per avaluar els resultats, etc. Anem a veure els més importants i els que tindrem en compte per entrenar els nostres models [12]:

- **Set de dades**

  El set de dades és la base d'un entrenament, sense aquest no tenim res amb què entrenar, per això hem de buscar-ne el que més s'adapti als nostres objectius.

  Per escollir el més adient, uns dels aspectes més importants que ha de tenir és, si per exemple volguéssim fer un model que diferencies diversos tipus de mamífers, serien: que tingui una gran quantitat d'imatges, varietat en els animals, que hi hagi diverses imatges per cada animal en diferents enfocaments, ha d'haver-hi un equilibri en el nombre d'imatges per animal...

- **Augment de dades**

  Una tècnica àmpliament utilitzada en els entrenaments, sobretot en xarxes neuronals convolucionals, és l'augment del set de dades. La metodologia que se segueix per fer augmentar el nombre d'imatges en un set és, de forma aleatòria, agafar les imatges i aplicar un filtre, o desenfocament, o rotació, o ampliació d'una zona, entre d'altres [13].

  En la figura [Figura 20] podem veure un exemple de l'augment d'una imatge.

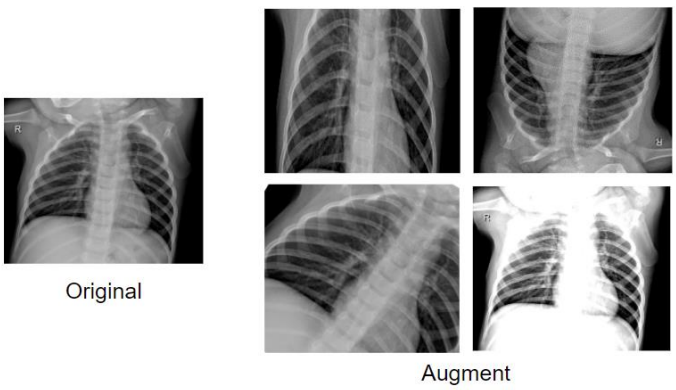


Figura 20: Augment d'imatges

- **Èpoques dels entrenaments**

  Quan estem entrenant un model i fem una única passada del nostre set de dades és molt complicat que l'entrenament sigui efectiu pel fet que en general un model està compost de milions de paràmetres a entrenar i sigui la mida que sigui el set de dades, aquests paràmetres necessiten tornar a fer més passades del set. Aquestes passades s'anomenen èpoques i, en general, el nombre d'èpoques estarà definit per la mesura d'aprenentatge del model, és a dir, si analitzant les mètriques s'observa que el model continua millorant es van augmentant les èpoques.

- **Mètriques**

  Les mètriques [14] són funcions que s'utilitzen per mesurar com aprèn un model en funció del nombre d'èpoques o quan de bé està fent les prediccions. Existeixen diferents mètriques que poden ser més útils o menys segons els nostres objectius, anem a veure les més utilitzades:

- Exactitud (Accuracy)

  Aquesta mètrica crea dues variables locals, totals i compteig, que s'utilitzen per calcular la freqüència amb què les prediccions coincideixen amb els valors reals. Aquesta freqüència es torna finalment com a exactitud: una operació idempotent que simplement divideix el total entre el compteig.

- Exactitud binària (Binary Accuracy)

  Aquesta mètrica crea dues variables locals, totals i compteig, que s'utilitzen per calcular la freqüència amb què les prediccions coincideixen amb els valors reals. Aquesta freqüència es torna finalment com a exactitud binària: una operació idempotent que simplement divideix el total entre el compteig. A diferència de la exactitud normal, la exactitud binària s'utilitza en problemes de classificacions binàries.

- Precisió (Precision)

  La mètrica crea dues variables locals, true_positives i false_positives que s'utilitzen per calcular la precisió. Aquest valor es torna finalment com a precisió, una operació idempotent que simplement divideix true_positives per la suma de true_positives i false_positives.

- Record (Recall)

  Aquesta mètrica crea dues variables locals, true_positives i false_negatives, que s'utilitzen per calcular la recuperació. Aquest valor es torna finalment com a record, una operació idempotent que simplement divideix true_positives per la suma de true_positives i false_negatives.

2.4. Pneumònia

La pneumònia [15] és una malaltia del sistema respiratori que consisteix en la inflamació de naturalitat infecciosa dels espais alveolars dels pulmons.

Existeixen alguns tipus de pneumònia, però per aquest treball ens enfocarem en solament la diagnosi general de pneumònia que es pot identificar a través d'una radiografia de tòrax en forma d'una taca que coincidirà amb la inflamació dels espais alveolars, com en la imatge [Figura 21].

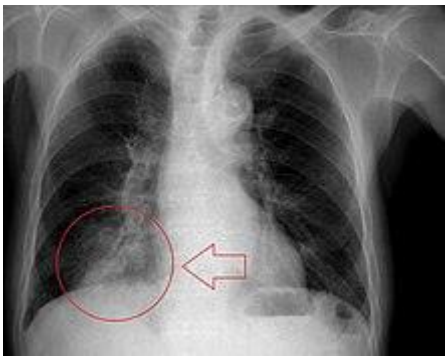


Figura 21: Pneumònia

L'objectiu dels nostres models d'intel·ligència artificial serà el d'identificar aquestes taques per fer el diagnosi.

3. Entorn de desenvolupament dels experiments

Abans d'endinsar-nos en el codi i començar a provar diferents implementacions de xarxes neuronals convolucionals, és imprescindible que establim des d'un començament l'entorn sobre el qual treballarem, quina arquitectura tindrà el projecte i el procediment experimental que seguirem, perquè no ens desviem dels nostres objectius a mesura que anem avançant. D'aquesta forma en aquesta secció assentarem les bases del projecte i de la part experimental.

3.1. Sets de dades

En aquest apartat llistarem els sets de dades que utilitzarem per entrenar els nostres models.

- **RSNA Pneumonia Detection Challenge**

  Aquest set de dades [16] consisteix en un concurs d'algorismes d'intel·ligència artificial, potenciat per al RSNA (Radiological Society of North America) per a detectar pneumònies en imatges de radiografies.

  El set està format per: 26.684 imatges etiquetades que utilitzarem per a l'entrenament, test i validació fent una divisió, a més hi ha 3.000 imatges sense etiquetar i informació addicional dels pacients que descartarem, ja que per als nostres objectius no ens interessa.

  Pel que hem pogut observar en aquest set de dades les imatges d'aquest són molt variades, en el sentit de què hi ha imatges que es veuen bé i altres en les que hi ha molt soroll, haurem de veure si els models són capaços de trobar pneumònies en tota mena d'imatges.

- **Chest X-Ray Images (Pneumonia)**

  Aquest set de dades [17] consisteix en un conjunt d'imatges classificades de radiografies de pacients amb pneumònia i està disponible a Kaggle. Posa a disposició del públic: 5.856 imatges etiquetades que ja estan separats en els tres conjunts d'entrenament, test i validació que utilitzarem en els nostres models.

  Pel que hem pogut observar en aquest set de dades les imatges són molt més netes que les de l'anterior set, ja que totes les imatges estan enfocades de la mateixa forma i estan bastant netes, haurem de veure si per als models és més fàcil trobar pneumònies en sets de dades més nets.

3.2. Entorn de desenvolupament

Per a fer el desenvolupament dels laboratoris utilitzarem Python 3.8 i per a facilitar la instal·lació de paquets i entorns utilitzarem Anaconda 3, ja que ofereix una forma senzilla de crear entorns Python per a provar diferents versions de llibreries i paquets.

La raó de fer servir Python és que una gran quantitat de llibreries d'intel·ligència artificial estan disponibles per aquest llenguatge, i inclòs són compatibles entre elles, d'aquesta forma tenim un ecosistema molt potent per a fer aplicacions i estudis amb intel·ligència artificial i ciències de les dades en general.

Les principals llibreries que usarem per a implementar xarxes neuronals convolucionals i d'altres per a analitzar el funcionament d'aquestes són:

- Per a fer les implementacions utilitzarem la capa Keras de Tensorflow.
- Pandas i NumPy per a fer gestió de dades en brut i d'imatges.
- Lime [18] per a veure en què s'està basant la xarxa neuronal per prendre decisions.

- Matplotlib, Sklearn i Mlxtend per a mostrar gràfics.
- Pydicom i Opencv per a convertir imatges a vectors.

Hi ha altres llibreries secundàries o dependències que es poden veure en la part de requisits del projecte que no utilitzarem directament.

3.3. Arquitectura del projecte

El projecte està dividit en dues parts: la part del tractament del set de dades on estarà total la lògica per a poder carregar les imatges, i la part dels laboratoris que estarà composta per diferents "notebooks".

3.3.1. Tractament i preparació de dades

La part de tractament i preparació de dades és l'encarregada de subministrar totes les imatges i les seves etiquetes en el format que Keras requereix, per fer-ho tenim una sèrie de capes de processament de dades que són els passos necessaris per passar de les dades en brut a tenir dades netes i en el format adient.

Com a entrada d'aquesta part tenim els dos sets de dades: PDC (Pneumonia Detection Challenge) i CXR (Chest-X-Ray) que són els dos sets que estan disponibles a Kaggle i com a sortida tenim tres vectors (entrenament, test i validació) on estaran totes les imatges en format píxel RGB i tres llistes amb les seves respectives etiquetes (pneumònia, no pneumònia). Existeix una dificultat en fer aquest disseny i és que els dos sets estan disponibles en dos formats molt diferents i si en un futur en volem afegir més, hem de tenir un sistema modular que es pugui adaptar a diferents sets d'entrada. Per aquest motiu aquesta part es divideix en diferents capes on cada una d'aquestes la seva sortida ha de tenir unes regles estrictes.

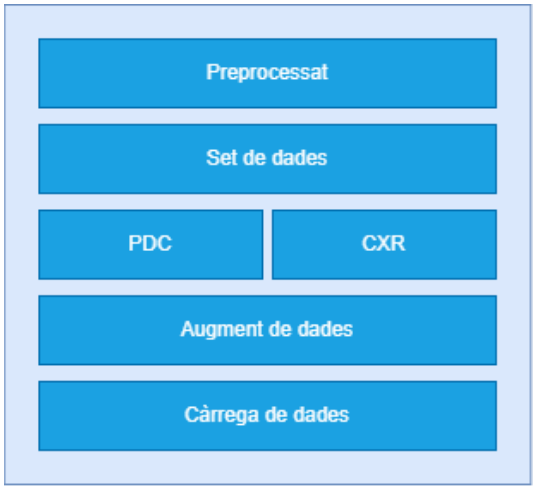


Figura 22: Estructura del projecte

Anem a veure quines són les responsabilitats de cada capa que podem veure en la figura [Figura 22]:

- **Preprocessat**
  - Entrada: un directori que contingui el set de dades en el format qui vingui.
  - Sortida: un directori on hi haurà un subdirectori amb totes les imatges dividides en tres parts: part d'entrenament, part de test i part de validació, a més, hi haurà tres fitxers en format csv amb la següent informació de les tres parts: el nom de la imatge, la nova ruta on estarà localitzada la imatge i la diagnosi.

La funció d'aquesta capa és la de crear una estructura de directoris que serà la mateixa per a tots els sets de dades, això amb l'objectiu de simplificar la implementació de les capes posteriors, també, tindrà l'opció de reduir el nombre d'imatges si s'indica.

- **Set de dades**
  - Entrada: un directori que contingui el set de dades amb el format que dona com a sortida el preprocessat.
  - Sortida: la sortida és la mateixa capa de càrrega de dades

  Aquesta, és una capa abstracta per generalitzar algunes de les funcions dels sets de dades (PDC i CXR) amb l'objectiu de no repetir codi.

- **PDC i CXR**
  - Entrada: l'entrada és la mateixa capa de set de dades.
  - Sortida: la sortida és la mateixa capa de càrrega de dades.

  Aquesta capa és una capa intermèdia, formada per tantes capes com sets de dades hi hagi, on hi ha implementats mètodes específics per carregar en memòria els diferents formats de les imatges i d'altres tractaments que es requereixin.

- **Càrrega de dades**
  - Entrada: el directori on hi hagi les imatges.
  - Sortida: un vector en memòria on hi haurà totes les imatges.

  Aquesta capa carrega en memòria els sets: d'entrenament, test i validació.

- **Augment de dades**
  - Entrada: el directori on hi hagi les imatges o les imatges allotjades en un vector en memòria.
  - Sortida: un vector en memòria on hi haurà totes les imatges amb diferents modificacions.

  Aquesta última capa té per objectiu augmentar el set de dades de diferents formes que s'especifiquin com rotacions, ampliació, efecte mirall...

3.3.2. Estructura dels experiments

Tots els experiments estan implementats en diferents Jupyter Notebooks, els quals s'executaran i es quedaran els resultats enregistrats perquè puguin ser consultats sense haver de tornar a fer cap execució extra.

L'estructura dels laboratoris és la següent:

- **Experiment 1**

  Aquest experiment té per objectiu fer una bona preparació dels sets de dades que es faran servir per a entrenar, testejar i validar una xarxa neuronal, provant diferents tècniques per equilibrar i augmentar el conjunt de dades, a més es provaran diferents profunditats per a veure com afecta en l'entrenament i els resultats.

