Deep Learning en imatges mèdiques

Francisco Javier Ruz Torres
Máster d'Enginyeria Informàtica
Intel·ligència Artificial

Consultor/a : Samir Kanaan Izquierdo
Professor/a responsable de l'assignatura : Carles Ventura Royo

01 Juny del 2016

Títol del treball:	*Deep Learning en imatges mèdiques*
Nom de l'autor:	*Francisco Javier Ruz Torres*
Nom del consultor/a:	*Samir Kanaan Izquierdo*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament (mm/aaaa):	*01/06/2016*
Titulació o programa:	*Máster d'Enginyeria Informàtica*
Àrea del Treball Final:	*Intel·ligència Artificial*
Idioma del treball:	*Català*
Paraules clau	*Deep Learning* *Processament d'imatges* *Aprenentatge*

Resum:

Les tècniques basades en intel·ligència artificial estan cada cop més presents a les organitzacions actuals. Deep Learning, o en la seva traducció aprenentatge profund, és una d'aquestes tècniques que han arribat a les organitzacions amb l'objectiu de quedar-s'hi. La idea de Deep Learning no és una altre que utilitzar tècniques d'intel·ligència artificial ja conegudes, com per exemple les xarxes neuronals, per a simular el comportament del cervell humà i com fa aquest per aprendre.

Gràcies a la maduresa dels sistemes d'informació de les organitzacions, disposem d'incalculables fonts de dades que, tractades de forma adequada, ens poden ajudar a obtenir una major intel·ligència, aportant un valor afegit envers la competència del nostre negoci. Deep Learning pot ser l'encarregat d'això, ja que amb el tractament adequat podem dotar els sistemes d'una intel·ligència no vista fins al dia d'avui.

El present projecte pretén realitzar l'estudi d'aquesta tècnica per adquirir els conceptes teòrics necessaris per finalment aplicar-la sobre imatges mediques reals i poder extraure característiques i anomalies d'aquestes.

Abstract:

Techniques based on artificial intelligence are becoming increasingly present in today's organizations. Deep Learning is one of those techniques that have reached organizations in order to stay. The main idea is none other than to use known techniques, such as neural networks, to simulate the human brain behavior and how does it do to learn.

Thanks to the maturity of information systems in organizations, invaluable data sources are available which, handled properly, can help us gain greater intelligence providing an added value towards the competition of our business. Deep Learning may be responsible for this because with proper treatment we can provide systems with an intelligence not seen before.

This project aims to study this technique to acquire the essential theoretical concepts and to finally apply them on real medical images in order to assess features and anomalies.

Dedicat a la meva dona i la meva filla

Agraïments

Primer de tot m'agradaria donar les gràcies a Samir, consultor del projecte, per als consells donats i per introduir-me en aquest món tant apassionant de la intel·ligència artificial.

També especial agraïments al Consorci Sanitari del Maresme, per a permetre'm continuar amb els meus estudis i millorar els meus coneixements professionals.

Per últim, donar les gràcies a la meva família per la seva paciència i en especial a la meva dona i la meva filla Lucía, per les hores de jocs i de parc que no hem pogut gaudir, però que de bon tros li recompensaré.

Índex

1 - Introducció

1.1 - Estructura del projecte

L'estructura del projecte consta per una part d'una memòria amb la teòrica necessària per al seguiment i la comprensió del projecte, a més a més de l'explicació de la segona part del projecte que es basa en el desenvolupament d'un prototip software on es posa en practica els coneixements adquirits.

La memòria consta d'un total de 9 capítols. El primer capítol o introducció, està format per les motivacions personals i els objectius que es pretenen assolir amb el projecte, així com una descripció del mateix.

El segon capítol pretén donar a conèixer la versant més teòrica dels conceptes aplicats com son la intel·ligència artificial, el Deep Learning i el model de xarxes neuronals.

El tercer capítol del projecte explica tot el context tecnològic per a dur a terme el prototip, pretén donar a conèixer l'entorn de treball, els diferents FrameWorks valorats, el DataSet que s'utilitzarà i tota la informació necessària sobre les dades a tractar.