  La seva estructura és la següent:

- Etapa 1: en aquesta etapa es provaran diferents arquitectures de xarxes neuronals convolucionals amb un conjunt de dades d'entrada sense cap tractament previ, sense importar si els casos positius i negatius estan desequilibrats.
- Etapa 2: en aquesta segona etapa es provaran diferents arquitectures de xarxes neuronals convolucionals amb un conjunt de dades d'entrada amb un tractament previ per a equilibrar els casos positius i negatius.
- Etapa 3: en aquesta última etapa es provaran diferents arquitectures de xarxes neuronals convolucionals amb un conjunt de dades d'entrada amb un tractament previ per a equilibrar els casos positius i negatius i s'aplicaran tècniques d'augment de dades.

- **Experiment 2**

  Aquest experiment té per objectiu buscar l'arquitectura de xarxa neuronal convolucional que doni els millors resultats possibles.

  La seva estructura és la següent:

  - Arquitectura 1: es provarà la següent arquitectura: 2 x Convolucions (kernels = 16) + Normalització + Reducció + 2 x Convolucions (kernels = 32) + Normalització + Reducció + 2 x Convolucions (kernels = 64) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció
  - Arquitectura 2: es provarà la següent arquitectura: 2 x Convolucions (kernels = 16) + Normalització + Reducció + 2 x Convolucions (kernels = 32) + Normalització + Reducció + 2 x Convolucions (kernels = 64) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció
    En comptes de la funció d'activació ReLu, utilitzarem la LeackyReLu.
  - Arquitectura 3: es provarà la següent arquitectura: 3 x Convolucions (kernels = 32) + Normalització + Reducció + 3 x Convolucions (kernels = 64) + Normalització + Reducció + 3 x Convolucions (kernels = 128) + Normalització + Reducció + 3 x Convolucions (kernels = 256) + Normalització + Reducció + 3 x Convolucions (kernels = 256) + Normalització + Reducció
  - Arquitectura 4: es provarà la següent arquitectura: 2x Convolució (kernels = 16) + Normalització + Reducció + Convolució en profunditat (kernels = 32) + Convolucions (kernels = 32) + Normalització + Reducció + Convolució en profunditat (kernels = 64) + Convolucions (kernels = 64) + Normalització + Reducció + Convolució en profunditat (kernels = 128) + Convolucions (kernels = 128) + Normalització + Reducció + Convolució en profunditat (kernels = 128) + Convolucions (kernels = 128) + Normalització + Reducció

- **Experiment 3**

Aquest Experiment té per objectiu provar l'arquitectura VGG16 [19] preentrenada amb el set de dades Imagnet i tornar a entrenar-la amb els nostres sets de dades.

4. Experiments

4.1. Experiment 1

En aquest primer experiment veurem com podem fer una bona preparació del set de dades. Ho farem mitjançant diferents tècniques de preparació amb l'objectiu d'obtenir els millors resultats possibles en els entrenaments i els resultats, també, aprofitarem que estem fent proves amb diferents volums de dades per provar en quin grau afecta la profunditat de les xarxes neuronals convolucionals.

Les arquitectures que provarem en les tres fases són les següents:

Taula 1: Arquitectures del experiment 1

Número	Arquitectura
1	Convolució (kernels = 8) + Reducció + Convolució (kernels = 16) + Reducció + Convolució (kernels = 32) + Reducció
2	Convolució (kernels = 8) + Reducció + Convolució (kernels = 8) + Reducció + Convolució (kernels = 16) + Reducció + Convolució (kernels = 32) + Reducció
3	Convolució (kernels = 16) + Reducció + Convolució (kernels = 16) + Reducció + Convolució (kernels = 32) + Reducció + Convolució (kernels = 64) + Reducció
4	Convolució (kernels = 8) + Reducció + Convolució (kernels = 16) + Reducció + Convolució (kernels = 32) + Reducció + Convolució (kernels = 128) + Reducció + Convolució (kernels = 128) + Reducció
5	Convolució (kernels = 16) + Reducció + Convolució (kernels = 16) + Reducció + Convolució (kernels = 32) + Reducció + Convolució (kernels = 64) + Reducció + Convolució (kernels = 128) + Reducció

Totes aquestes arquitectures estan connectades amb la següent xarxa neuronal: capa oculta (256 neurones, funció relu) + capa oculta (128 neurones, funció relu) + capa de sortida (1 neurona, funció sigmoide)

Tots els experiments estan disponibles al directori de notebooks on les arquitectures són l'últim valor del nom.

4.1.1. Fase 1

En la primera fase no farem cap preprocessat dels sets de dades amb l'objectiu de comprovar si es genera un sobre ajustament en els models, en funció de si hi ha més casos positius que negatius o a l'inrevés, això hauria de provocar un biaix en els resultats, per exemple: si en un set de dades tenim molts més casos amb pneumònia, en qualsevol model les prediccions haurien de sortir majoritàriament pneumònies.

- **Informació dels sets de dades**

Taula 2: Dades PDC, fase 1, experiment 1

Pneumonia Detection Challenge			
	Entrenament	Test	Validació
Normal	15477	1558	16
Pneumònia	4536	1476	4

Taula 3: Dades CXR, fase 1, experiment 1

Chest X-Ray			
	Entrenament	Test	Validació
Normal	1341	234	8
Pneumònia	3875	390	8

- **Resultats dels entrenaments**

L'entrenament per al set de dades de CXR l'hem fet en 15 èpoques i per al PDC 35 èpoques.

Taula 4: Entrenament amb CXR, experiment 1, fase 1

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.0048	0.9993	0.9998	0.9994
2	0.0151	0.9944	0.9967	0.9958
3	0.0167	0.9942	0.9958	0.9964
4	0.0141	0.9947	0.9977	0.9952
5	0.0104	0.9962	0.9977	0.9971

Taula 5: Entrenament amb PDC, experiment 1, fase 1

Pneumonia Detection Challenge				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.0365	0.9856	0.9666	0.9697
2	0.0578	0.9791	0.9540	0.9537
3	0.0812	0.9685	0.9335	0.9250
4	0.0869	0.9661	0.9338	0.9125
5	0.1235	0.9528	0.9114	0.8782

- **Resultats dels test**

Taula 6: Resultats amb CXR, experiment 1, fase 1

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	3.1519	0.7163	0.6878	1.0000
2	2.0573	0.7628	0.7249	1.0000
3	3.8344	0.7228	0.6934	0.9974
4	1.8298	0.7596	0.7256	0.9897
5	2.6016	0.7644	0.7280	0.9949

Taula 7: Encerts amb CXR, experiment 1, fase 1

Chest X-Ray		
	% d'encerts (encerts/total)	
Experiment	**Pneumònia**	**Normal**
1	100	24.36

2	100	36.75
3	99.74	26.50
4	98.97	37.60
5	99.49	39.03

Taula 8: Resultats amb PDC, experiment 1, fase 1

Pneumonia Detection Challenge				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	2.7669	0.6655	0.7794	0.4356
2	1.8409	0.6632	0.8027	0.4079
3	1.8448	0.6691	0.8260	0.4051
4	2.4444	0.6711	0.8128	0.4207
5	1.3703	0.6777	0.7978	0.4519

Taula 9: Encerts amb PDC, experiment 1, fase 1

Pneumonia Detection Challenge		
	% d'encerts (encerts/total)	
Experiment	**Pneumònia**	**Normal**
1	43.64	88.32
2	40.79	90.50
3	40.51	91.91
4	42.07	90.82
5	45.19	89.15

- **LIME**

Els resultats de LIME són de la cinquena arquitectura de CXR i PDC respectivament.

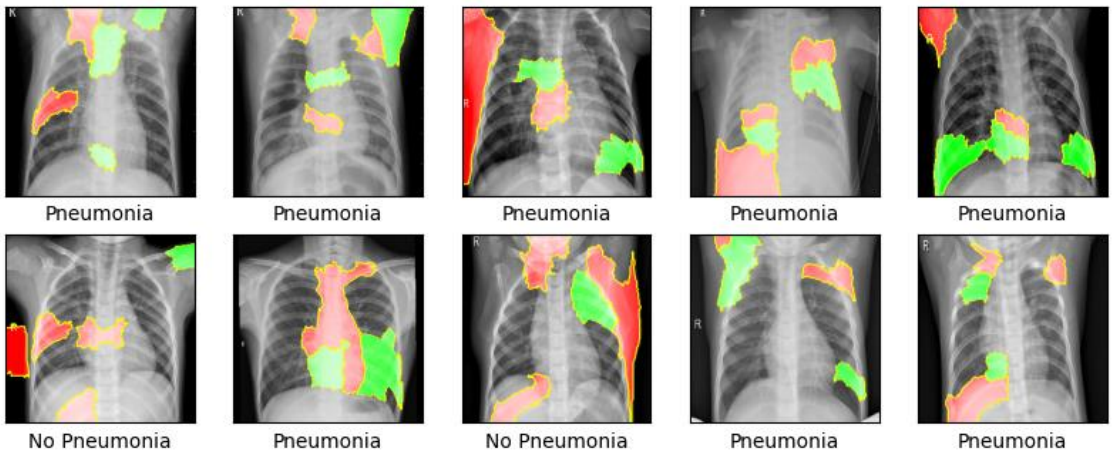


Figura 23: Resultat de LIME amb CXR experiment 1, fase 1

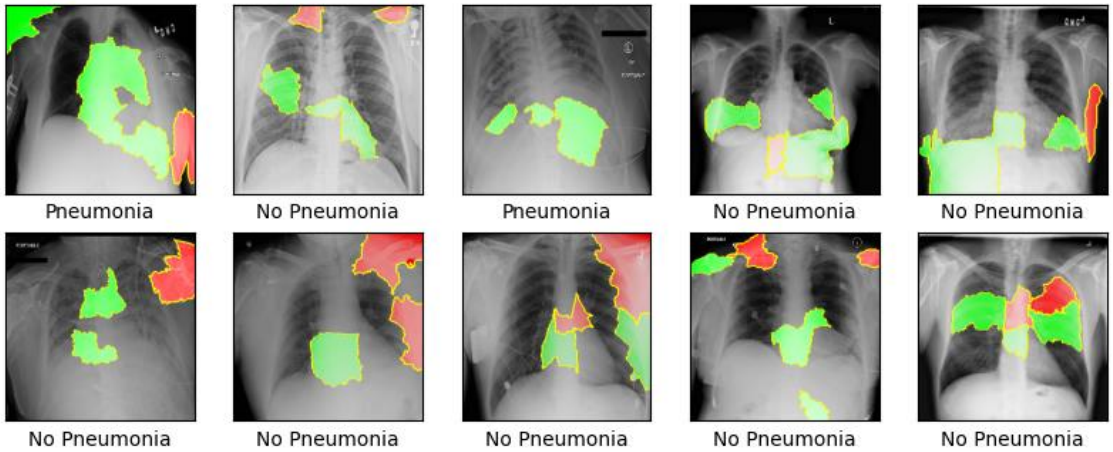


Figura 24: Resultat de LIME amb PDC experiment 1, fase 1

4.1.2. Fase 2

En la segona fase farem un preprocessat dels sets de dades equilibrant els casos positius i negatius, amb l'objectiu de comprovar si en aquest cas els resultats s'equilibren. Concretament, agafarem els casos majoritaris i els rebaixarem al nivell dels minoritaris, amb això no hauríem de tenir biaix en els resultats, ja que la xarxa neuronal convolucional podrà analitzar tants casos positius com de negatius.

- **Informació dels sets de dades**

Taula 10: Dades PDC, experiment 1, fase 2

Pneumonia Detection Challenge			
	Entrenament	Test	Validació
Normal	4643	1558	16
Pneumònia	4536	1476	4

Taula 11: Dades CXR, experiment 1, fase 2

Chest X-Ray			
	Entrenament	Test	Validació
Normal	1341	234	8
Pneumònia	1938	390	8

- **Resultats dels entrenaments**

L'entrenament per al set de dades de CXR l'hem fet en 15 èpoques i per al PDC 35 èpoques.

Taula 12: Entrenament amb CXR, experiment 1, fase 2

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.0140	0.9962	0.9991	0.9944
2	0.0145	0.9932	0.9942	0.9942
3	0.0122	0.9947	0.9982	0.9926
4	0.0251	0.9919	0.9940	0.9923

5	0.0341	0.9905	0.9922	0.9917

Taula 13: Entrenament amb PDC, experiment 1, fase 2

Pneumonia Detection Challenge				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.0336	0.9881	0.9871	0.9895
2	0.0739	0.9693	0.9655	0.9722
3	0.0463	0.9844	0.9819	0.9869
4	0.1060	0.9604	0.9597	0.9605
5	0.1113	0.9563	0.9551	0.9552

- **Resultats dels test**

Taula 14: Resultats amb CXR, experiment 1, fase 2

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	2.0248	0.7933	0.7574	0.9846
2	1.5469	0.7853	0.7510	0.9821
3	2.5470	0.7740	0.7362	0.9949
4	1.6633	0.7981	0.7640	0.9795
5	2.3003	0.7756	0.7376	0.9949

Taula 15: Encerts amb CXR, experiment 1, fase 2

Chest X-Ray		
	% d'encerts (encerts/total)	
Experiment	**Pneumònia**	**Normal**
1	98.46	47.46
2	98.20	45.73
3	99.49	40.60
4	97.95	49.57
5	99.49	41.03

Taula 16: Resultats amb PDC, experiment 1, fase 2

Pneumonia Detection Challenge				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	1.6170	0.7096	0.7017	0.7012
2	1.6891	0.7228	0.7013	0.7493
3	2.0631	0.7179	0.7031	0.7270
4	1.6323	0.7156	0.6941	0.7425
5	1.6926	0.7034	0.6863	0.7188

Taula 17: Encerts amb PDC, experiment 1, fase 2

Pneumonia Detection Challenge	
	% d'encerts (encerts/total)

Experiment	Pneumònia	Normal
1	70.17	71.76
2	74.93	69.77
3	72.70	70.92
4	73.76	69.00
5	71.88	68.87

- **LIME**

Els resultats de LIME són de la cinquena arquitectura de CXR i PDC respectivament.

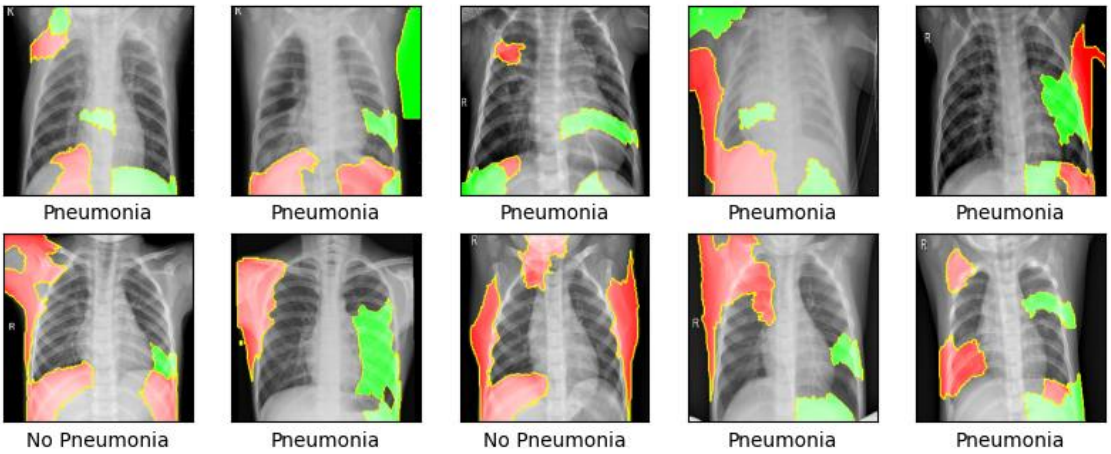


Figura 25: Resultat de LIME amb CXR experiment 1, fase 2

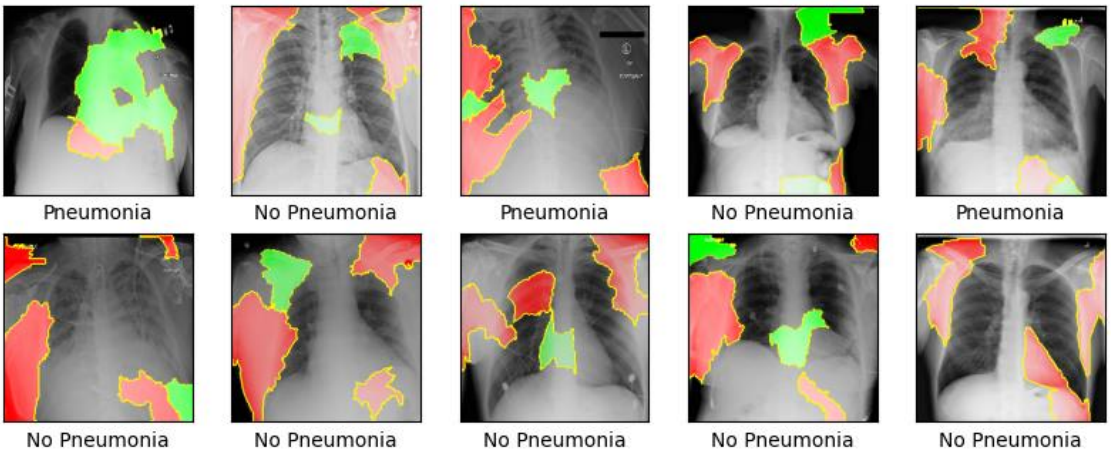


Figura 26: Resultat de LIME amb PDC experiment 1, fase 2

4.1.3. Fase 3

En l'última fase d'aquest experiment, com en l'anterior, equilibrarem els casos positius i negatius, i a més utilitzarem una tècnica d'augment de dades . L'objectiu d'aplicar tècniques d'augment de dades és el d'evitar que la xarxa neuronal convolucional s'acostumi al set de dades d'entrenament provocant un sobre ajustament dels models. En la fase anterior havíem pogut equilibrar els resultats reduint el biaix, però continuem tenint un problema i és que els resultats

dels entrenaments són molt bons, però quan li passem als models el set de test, la qualitat de les prediccions és molt inferior i tenim resultats bastant dolents.

- **Informació dels sets de dades**

Taula 18: Dades PDC, experiment 1, fase 3

Pneumonia Detection Challenge			
	Entrenament	Test	Validació
Normal	4643	1558	16
Pneumònia	4536	1476	4

Taula 19: Dades CXR, experiment 1, fase 3

Chest X-Ray			
	Entrenament	Test	Validació
Normal	1341	234	8
Pneumònia	1938	390	8

- **Resultats dels entrenaments**

L'entrenament per al set de dades de CXR l'hem fet en 50 èpoques i per al PDC 100 èpoques.

Taula 20: Entrenament amb CXR, experiment 1, fase 3

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.2131	0.9118	0.9370	0.9129
2	0.1788	0.9331	0.9514	0.9342
3	0.1893	0.9287	0.9539	0.9259
4	0.2014	0.9261	0.9386	0.9357
5	0.1774	0.9296	0.9544	0.9261

Taula 21: Entrenament amb PDC, experiment 1, fase 3

Pneumonia Detection Challenge				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.5633	0.7123	0.7011	0.7200
2	0.5627	0.7141	0.7063	0.7204
3	0.5410	0.7308	0.7279	0.7338
4	0.5281	0.7393	0.7268	0.7536
5	0.5257	0.7466	0.7489	0.7408

- **Resultats dels test**

Taula 22: Resultats amb CXR, experiment 1, fase 3

Chest X-Ray				
Experiment	**Loss**	**Binary Accuracy**	**Precision**	**Recall**
1	0.2671	0.8862	0.9443	0.8692
2	0.3008	0.8942	0.8699	0.9769
3	0.2761	0.8942	0.8821	0.9590

4	0.2357	0.9215	0.9231	0.9538
5	0.2471	0.9103	0.9304	0.9304

Taula 23: Encerts amb CXR, experiment 1, fase 3

Chest X-Ray		
	% d'encerts (encerts/total)	
Experiment	Pneumònia	Normal
1	86.92	91.45
2	97.69	75.64
3	95.90	78.63
4	95.38	85.76
5	92.56	88.46

Taula 24: Resultats amb PDC, experiment 1, fase 3

Pneumonia Detection Challenge				
Experiment	Loss	Binary Accuracy	Precision	Recall
1	0.5308	0.7416	0.7050	0.8062
2	0.5425	0.7357	0.7463	0.6917
3	0.4995	0.7587	0.7587	0.7392
4	0.5051	0.7584	0.7667	0.7236
5	0.4971	0.7703	0.7409	0.8117

Taula 25: Encerts amb PDC, experiment 1, fase 3

Pneumonia Detection Challenge		
	% d'encerts (encerts/total)	
Experiment	Pneumònia	Normal
1	80.82	68.04
2	69.17	77.73
3	73.92	77.74
4	72.36	79.14
5	81.17	73.11

- **LIME**

Els resultats de LIME són de la cinquena arquitectura de CXR i PDC respectivament.

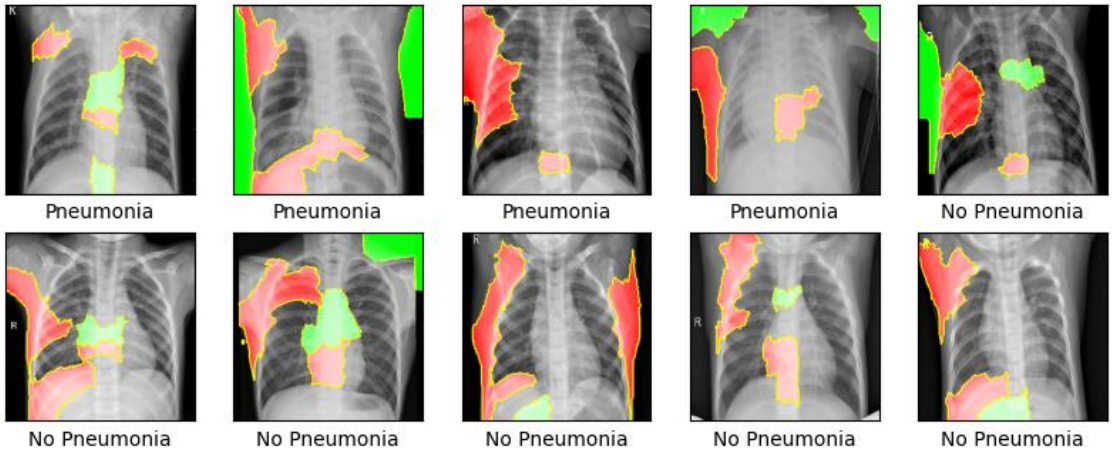


Figura 27: Resultat de LIME amb CXR experiment 1, fase 3

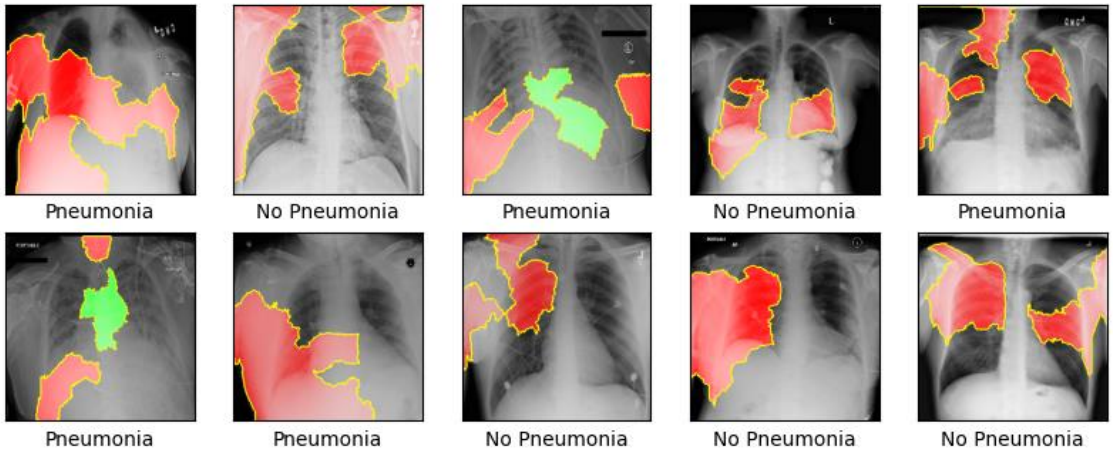


Figura 28: Resultat de LIME amb PDC experiment 1, fase 3

4.2. Experiment 2

Ara que tenim un bon coneixement dels sets de dades i sabem com preparar-los correctament per a continuar millorant les nostres implementacions de xarxes neuronals, ara és el moment de buscar i experimentar amb diferents arquitectures, ja sigui provar vàries funcions d'activació, operacions de convolució, profunditats i amplades, etc.

L'objectiu d'aquest experiment és implementar i provar múltiples arquitectures de xarxes neuronals convolucionals que puguin donar els millors resultats possibles.

Totes les arquitectures tindran com a entrenament, validació i test les mateixes dades:

Taula 26: Dades PDC, experiment 2

Pneumonia Detection Challenge			
	Entrenament	Test	Validació
Normal	4643	1558	16
Pneumònia	4536	1476	4

Taula 27: Dades CXR, experiment 2

Chest X-Ray			
	Entrenament	Test	Validació
Normal	1341	234	8
Pneumònia	1938	390	8

Totes aquestes arquitectures estan connectades amb la següent xarxa neuronal: capa oculta (516 neurones) + capa oculta (256 neurones) + capa oculta (128 neurones) + capa oculta (64 neurones) + capa de sortida (1 neurona, funció sigmoide)

4.2.1. Arquitectura 1

L'arquitectura està formada per:

2 x Convolucions (kernels = 16) + Normalització + Reducció + 2 x Convolucions (kernels = 32) + Normalització + Reducció + 2 x Convolucions (kernels = 64) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció

L'element diferenciador d'aquesta arquitectura és que en comptes de tenir una capa de convolució abans de les reduccions ara n'afegirem una segona, amb l'objectiu de donar més profunditat al model.