El quart capítol s'endinsa a la part més practica del projecte, on s'explica el cas d'estudi que volem analitzar i el seu desenvolupament, així com les diferents proves realitzades per dur a terme la investigació.

Al cinquè capítol s'analitzaran els resultats obtinguts del prototip realitzat.

El sisè capítol pretén resumir l'esforç realitzat i els recursos necessaris per la realització del projecte.

Al setè capítol s'expliquen les conclusions finals del projecte, les dificultats que s'han trobat i el resum dels objectius assolits.

Al vuitè i penúltim capítol es detallen els annexos adjunts a la memòria del projecte.

Finalment al darrer capítol es fa una recopilació de les diferents fonts d'informació i bibliografia utilitzada per l'execució del projecte.

1.2 - Motivacions del projecte i Objectius

El món de la Intel·ligència Artificial ha sigut un àrea que sempre m'ha aixecat bastant admiració, i més encara un cop cursades les diferents assignatures del Màster d'Enginyeria Informàtica relacionades amb aquesta temàtica.

Degut a la meva vinculació des de fa més de 13 anys als sistemes d'informació en entorns sanitaris, volia aprofitar la meva experiència en les TIC Salut i el meu neguit per la Intel·ligència Artificial per a dur a terme un projecte que vincules aquestes dos branques tecnològiques, que personalment tinc la sensació que estan ben poc aprofitades, i poden arribar a aportar beneficis importants als sistemes d'informació sanitaris actuals.

En l'actualitat les organitzacions sanitàries, tecnològicament parlant, es centren molt en la digitalització d'informació, l'intercanvi d'informació entre centres, en la creació de la historia clínica electrònica, en donar suport digital a tràmits de pacients, etc... Això fa que es generi un volum d'informació important, que tractant-se de la forma adequada, es pot aprofitar per donar més intel·ligència al negoci. Les organitzacions disposen de molta informació però molts cops no saben que fer amb ella.

Gracies al context tecnològic actual es pot dir que les organitzacions, inclús les sanitàries, tenen un grau maduresa d'informatització prou elevat, amb la qual cosa és molt important saber que podem arribar a fer amb la informació de que disposen. Les tasques actuals per donar coneixement a les organitzacions es basen en la creació de quadres de comandaments per la direcció, o la creació d'informació per saber quina és la situació actual d'un cert servei, per posar uns

exemples, però malauradament no s'aprofita tot el volum d'informació registrada els darrers anys per dotar de molta més intel·ligència al nostre negoci.

La realització d'aquest projecte em farà assolir, principalment, tres objectius. El primer dels objectius és aprofundir més en una àrea de la intel·ligència artificial com és el Deep Learning i les xarxes neuronals. El segon dels objectius és la creació d'un prototip d'aplicació de Deep Learning sobre dades reals, dades que es poden arribar a registrar en qualsevol sistema d'informació sanitari. I per últim, el darrer objectiu, és demostrar que es pot obtenir molt més profit de tota la informació disponible els sistemes d'informació sanitaris actuals, informació que si la tractem de forma correcta podrien donar molta més intel·ligència al negoci.

1.3 - Descripció del projecte

Darrerament les tècniques basades en intel·ligència artificial estan cada cop més present als sistemes d'informació de les organitzacions actuals. Aquestes tècniques aplicades de forma adequada sobre un conjunt de dades concret, poden dotar d'una intel·ligència i/o saviesa a les corporacions, millorant els seus avantatges competitius respecte a d'altres empreses.

Més concretament, i centrant-nos en una d'aquestes tècniques que avui dia esta de moda, Deep Learning o en la seva traducció aprenentatge profund, esta agafant força en grans organitzacions com poden ser Google, Facebook o Apple. Deep Learning consisteix en la conjunció de diferents tècniques ja conegudes d'intel·ligència artificial, per exemple les xarxes neuronals, amb l'objectiu de simular el comportament del cervell humà i com fa aquest per aprendre. Deep Learning ens permet entrenar sistemes amb les configuracions que necessitem per a ser capaços de classificar-nos i/o identificar objectes d'informació.