- **Resultats dels entrenaments**

Taula 28: Entrenament experiment 2, arquitectura 1

Set de dades	Loss	Binary Accuracy	Precision	Recall	Èpoques	Temps (s)
PDC	0.5221	0.7393	0.7300	0.7445	50	4267
CXV	0.2017	0.9193	0.9450	0.9149	30	905

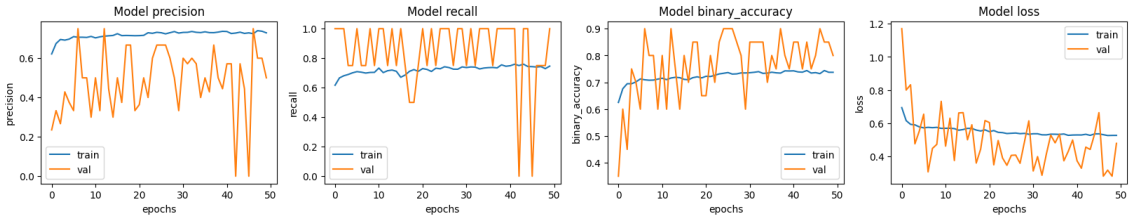


Figura 29: Gràfic de l'entrenament amb PDC experiment 2, arquitectura 1

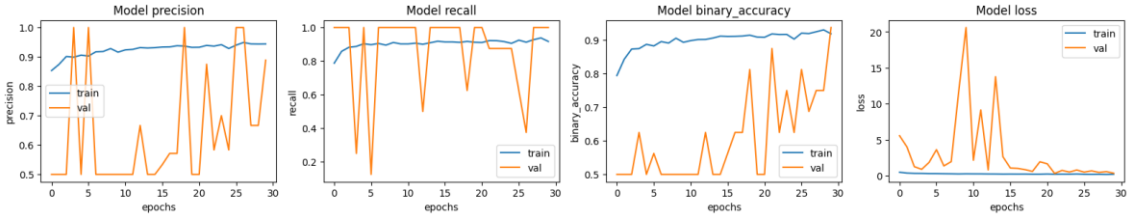


Figura 30: Gràfic de l'entrenament amb CXV experiment 2, arquitectura 1

- **Resultats dels test**

Taula 29: Resultats experiment 2, arquitectura 1

Set de dades	Loss	Binary Accuracy	Precision	Recall
PDC	0.5016	0.7614	0.7124	0.8543
CXV	0.2843	0.9022	0.9229	0.9205

Taula 30: Encerts experiment 2, arquitectura 1

	% d'encerts (encerts/total)	
Set de dades	**Pneumònia**	**Normal**
PDC	85.43	63.33
CXV	92.05	87.18

- **LIME**

Els resultats de LIME són de PDC i CXR respectivament, les imatges de la primera fila són casos amb pneumònia i la de sota normals.

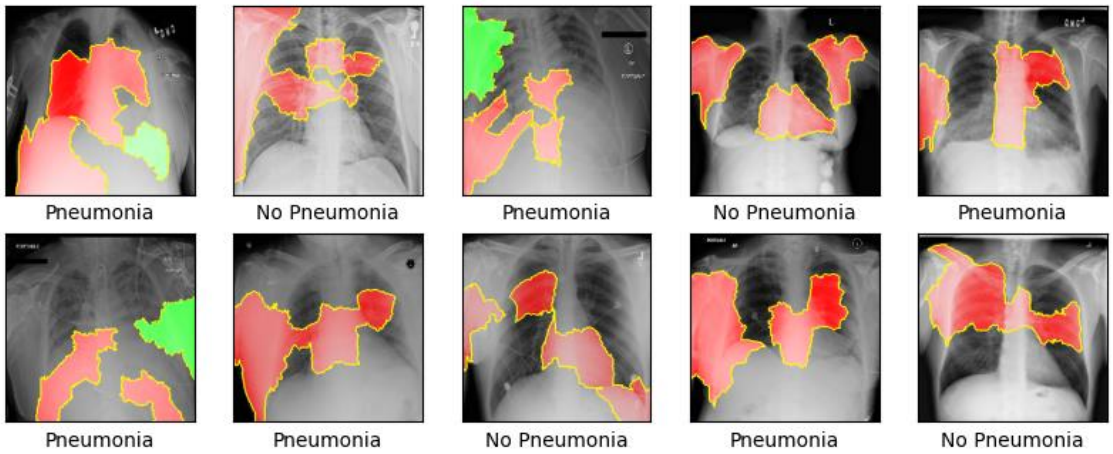


Figura 31: Resultat de LIME amb PDC experiment 2, arquitecura 1

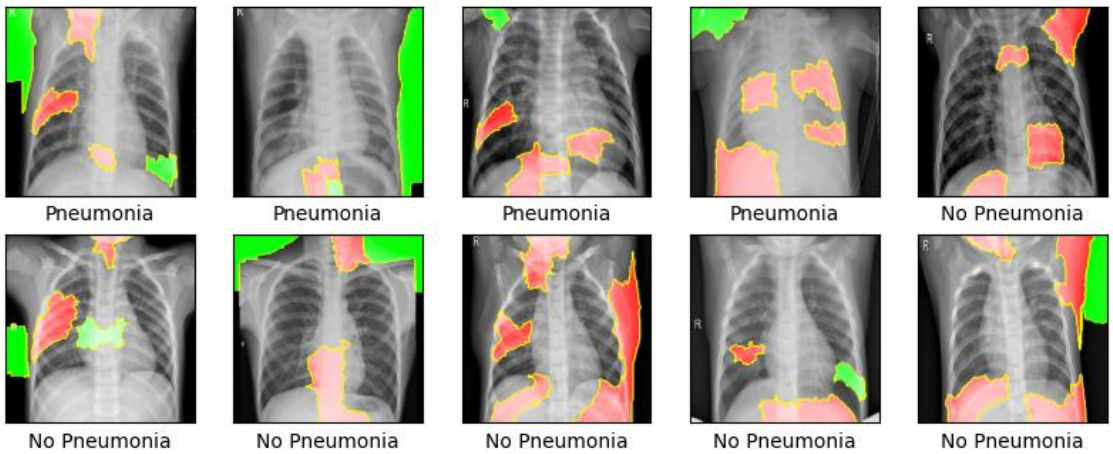


Figura 32: Resultat de LIME amb CXV experiment 2, arquitecura 1

4.2.2. Arquitectura 2

La arquitectura està formada per:

2 x Convolucions (kernels = 16) + Normalització + Reducció + 2 x Convolucions (kernels = 32) + Normalització + Reducció + 2 x Convolucions (kernels = 64) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció + 2 x Convolucions (kernels = 128) + Normalització + Reducció

Aquesta arquitectura és igual a la primera arquitectura, però amb una diferència, i és que en comptes d'utilitzar la funció d'activació ReLu utilitzarem la funció LeackyReLu [20], que resumidament, és una variant de la ReLu que permet tenir un major aprenentatge, ja que evita la mort de neurones.

- **Resultats dels entrenaments**

Taula 31: Entrenament experiment 2, arquitectura 2

Set de dades	Loss	Binary Accuracy	Precision	Recall	Èpoques	Temps (s)
PDC	0.5219	0.7455	0.7375	0.7489	50	4156
CXV	0.2162	0.9148	0.9392	0.9140	30	935

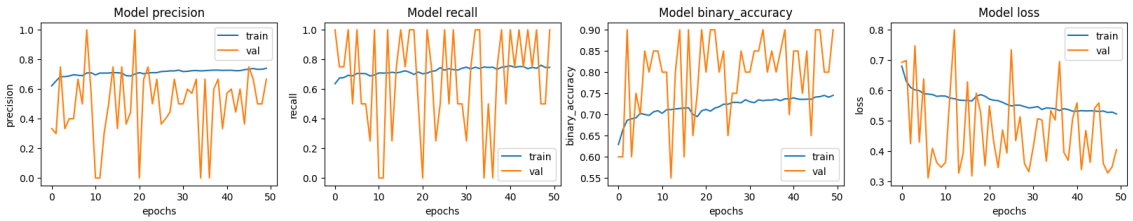


Figura 33: Gràfic de l'entrenament amb PDC experiment 2, arquitectura 2

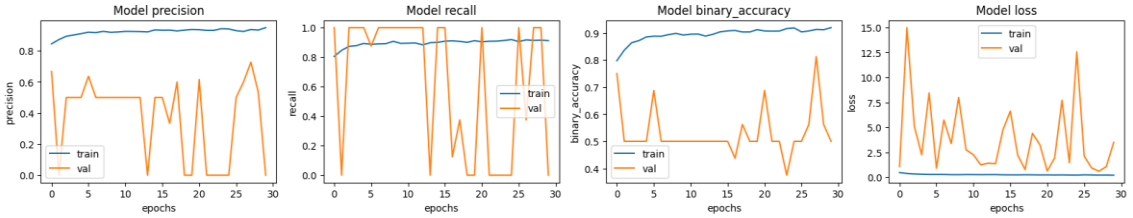


Figura 34: Gràfic de l'entrenament amb CXV experiment 2, arquitectura 2

- **Resultats dels test**

Taula 32: Resultats experiment 2, arquitectura 2

Set de dades	Loss	Binary Accuracy	Precision	Recall
PDC	0.4841	0.7703	0.7792	0.7364
CXV	3.0665	0.4038	0.9091	0.0513

Taula 33: Encerts experiment 2, arquitectura 2

	% d'encerts (encerts/total)	
Set de dades	Pneumònia	Normal

PDC	73.64	80.23
CXV	5.13	99.15

- **LIME**

Els resultats de LIME són de PDC i CXV respectivament, les imatges de la primera fila són casos amb pneumònia i la de sota normals.

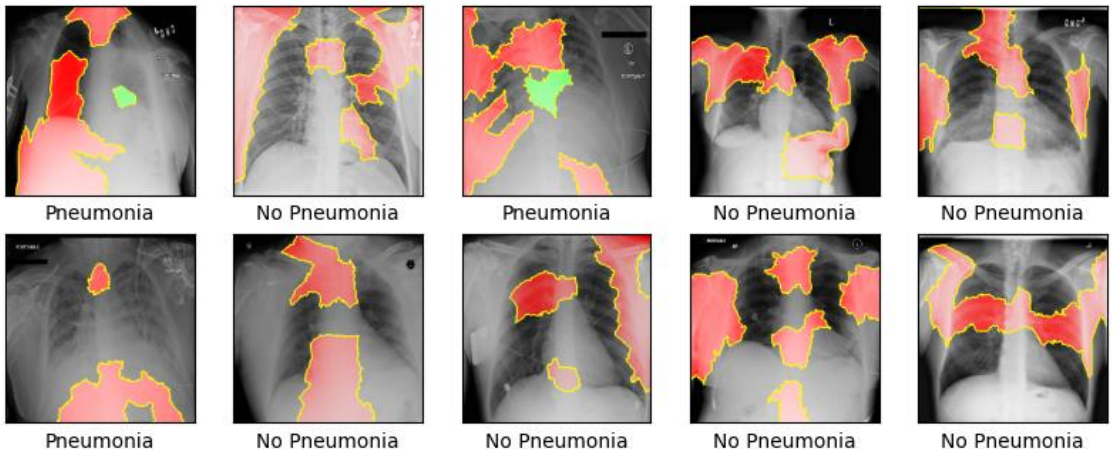


Figura 35: Resultat de LIME amb PDC experiment 2, arquitecura 2

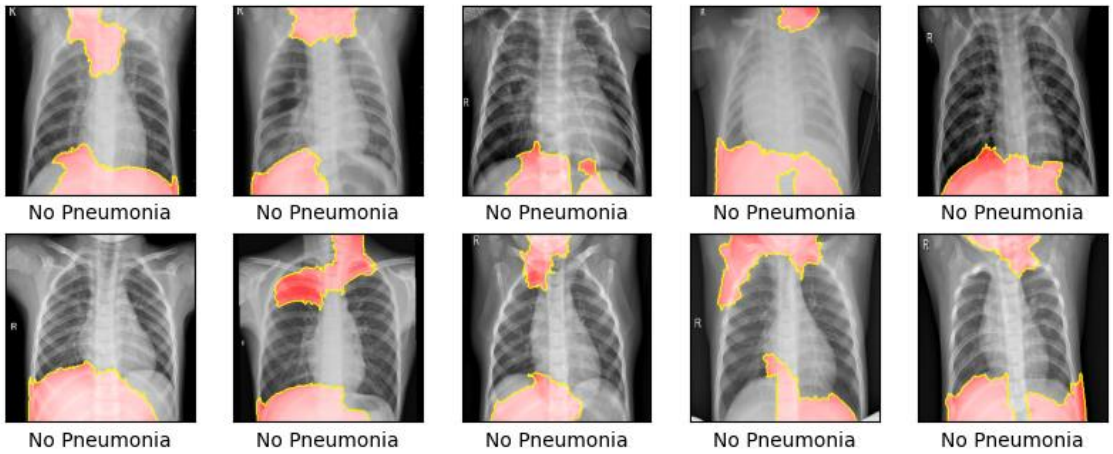


Figura 36: Resultat de LIME amb CXV experiment 2, arquitecura 2

4.2.3. Arquitectura 3

La arquitectura està formada per:

3 x Convolucions (kernels = 32) + Normalització + Reducció + 3 x Convolucions (kernels = 64) + Normalització + Reducció + 3 x Convolucions (kernels = 128) + Normalització + Reducció + 3 x Convolucions (kernels = 256) + Normalització + Reducció + 3 x Convolucions (kernels = 256) + Normalització + Reducció

Aquest cop en l'arquitectura hem afegit una tercera capa convolucional augmentant encara mes la profunditat, també hem augmentat l'amplada del model augmentant el número de kernels en cada convolució.

- **Resultats dels entrenaments**

Taula 34: Entrenament experiment 2, arquitectura 3

Set de dades	Loss	Binary Accuracy	Precision	Recall	Èpoques	Temps (s)
PDC	0.5376	0.7386	0.7385	0.7272	50	4207
CXV	0.2104	0.9207	0.9446	0.9192	30	905

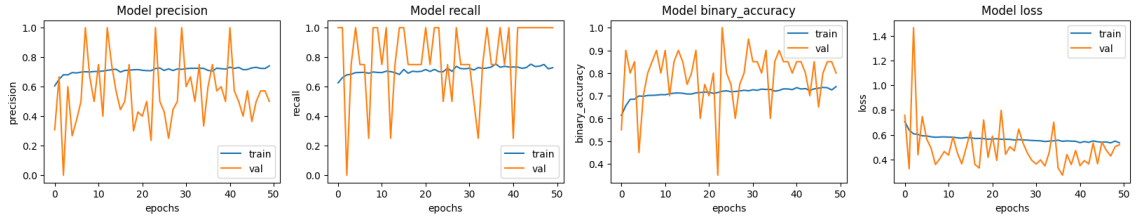


Figura 37: Gràfic de l'entrenament amb PDC experiment 2, arquitectura 3

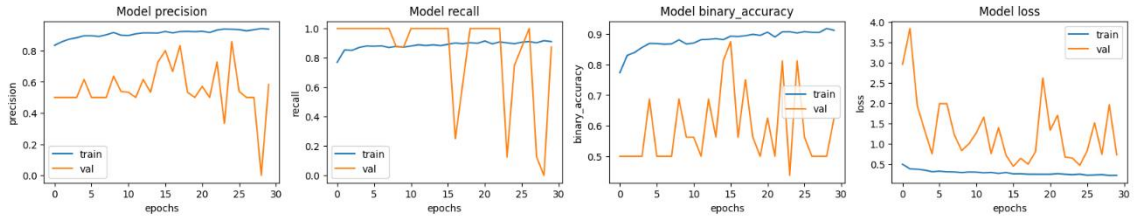


Figura 38: Gràfic de l'entrenament amb CXV experiment 2, arquitectura 3

- **Resultats dels test**

Taula 35: Resultats experiment 2, arquitectura 3

Set de dades	Loss	Binary Accuracy	Precision	Recall
PDC	0.5121	0.7574	0.7065	0.8577
CXV	0.4037	0.8333	0.8611	0.8744

Taula 36: Encerts experiment 2, arquitectura 3

	% d'encerts (encerts/total)	
Set de dades	**Pneumònia**	**Normal**
PDC	85.77	66.24
CXV	87.44	76.50

- **LIME**

Els resultats de LIME són de PDC i CXR respectivament, les imatges de la primera fila són casos amb pneumònia i la de sota normals.

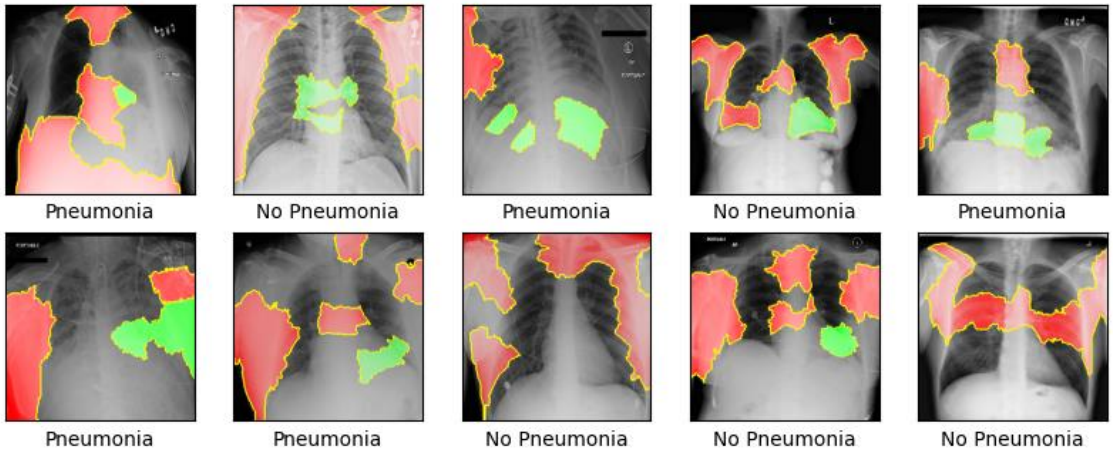


Figura 39: Resultat de LIME amb PDC experiment 2, arquitecura 3

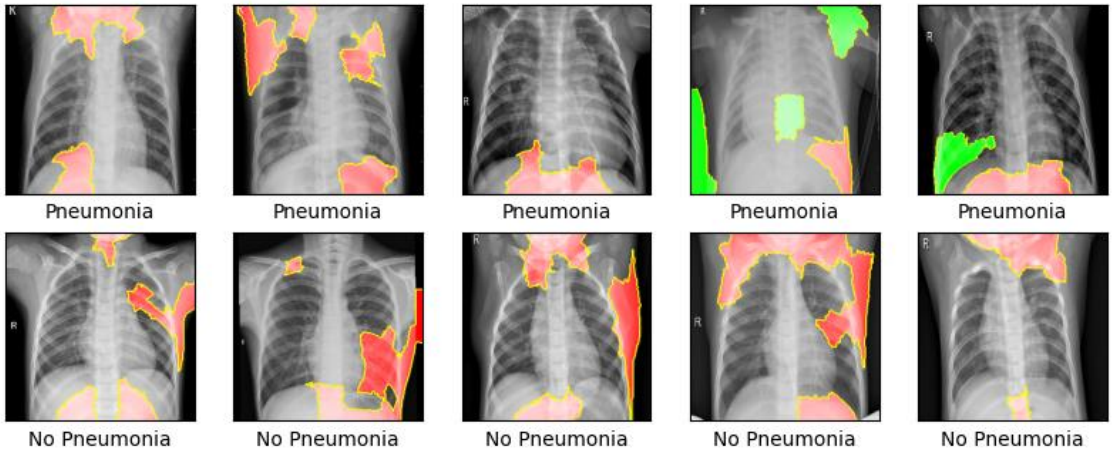


Figura 40: Resultat de LIME amb CXV experiment 2, arquitecura 3

4.2.4. Arquitectura 4

L'arquitectura està formada per:

2x Convolució (kernels = 16) + Normalització + Reducció + Convolució en profunditat (kernels = 32) + Convolucions (kernels = 32) + Normalització + Reducció + Convolució en profunditat (kernels = 64) + Convolucions (kernels = 64) + Normalització + Reducció + Convolució en profunditat (kernels = 128) + Convolucions (kernels = 128) + Normalització + Reducció + Convolució en profunditat (kernels = 128) + Convolucions (kernels = 128) + Normalització + Reducció

Per aquesta última arquitectura hem afegit una capa nova anomenada convolució en profunditat [21], la qual és un tipus de convolució en la que s'aplica un únic filtre convolucional per cada canal d'entrada.

- **Resultats dels entrenaments**

Taula 37: Entrenament experiment 2, arquitectura 4

Set de dades	Loss	Binary Accuracy	Precision	Recall	Èpoques	Temps (s)

PDC	0.5549	0.7328	0.7231	0.7289	80	24174
CXV	0.2374	0.9031	0.9266	0.9088	50	5505

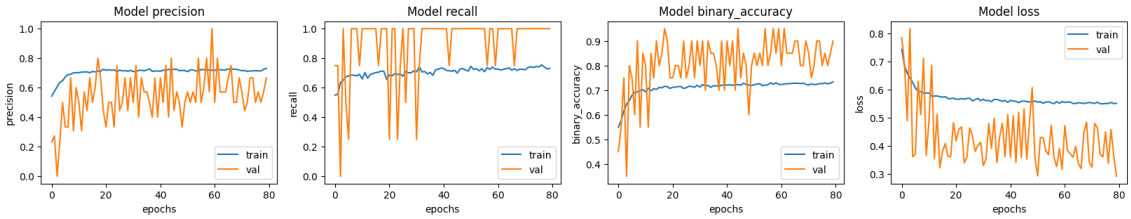


Figura 41: Gràfic de l'entrenament amb PDC experiment 2, arquitectura 4

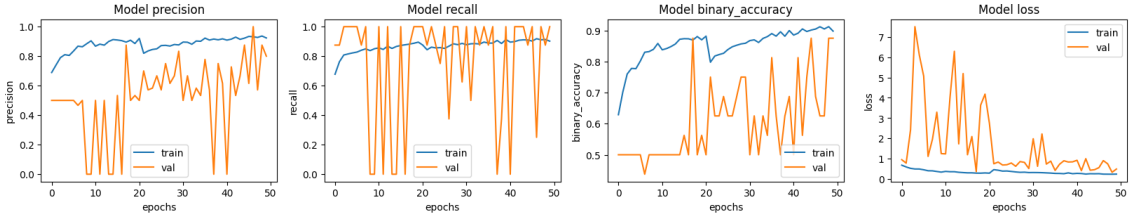


Figura 42: Gràfic de l'entrenament amb CXV experiment 2, arquitectura 4

- **Resultats dels test**

Taula 38: Resultats experiment 2, arquitectura 4

Set de dades	Loss	Binary Accuracy	Precision	Recall
PDC	0.5263	0.7432	0.7576	0.6944
CXV	0.3117	0.8798	0.8689	0.9513

Taula 39: Encerts experiment 2, arquitectura 4

	% d'encerts (encerts/total)	
Set de dades	**Pneumònia**	**Normal**
PDC	69.44	78.95
CXV	95.13	76.07

- **LIME**

Els resultats de LIME són de PDC i CXR respectivament, les imatges de la primera fila són casos amb pneumònia i la de sota normals.

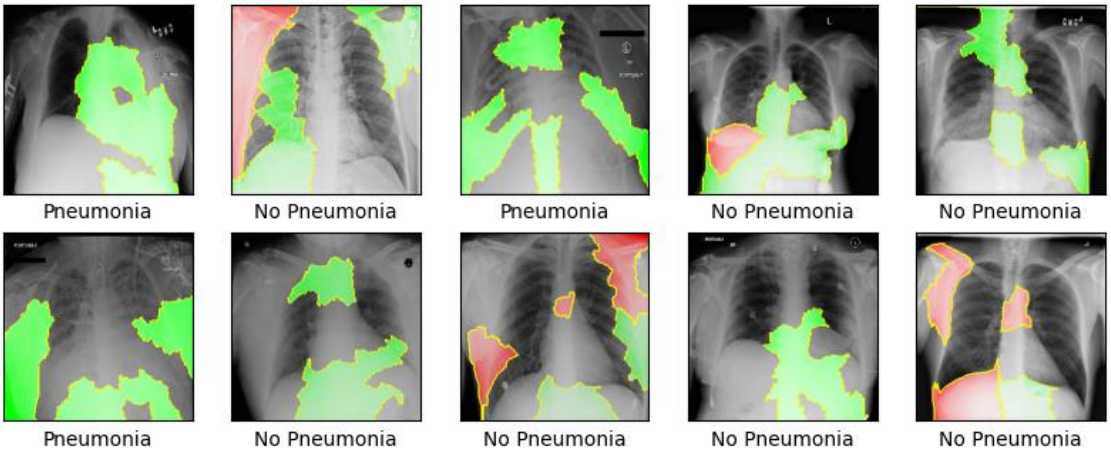


Figura 43: Resultat de LIME amb PDC experiment 2, arquitecura 4

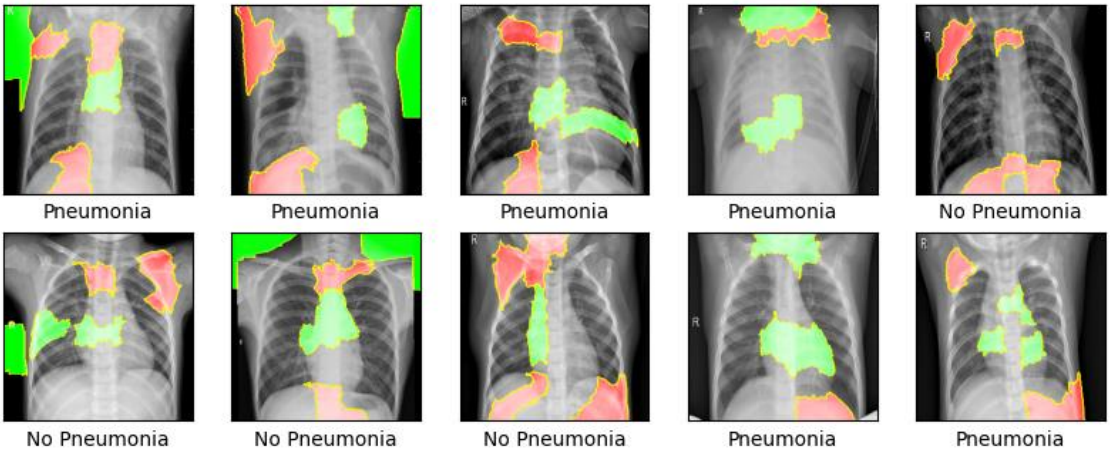


Figura 44: Resultat de LIME amb CXV experiment 2, arquitecura 4

4.3. Experiment 3

Després d'haver provar vàries arquitectures de xarxes neuronals convolucionals, com a últim experiment ens interessa provar una arquitectura ja creada per experts en el sector. L'arquitectura que provarem serà la VGG16 preentrenada amb el set de dades Imagnet [22].