Son múltiples els estudis tant d'anàlisi d'imatges, veu o text en sectors com internet, finances, telecomunicacions, transports o de diagnòstic mèdic basats en Deep Learning, on s'observa una millora en el percentatge d'encert envers a d'altres sistemes de classificació i/o predicció. És per aquest motiu que el present projecte va enfocat en l'estudi d'aquesta tècnica.

Degut a la meva experiència al sector IT Salut, he decidit enfocar el projecte en la utilització d'aquesta tècnica per l'anàlisi d'imatges mèdiques, més concretament en la detecció de característiques i/o anomalies de mamografies.

El projecte esta composat d'una primera part encarregada de dotar al lector de tots els coneixements teòrics per la comprensió de la tècnica Deep Learning i la seva implantació basant-se en xarxes neuronals.

Amb la comprensió de la primera part del projecte, ens centrarem en el desenvolupament d'un prototip per l'aplicació de diferents mètodes sobre imatges reals. Les primeres proves es realitzaran sobre un DataSet anomenat MNIST, que consisteix en la identificació i classificació de dígits manuscrits. Seguidament es desenvolupa un prototip per l'anàlisi del DataSet MIAS, corresponent a un conjunt d'imatges de mamografies prèviament classificades.

Amb el prototip desenvolupant, es detallarà l'estudi de diferents configuracions realitzades a les xarxes neuronals definides per a analitzar els resultats obtinguts, i veure la viabilitat de l'aplicació de tècniques Deep Learning sobre imatges mèdiques.

La finalitat d'aquest projecte és obtenir tot el coneixement necessari tant teòric, com tecnològic, com d'arquitectura, per l'aplicació de tècniques Deep Learning en entorn Python, amb un FrameWork en concret i analitzar els resultats sobre la seva aplicació en imatges reals.

2 - Informació general

2.1 - Deep Learning

Deep Learning o en la seva traducció aprenentatge profund, esta de moda aquets darrers anys, i tal com es pot observar al context tecnològic que es viu en l'actualitat no és fruit de la casualitat. Les grans empreses del sector tecnològic, com poden ser Google, Apple, Facebook o Microsoft, cada cop inverteixen més en l'aplicació d'algoritmes per al reconeixement d'imatges, veu, textos, etc...

Si reflexionem durant uns segons i pensem per exemple com funciona Siri de Apple, capaç de reconèixer la veu i mantenir una conversa, o com a Facebook apareixen anuncis de temes que has escrit al teu mur, o com Google anuncia temes de cerques que has fet al seu cercador..., es pot dir que tot això es possible gràcies al Deep Learning, i els grans avanços de la intel·ligència artificial.

La concepció inicial de tots aquets avenços es deuen al Machine Learning, o aprenentatge automàtic, que és una branca de la intel·ligència artificial on l'objectiu principal és que un sistema o maquina sigui capaç d'aprendre i analitzar informació sense la intervenció humana a base de processos d'entrenament o d'exemples, amb l'objectiu de poder predir exemples futurs. Deep Learning consisteix en un conjunt d'algoritmes i tècniques que faciliten l'aprenentatge automàtic.

La idea del Deep Learning no és una altre que imitar el cervell humà i com fa aquest per aprendre. Deep Learning son un conjunt d'algoritmes i tècniques d'intel·ligència artificial que permeten a les maquines o sistemes aprendre per si soles, capaces de reconèixer imatges, veus, textos, etc..

Donat això, els algoritmes d'aprenentatge persegueixen l'objectiu de simular al propi aprenentatge que fa l'ésser humà des de que és un nen. L'aprenentatge

per reforç engloba un grup de tècniques d'aprenentatge automàtic que sovint s'utilitzen en sistemes artificials. En aquets sistemes, al igual als nens, les situacions que son positives tendeixen a augmentar la probabilitat d'aparèixer, en canvi les situacions negatives es castiguen i la probabilitat d'aparèixer disminueixen.