Perquè nosaltres puguem entrenar amb els nostres sets de dades de pneumònies un model ja preentrenat, utilitzarem la tècnica de "transfer learning" [23] que consisteix a congelar l'entrenament d'algunes capes mentre les altres sí que s'entrenaran, llavors nosaltres congelarem totes les capes convolucionals i entrenament les capes posteriors a la capa d'entrada.

L'objectiu d'aquest experiment és veure quins resultats obtenim amb un gran model preentrenat amb un set de dades gegant, que sabem que dona bons resultats, i entrenar una part amb els nostres sets de dades.

El model tindrà com a entrenament, validació i test les següents dades:

Taula 40: Dades PDC, experiment 3

Pneumonia Detection Challenge

	Entrenament	Test	Validació
Normal	4643	1558	16
Pneumònia	4536	1476	4

Taula 41: Dades CXR, experiment 3

Chest X-Ray			
	Entrenament	Test	Validació
Normal	1341	234	8
Pneumònia	1938	390	8

El model VGG16 està connectat amb la següent xarxa neuronal: capa oculta (1000 neurones).

- **Resultats dels entrenaments**

Taula 42: Entrenament experiment 3

Set de dades	Loss	Binary Accuracy	Precision	Recall	Èpoques	Temps (s)
PDC	0.5473	0.7248	0.7162	0.7346	15	1244
CXV	0.1391	0.9459	0.9544	0.9527	15	457

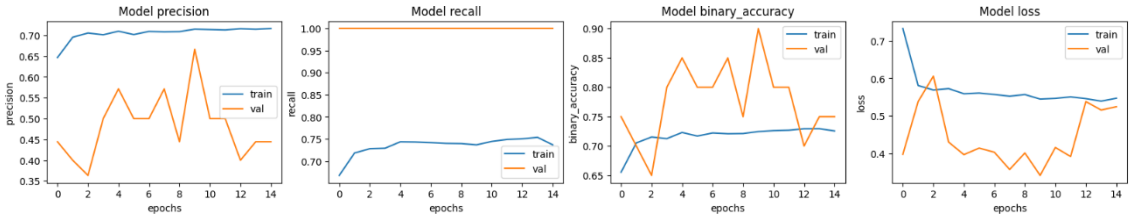


Figura 45: Gràfic de l'entrenament amb PDC experiment 3

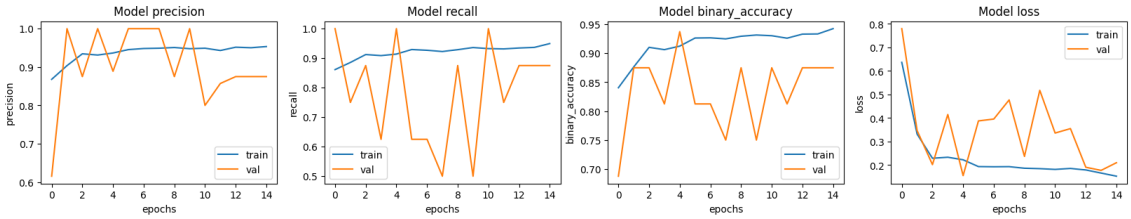


Figura 46: Gràfic de l'entrenament amb CXV experiment 4

- **Resultats dels test**

Taula 43: Resultats experiment 3

Set de dades	Loss	Binary Accuracy	Precision	Recall
PDC	0.5115	0.7472	0.7050	0.8259
CXV	0.2184	0.9135	0.9516	0.9077

Taula 44: Encerts experiment 3

	% d'encerts (encerts/total)

Set de dades	Pneumònia	Normal
PDC	90.56	67.27
CXV	90.80	92.31

- **LIME**

Els resultats de LIME són de PDC i CXR respectivament, les imatges de la primera fila són casos amb pneumònia i la de sota normals.

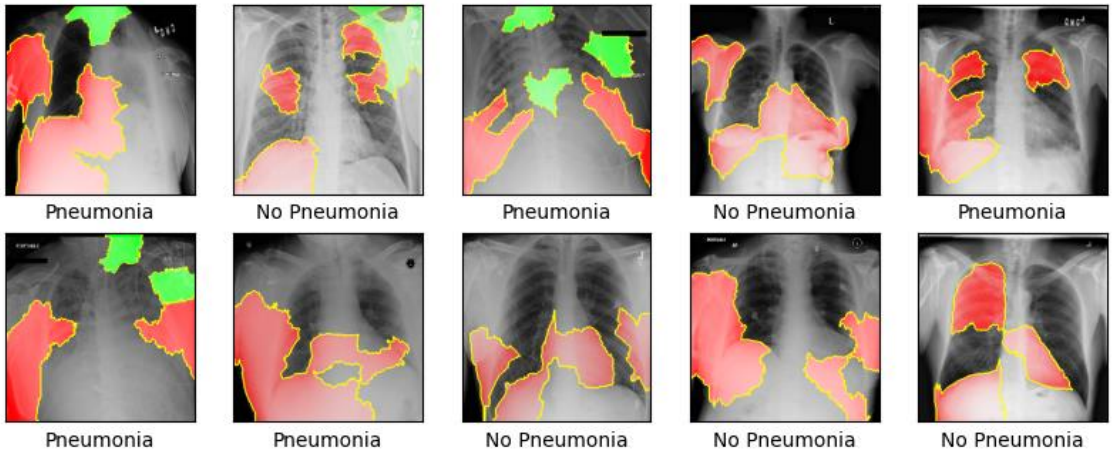


Figura 47: Resultat de LIME amb PDC experiment 3

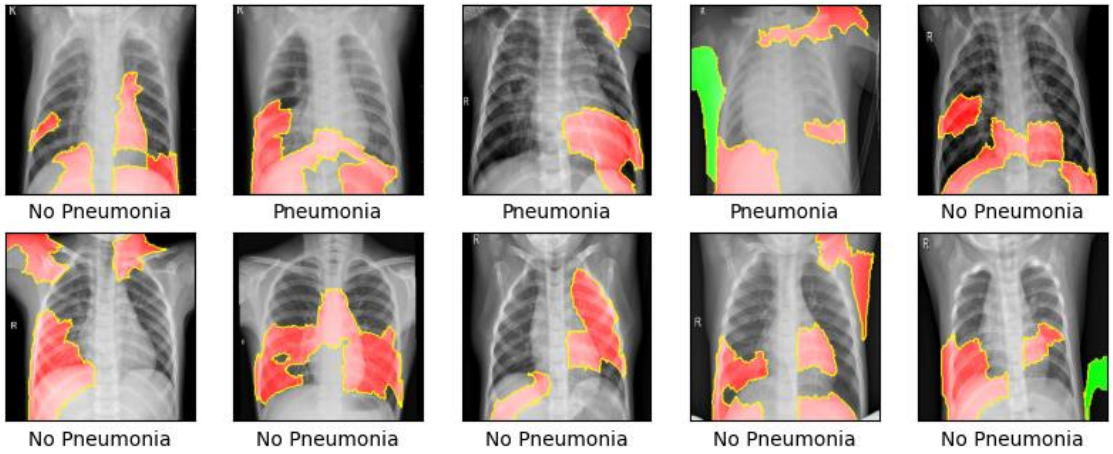


Figura 48: Resultat de LIME amb CXV experiment 3

5. Conclusions

5.1. Experiment 1

Ja tenint els resultats de les tres fases del primer experiment, anem a fer l'anàlisi dels resultats obtinguts. Primer analitzarem una a una les tres fases, pel que fa al tractament del set de dades, quines eren les seves expectatives i quins han set els seus resultats finalment, i per últim veurem quines són les conclusions del primer experiment respecte al comportament dels models segons la seva profunditat avaluant els seus entrenaments i resultats.

- **Fase 1**

  En aquesta primera fase teníem com a objectiu comprovar si al no fer cap mena de preprocessat ni tractament dels sets de dades, tenint casos de pneumònia descompensats, es provocaria un sobre ajustament dels models causant un biaix en els resultats, sigui en una direcció o en l'altra.

  Els resultats que hem obtingut en aquestes proves són molt clars, en les taules [Taula 7] i [Taula 9] podem veure el percentatge d'encerts dels models, de casos positius i negatius. En els dos sets de dades s'observa un clar biaix.

  En el cas del set de dades PDC, els casos negatius són 15477 i 4536 casos positius, tres cops més casos positius, tots els models tenen un percentatge d'encert d'un 90% en els casos negatius, i un 40% en els casos positius, això ens confirma que els models tenen una tendència molt forta per decantar-se, en les prediccions, a favor dels casos negatius. Pel que fa al set de dades CXR, els casos negatius són 1341 i 3875 casos positius, aquest cop els resultats són al revés dels del PDC, ara tots els models tenen un percentatge d'encert d'un 99% en els casos positius i un 30% en els casos negatius.

  En l'explicació de LIME, imatges [Figura 23] i [Figura 24], es veu com els models tenen molta tendència a remarcar formes bastant aleatòries, ja sigui parts de la columna, estómac, vores... En aquesta fase sembla que no és capaç de trobar cap patró de pneumònies.

  Amb aquests resultats podem confirmar que un set de dades descompensat ens provocarà un sobre ajustament dels models provocant un biaix en la direcció que hi hagi més casos.

- **Fase 2**

  En la segona fase del primer experiment, l'objectiu era fer un preprocessat dels sets de dades, concretament, equilibrar els casos negatius o positius rebaixant els casos majoritaris, amb això volíem comprovar que el biaix, que es provocava amb un set de dades descompensat, es redueix.

  Els resultats que hem obtingut en aquestes proves són els esperats, en les taules [Taula 15] i [Taula 17] podem veure el percentatge d'encerts dels models, de casos positius i negatius. En els dos sets de dades s'observa una reducció del biaix.

  Aquesta vegada el nombre d'imatges per al set de dades PDC són de 4643 casos normals i 4536 casos amb pneumònia, amb aquestes dades els resultats d'encert per a casos positius i negatius són tots al voltant de 70%, el qual significa que hem aconseguit eliminar el biaix en el test. Pel que fa al set de dades CXR , els casos negatius són 1341 i 1938 casos positius, aquest cop els resultats no estan tan equilibrats com en el PDC, ja que el percentatge d'encerts en els casos positius és d'un 98%, i per als casos negatius un 45%, sí que podem

observar que per als casos negatius hem passat d'un 30%, que teníem a la fase 1, a un 45%, però continua havent-hi massa biaix en els resultats.

En l'explicació de LIME, imatges [Figura 25] i [Figura 26], a diferència de l'anterior fase, es veu com els models, en alguns casos, miren la part dels pulmons, no obstant però, continuen mirant formes aleatòries i no acaben de mirar les dues sones on estan ubicats els pulmons sinó que miren una part o cap.

Avaluant els resultats que hem obtingut en aquesta fase, i comparant-los amb els de la fase anterior, podem determinar que el fet d'equilibrar les dades en l'entrenament de xarxes neuronals convolucionals imprescindible per evitar que el model tendeixi més en favor d'un resultat que d'altres.

- **Fase 3**

  En l'última fase del primer experiment teníem com a objectiu evitar el sobre ajustament dels models en l'etapa d'entrenament que era provocat per una escassa varietat en les imatges. Les imatges dels nostres sets de dades són de radiografies de tòrax, això significa que hi haurà unes formes en les imatges que es repetiran sempre, com la columna vertebral, bressos, estómac, entre altres, probablement el model dona prioritat en observar aquestes formes que es repeteixen i intenta buscar algun patró, això ho hem d'evitar augmentant les imatges.

  Novament, en aquesta ocasió, els resultats d'aquest experiment han set molt satisfactoris respecte a l'objectiu d'aquest, en les taules [Taula 22] i [Taula 24] podem veure el percentatge d'encerts dels models, de casos positius i negatius. En els dos sets de dades s'observa una major reducció del biaix a més d'uns resultats millors en els tests.

  Pel que fa al set de dades CXR, hem passat d'un 45% d'encerts en els casos negatius, a un 85% mantenint uns encerts en els casos positius molt bons (95% de mitja). I en el cas de PDC hem aconseguit augmentar al voltant d'un 5% d'encerts, tant en els casos positius com els negatius.

  En l'explicació de LIME, imatges [Figura 27] i [Figura 28], en aquest últim experiment, es veu com els models segueixen tenint el mateix comportament que els de la segona fase, però si que és cert que en les imatges del set de dades de PDC sembla que encerta bastant la posició la qual ha d'observar per trobar pneumònies, almenys per determinar que no n´hi han.

  Analitzant els resultats podem concloure que aplicar tècniques d'augment de dades evitem que el model s'aprengui el set d'entrenament donant resultats propers al 95% d'encerts, però quan es passa el set de test, els resultats cauen entre un 10% i un 40%.

Per resumir i donar una visió global del primer experiment, en la figura [Figura 49] hem resumit en un gràfic on apareixen els percentatges d'encert dels dos sets de dades en la cinquena arquitectura de cada una de les fases, podem observar que a mesura en què avancem les fases, la taxa d'encert en els dos casos es va equilibrant fins al punt que queden pràcticament al mateix nivell. Aquest resultat és molt bo de cara als següents experiments, ja que ara sabem que si s'aplica augment de les dades aconseguim que no hi hagi biaix en els models i que aquestes tampoc s'aprendran el set de dades d'entrenament, cosa que és un punt de partida imprescindible per a millorar els models.

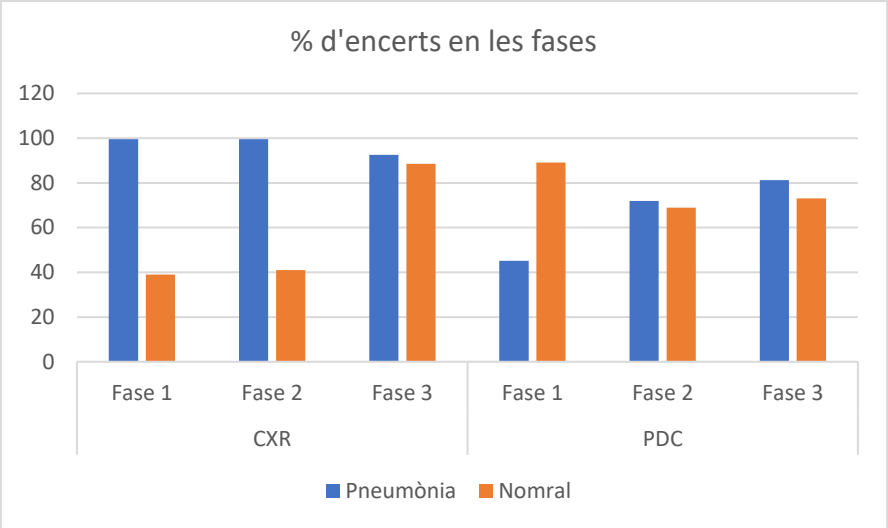


Figura 49: Percentatge d'encerts en les fases de l'experiment 1

Per acabar de confirmar els beneficis de fer un preprocessat i augmentament de les dades, en les figures [Figura 50] i [Figura 51] podem veure com, en cada fase, totes les arquitectures van millorant la seva exactitud (Accuracy).

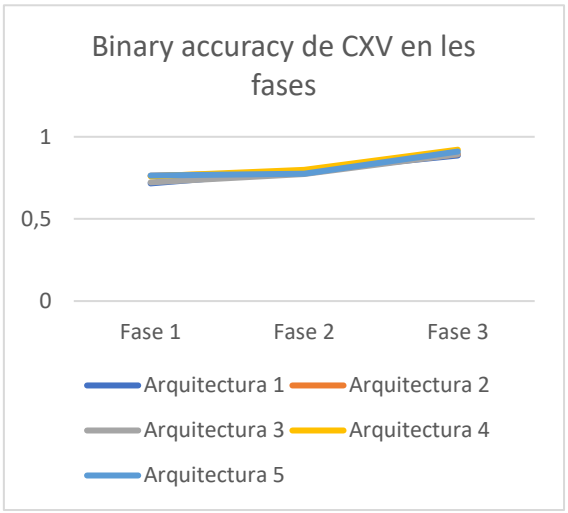


Figura 51: Binary Accuracy de CXV

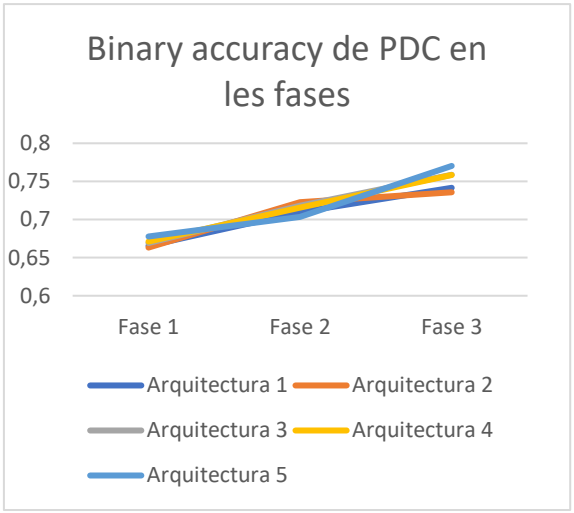


Figura 50: Binary Accuracy de PDC

Anant ja al segon objectiu de l'experiment, analitzar l'efecte de la profunditat de les xarxes neuronals convolucionals en els entrenaments i els resultats, hem pogut observar diferents aspectes en els quals la profunditat afecta.

- **Resultats de les prediccions**

  Tant amb el set de dades CXR en les taules [Taula 6], [Taula 14] i [Taula 22], i el PDC en les taules [Taula 8], [Taula 16] i [Taula 24], en general, a mesura que afegim més profunditat en els models, els resultats milloren fins a un 3%, però en les explicacions de LIME no s'observa que els models observin patrons diferents.

  Una major profunditat hauria de ser capaç de trobar patrons més patrons i únics en les imatges i en els nostres resultats sí que es veu una millor classificació dels resultats.

- **Resultats dels entrenaments**

En els entrenaments de la primera i la segona fase del primer experiment, en les taules [Taula 4], [Taula 5], [Taula 12] i [Taula 13], es pot veure que a mesura que augmenta la profunditat, els resultats són pitjors, això podria ser degut al fet que els models amb menor profunditat són capaços d'aprendre amb una major facilitat sets de dades d'entrenament petits, això dona sentit a l'apartat anterior, ja que els models poc profunds en aprendre's més fàcilment els sets de dades d'entrenament, donen pitjors resultats en els tests.
El cas de la fase tres, [Taula 20] i [Taula 21], encara confirma aquesta teoria, perquè en fer l'augment d'imatges, els models menys profunds perden aquesta capacitat d'acostumar-se i aprendre's les dades d'entrenament perquè les imatges són massa variades per aquests models.

- **Capacitat d'aprenentatge,**

  En l'anterior punt em vist que els models poc profunds s'aprenen el set d'imatges, això també ho podem confirmar analitzant els gràfics d'entrenament. En la figura [Figura 52] podem veure l'entrenament del primer experiment en la segona fase i del set de dades PDC de l'arquitectura 1, i en la figura [Figura 53] l'arquitectura 5:

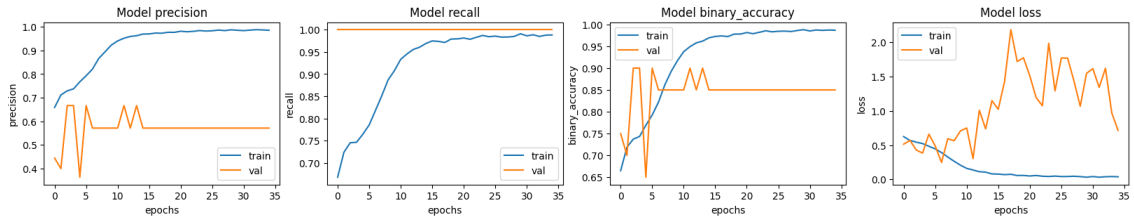


Figura 52: Entrenament amb PDC, experiment 1, fase 2, arquitectura 1

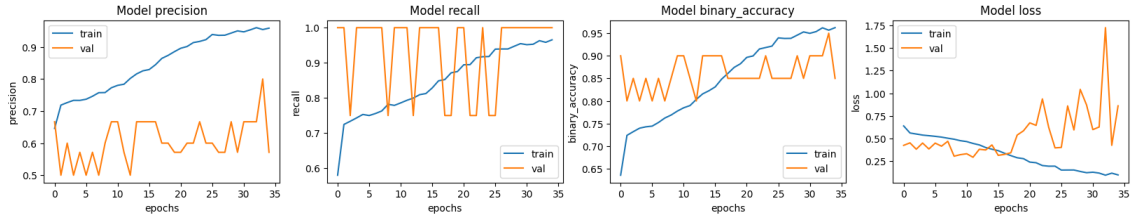


Figura 53: Entrenament amb PDC, experiment 1 fase 2, arquitectura 5

Hem escollit aquest entrenament, ja que es veu molt fàcilment un patró que es repeteix en els altres, i és que en la figura [Figura 52], la qual és de l'arquitectura 1, el seu aprenentatge és molt més ràpid comparat amb el de la figura [Figura 53] la qual és l'arquitectura 5, això té molt sentit, perquè l'arquitectura 1 en tenir poca profunditat no té tants paràmetres que puguin aprendre a diferència de la 5, provocant aquesta major velocitat d'aprenentatge, i el que fa és aprendre's les imatges globals en comptes dels patrons que hi puguin haver.

5.2. Experiment 2

Aquest experiment tenia la finalitat de buscar i provar diferents arquitectures de xarxes neuronals convolucionals amb l'objectiu de trobar la que millors resultats doni. Anem a analitzar els resultats obtinguts en les quatre arquitectures que hem posat a prova.

- **Arquitectura 1**

L'element característic d'aquesta arquitectura era l'addició d'una segona capa de convolució a totes les que ja teníem, quedant una arquitectura de dues convolucions més reducció per a cada nivell.

Els entrenaments en els dos casos s'estanquen en poques èpoques: en el cas del PDC l'entrenament un cop arriba a l'època 8 amb un 70.06% de Accuracy i avança fins al 73.93% en l'època 50, i per al CXR, l'entrenament un cop arriba a l'època 15 amb un 90.78% de Accuracy i avança fins al 91.93% en l'època 30, taula [Taula 28].

Els resultats en les prediccions que dona aquest model són bastant bons en els dos sets de dades, tenint: en la taula [Taula 30] en el PDC un percentatge d'encerts del 63.33% en els casos normals i un 85.43% en els casos amb pneumònia, i en el CXR un percentatge d'encerts del 87.18% en els casos normals i un 92.05% en els casos amb pneumònia.

Com hem dit són uns bons resultats, però aquest model no ha set capaç de millorar els resultats que teníem en el primer experiment i pel que fa als entrenaments, són també molt similars que els de la fase 3 del primer experiment.

- **Arquitectura 2**

  Aquesta arquitectura és idèntica a la primera, però amb l'element característic del canvi de la funció d'activació ReLu per LeackyReLu en totes les capes del model.

  Els entrenaments en els dos casos s'estanquen en poques èpoques: en el cas del PDC l'entrenament un cop arriba a l'època 6 amb un 70.03% de Accuracy i avança fins al 7.455% en l'època 50, i per al CXR, l'entrenament un cop arriba a l'època 15 amb un 91.36% de Accuracy i avança fins al 91.48% en l'època 30, taula [Taula 31].

  Els resultats en les prediccions que dona aquest model són estranys, per que en el cas de CXV en vàries probes que hem fet els resultats han set dolents, però per al cas del PDC, són els millors d'entre els 4 models per aquest set de dades. Tenim: en la taula [Taula 33] en el PDC un percentatge d'encerts del 80.23% en els casos normals i un 73.64% en els casos amb pneumònia, i en el CXR un percentatge d'encerts del 5.13% en els casos normals i un 99.15% en els casos amb pneumònia.

  En el cas del set de dades CXR els resultats són descartables perquè el model clarament tendeix a predir no pneumònia, ara bé, per al PDC la Accuracy i la Precision són els millors resultats. Aquest fet ens fa pensar que l'ús de la funció d'activació LeackyReLu requereix un set de dades gran per a poder-se entrenar bé.

- **Arquitectura 3**

  L'element característic d'aquesta arquitectura era l'addició d'una tercera capa de convolució a totes les que ja teníem en la primera arquitectura, a part d'augmentar la profunditat del model, també vam augmentar l'horitzontalitat afegint més kernels cada capa de convolució.

  Com en la primera arquitectura, els entrenaments en els dos casos s'estanquen en poques èpoques: en el cas del PDC l'entrenament un cop arriba a l'època 8 amb un 69.76% de Accuracy i avança fins al 73.86% en l'època 50, i per al CXR, l'entrenament un cop arriba a l'època 20 amb un 90.04% de Accuracy i avança fins al 92.07% en l'època 30, taula [Taula 34].

Els resultats en les prediccions que dona aquest model són bastant bons en els dos sets de dades, tenint: en la taula [Taula 36] en el PDC un percentatge d'encerts del 66.24% en els casos normals i un 85.77% en els casos amb pneumònia, i en el CXR un percentatge d'encerts del 76.50% en els casos normals i un 87.44% en els casos amb pneumònia, resultats inferiors als del primer model.

Com hem dit són uns bons resultats, però aquest model tampoc ha set capaç de millorar els resultats que teníem en el primer experiment. L'objectiu d'aquest model era el d'augmentar la capacitat d'entrenament del model augmentant la seva mesura, però els resultats no ho reflecteix quedant-se una mica per enrere.

- **Arquitectura 4**

  Per acabar, la quarta arquitectura vam provar una capa de convolució diferent anomenada convolució profunda, per veure que tal es comportava en l'entrenament i quins resultats que donava.

  Aquest últim model ha tingut un entrenament similar als altres, els entrenaments en el cas del PDC s'estanca en poques èpoques, però el CXV té un entrenament més progresiu: en el cas del PDC l'entrenament un cop arriba a l'època 10 amb un 69.97% de Accuracy i avança fins al 73.28% en l'època 80, i per al CXR, l'entrenament arriba al 90.31% sense estancar-se, taula [Taula 37]. Cal destacar que hem augmentat el nombres d'èpoques en aquest model, ja que tardava més estona a aprendre i també a la taula [Taula 37] es pot veure com aquesta nova capa té un càlcul molt pesat, pel fet que el temps d'entrenament es dispara.

  Els resultats en les prediccions que dona aquest model encara són bastant bons en els dos sets de dades, tenint: en la taula [Taula 39] en el PDC un percentatge d'encerts del 78.95% en els casos normals i un 69.44% en els casos amb pneumònia, i en el CXR un percentatge d'encerts del 76.07% en els casos normals i un 95.13% en els casos amb pneumònia, resultats inferiors als del primer model.

  Un model amb aquesta nova capa dona bons resultats, però seguim sense millorar els resultats que teníem en el primer experiment.

Abans de tancar el segon experiment analitzem de forma global els resultats de les arquitectures:

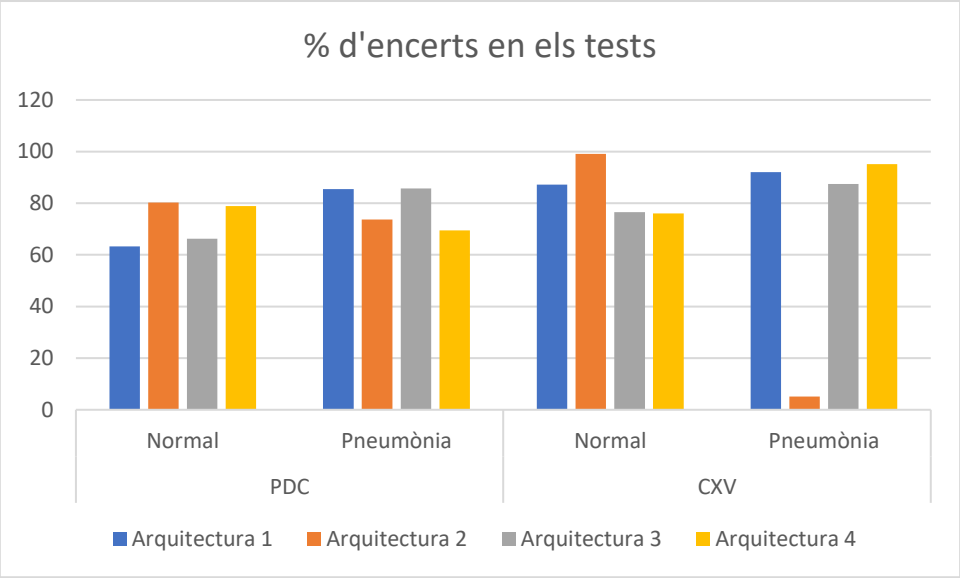


Figura 54: Percentatge d'encerts en l'experiment 2

En el gràfic de la figura [Figura 54] tenim resumits els resultats dels percentatges d'encerts en casos normals i amb pneumònia. Fàcilment, veiem que els quatre models donen resultats molt similars, tret de l'arquitectura 2 que ja hem comentat, i probablement si féssim diferents execucions de tots els models i calculéssim la mitja els resultats estarien equilibrats. Ara bé, tenint en compte que cap d'aquests 4 models ha set capaç de millorar els resultats que ja teníem en l'etapa final del primer experiment, hauríem de valorar altres tècniques d'enteniment, netejar els sets de dades d'entrenament i ampliar-los, o també aplicar "fine tuning" als nostres models, que consistiria resumidament a canviar paràmetres dels models com la ràtio d'aprenentatge, d'entre altres.

En els resultats de LIME tampoc es veu res destacable en comparació al primer experiment, però sí que podem destacar que es nota una diferència en les explicacions que es repeteix en tots els models, i és que per al set de dades PDC els models tenen unes observacions més generals analitzant àrees, en canvi, per al CXV, els models normalment tenen tendència per remarcar formes com costelles ossos de la columna vertebral, òrguens...

5.3. Experiment 3

Aquest experiment tenia l'objectiu provar l'arquitectura VGG16 preentrenada amb el set de dades Imagnet amb l'objectiu d'aplicar "transfer learning" congelant les capes convolucionals i entrenar les finals.

Els entrenaments en els dos casos s'estanquen en poques èpoques: en el cas del PDC l'entrenament un cop arriba a l'època 5 amb un 72.18% de Accuracy i avança fins al 72.48% en l'època 15, i per al CXR, l'entrenament un cop arriba a l'època 6 amb un 93.92% de Accuracy i avança fins al 94.59% en l'època 15, taula [Taula 42].

Els resultats en les prediccions que dona aquest model són bons en els dos sets de dades, tenint: en la taula [Taula 44] en el PDC un percentatge d'encerts del 67.27% en els casos normals i un 90.56% en els casos amb pneumònia, i en el CXR un percentatge d'encerts del 92.31% en els casos normals i un 90.80% en els casos amb pneumònia.

En els resultats que ens dona LIME encara veiem com el model en algunes imatges sembla que es feixa en les dues zones dels pulmons, però en altres imatges fa la predicció mirant parts exteriors de cos o parts diferents de les dels pulmons.

Els resultats de l'arquitectura VGG16 són uns resultats molt similars als de segon experiment, és a dir, provant una arquitectura ja provada i coneguda preentrenada amb un set de dades gegant més els nostres, no ha suposat cap avantatge respecte a les anteriors proves que hem fet, ara bé un punt a favor d'aquest model el tenim en l'entrenament perquè com hem pogut veure amb poc més de 5 èpoques el model ja es capaç d'obtenir els resultats que tenien els altres en bastants més.

5.4. Conclusions finals

Ja tenim tots els resultats i les seves respectives anàlisis i ens agradaria aprofitar aquest últim apartat per resumir i fer unes conclusions finals de tots els experiments que hem fet i valorar els resultats.

El primer experiment és molt esclaridor, ja que mirant les dades que han donat els resultats, és evident la seva millora en cada fase, i és que gràcies a aquest experiment podem concloure que és imprescindible fer un bon preprocessament del set de dades que utilitzarem per entrenar el nostre model per evitar el biaix equilibrant els casos i, tanmateix, evitar el sobre ajustament del model aplicant diferents tècniques d'augment d'imatges. Ara bé, ens hem deixat moltes altres tècniques per tractar les imatges d'entrenament, com per exemple, fer una neteja exhaustiva d'imatges descarta-les, aplicar filtres per eliminar possible soroll, augmentar tot el possible el set, etc.

L'altre aspecte important del primer experiment és la profunditat dels models, les conclusions que podem treure són que, com més profunditat millor, perquè un model molt profund té una major capacitat d'aprenentatge, i, per tant, una major capacitat de trobar característiques en les imatges més complexes, nosaltres hem provat imatges amb una mida de 256 x 256 píxels, el qual ens limita a una certa profunditat, però podríem augmentar la mida de les imatges i d'aquesta forma augmentar també la profunditat dels models, el qual causaria un augment de la mida del set de dades augmentant també el temps d'entrenament dels models.

En el segon experiment hem de reconèixer que esperàvem uns resultats diferents, i és que en cap dels quatre models que hem provat, els resultats han set millors que els del primer experiment, en el cas del set CXV no creiem que es puguin millorar els resultats, ja que hem arribat a tenir un percentatge d'encert d'un 91%, el qual és un percentatge bastant elevat, però per al cas del set PDC la cosa canvia, sent el millor resultat un 75% el qual és molt millorable. Aquest fet ens fa pensar que tenim algun problema en el set d'entrenament, perquè els models no són capaços de trobar l'element característic de la pneumònia i donar millor resultats. En qualsevol cas, és important que quan estiguem desenvolupant un model de xarxa neuronal convolucional, anem provant diferents capes convolucionals, funcions d'activació, augmentar la profunditat i l'horitzontalitat, ja que es pot donar el cas que unes arquitectures funcionin millor per alguns tipus d'imatges.

Finalment, l'objectiu del tercer experiment era el de provar una arquitectura coneguda per la comunitat de la intel·ligència artificial com l'VGG16. Els resultats han set els mateixos que els del segon experiment, amb l'única diferència que a l'estar ja preentrenat el model, els entrenaments són més ràpids. Existeixen molts altres models que donen molt bons resultats, i és una bona idea provar-n'hi de ja creats perquè podem agafar bones idees i veure configuracions diferents.

5.5. Treball futur

Abans de tancar aquest treball volíem dedicar un apartat per fer una petita mirada al futur i plantejar possibles camins que es podrien seguir per a continuar amb aquest projecte. Els

experiments i les implementacions de xarxes neuronals convolucionals que hem realitzat al llarg d'aquest treball, és la punta de l'iceberg de tot el ventall de possibilitats que hi ha en aquest sector, siguin eines o tècniques de desenvolupament.

Així doncs, si volguéssim anar més enllà del que hem aconseguit, que insistim que queda molt per fer, un primer pas molt important seria la col·laboració amb personal sanitari expert en la diagnosi de pneumònies, el qual ens podria ajudar d'una forma òptima a localitzar els patrons que caracteritzen aquesta malaltia, a part de la possibilitat de poder obtenir sets de dades. Amb un coneixement més profund del problema que intentem abordar podem afinar els nostres models ajustant-los amb més precisió creant models més efectius i aplicant fine-tuning i transfer learning, els quals són tècniques molt rellevants d'aplicar si volem aconseguir els millors resultats possibles.

El segon pas seria ampliar els nostres coneixements sobre xarxes neuronals, models estadístics, tractament d'imatges, tecnologies, i en general, tot el que envolta el món del "machine learning".

En definitiva, els següents passos a seguir consisteixen a augmentar els nostres coneixements sobre xarxes neuronals convolucionals i el problema al qual ens enfrontem, per a tenir la capacitat de crear models òptims per diagnosticar pneumònies analitzant imatges de radiografies de tòrax.

6. Bibliografia

[1] «danifarre/pneumonia-detection». https://github.com/danifarre/pneumonia-detection (accedit maig 26, 2022).

[2] J. Schmidhuber, «Deep Learning in Neural Networks: An Overview», 2014, Accedit: juny 09, 2022. [En línia]. Disponible a: http://www.idsia.ch/˜juergen/DeepLearning8Oct2014.texCompleteBIBTEXfile.

[3] «Repositorio Digital IPN: Neurona artificial de McCulloch & Pitts». https://www.repositoriodigital.ipn.mx/handle/123456789/8640 (accedit maig 31, 2022).

[4] «Understanding Activation Functions in Neural Networks | by Avinash Sharma V | The Theory Of Everything | Medium». https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (accedit abr. 05, 2021).

[5] «Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun | by Gavril Ognjanovski | Towards Data Science». https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a (accedit abr. 29, 2021).

[6] D. P. Kingma i J. Lei Ba, «ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION».

[7] J. Wu, «Introduction to Convolutional Neural Networks», 2017.

[8] «3D Visualization of a Convolutional Neural Network». https://www.cs.cmu.edu/~aharley/vis/conv/ (accedit juny 09, 2022).

[9] «Convolution». http://www.dspguide.com/ch13/2.htm (accedit maig 31, 2022).

[10] «Example of 2D Convolution». http://www.songho.ca/dsp/convolution/convolution2d_example.html (accedit maig 31, 2022).

[11] D. Scherer, A. Müller, i S. Behnke, «Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition», Accedit: juny 09, 2022. [En línia]. Disponible a: http://www.ais.uni-bonn.de.

[12] «Training a Convolutional Neural Network from scratch | by Victor Zhou | Towards Data Science». https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754 (accedit maig 18, 2022).

[13] «Data Augmentation | How to use Deep Learning when you have Limited Data». https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ (accedit maig 18, 2022).

[14] «Metrics». https://keras.io/api/metrics/ (accedit maig 07, 2022).

[15] «American Thoracic Society PATIENT EDUCATION | INFORMATION SERIES www.thoracic.org What causes pneumonia?», 2016, Accedit: maig 31, 2022. [En línia]. Disponible a: http://www.cdc.gov/pneumonia/index.html.

[16] «RSNA Pneumonia Detection Challenge | Kaggle». https://www.kaggle.com/c/rsna-pneumonia-detection-challenge (accedit maig 08, 2022).

[17] «Chest X-Ray Images (Pneumonia) | Kaggle». https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (accedit maig 08, 2022).

[18] «GitHub - marcotcr/lime: Lime: Explaining the predictions of any machine learning classifier». https://github.com/marcotcr/lime (accedit maig 17, 2022).

[19] «An overview of VGG16 and NiN models | by Khuyen Le | MLearning.ai | Medium». https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484 (accedit maig 08, 2022).

[20] «tf.keras.layers.LeakyReLU | TensorFlow Core v2.8.0». https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU (accedit maig 08, 2022).

[21] «tf.keras.layers.DepthwiseConv2D | TensorFlow Core v2.8.0». https://www.tensorflow.org/api_docs/python/tf/keras/layers/DepthwiseConv2D (accedit maig 08, 2022).

[22] «imagenet2012 | TensorFlow Datasets». https://www.tensorflow.org/datasets/catalog/imagenet2012 (accedit maig 08, 2022).

[23] «Transferencia de aprendizaje y ajuste | TensorFlow Core». https://www.tensorflow.org/tutorials/images/transfer_learning (accedit maig 08, 2022).