Aquest enfocament és el que s'anomena aprenentatge supervisat, pues requereix de la intervenció humana per indicar el que és positiu i el que no, proporciona el reforç. Partint d'un exemple real, si volem dissenyar un software que classifiqui una sèrie d'articles de premsa, el que primer s'ha de fer és un conjunt significatiu de documents ja categoritzats en forma d'entrenament per a que posteriorment el sistema pugui aprendre. Un cop el sistema estigui entrenat, podrà dur a terme la classificació de nous articles de premsa de forma automàtica sense la intervenció humana.

Però el futur de l'aprenentatge automàtic no està en l'aprenentatge supervisat, sinó en l'altre branca com és la de l'aprenentatge no supervisat, on els sistemes puguin arribar a aprendre o interpretar dades sense intervenció humana. Deep Learning és molt important degut a les seves capacitats d'aproximar-se a la potencia de la perspectiva humana.

Es pot classificar el tipus d'aprenentatge en:

- **Supervisat**: necessita un conjunt de dades d'entrada prèviament classificat.
- **No supervisat**: no necessita entrenament previ, son capaces d'aprendre soles mitjançant associacions de les dades.
- **Per reforç**: aquest tipus d'aprenentatge s'ubica entremig dels dos anteriors.

De forma molt resumida, el Deep Learning funciona com un procés de capes que simulen el funcionament bàsic de cervell que es realitza a traves de neurones, és a dir, per capes de neurones. Això evidencia que un dels models per la implementació de Deep Learning es basa en xarxes neuronals, que s'explica amb mes profunditat al capítol 2.3 d'aquesta memòria i en la qual es centra aquest projecte.

Existeixen d'altres models per a la utilització de tècniques capaces d'aprendre per si mateix a traves d'aprenentatge profund com poden ser: arbres de decisió, algoritmes genètics, xarxes bayesianes, etc...

Deep Learning no està basat en tècniques noves, sinó que utilitza tècniques que fa molt de temps que existeixen, algunes de les quals son molt complexes com per exemple l'entrenament de xarxes neuronals. El gran avanç no ha estat un altre que poder combinar totes aquestes tècniques per aconseguir que tant l'aprenentatge supervisat com el no supervisat, sigui eficient.

Com s'ha fet referència, l'aprenentatge profund no és una idea innovadora. L'investigador japonès Kunihiko Fukushima al voltant dels anys vuitanta, va proposar un model neuronal de cinc i sis capes al que va denominar Neocognitron, però degut a la dificultat per al seu desenvolupament i/o a la falta de finançament va fer que aquesta tècnica quedes en un segon lloc fins fa aproximadament una dècada on s'ha reactivat l'interès i la inversió per part de les grans empreses.

2.2 - Deep Learning: I ara, perquè?

La transformació digital al sistema empresarial està prou instaurada, i el terme Big Data ha revolucionat les organitzacions. Les organitzacions s'han convertit en un pou interminable de dades, i gràcies a això existeix una demanda generalitzada de sistemes amb una intel·ligència avançada que siguin capaces de processar-les. Això esta passant a pràcticament tots els sectors ja que poden obtenir grans avantatges competitius amb l'anàlisi intel·ligent i automàtic de les dades.

En l'actualitat és pot dir que s'alineen tots els factors necessaris per dur a terme aquesta revolució tecnològica: tecnològica necessària, grans volums de dades i informació, i inversió empresarial.

Es pot afirmar que el Big Data és el provocador d'aquesta revolució tecnològica. En l'actualitat estem veient que el Big Data prolifera de forma rapida a totes les organitzacions i sectors, disposant d'infinitat de dades capaces de ser analitzades. Gràcies a la tecnologia actual es poden analitzar aquestes dades, reduint el temps del que suposaria el mateix anàlisis equivalent a l'esser humà, inclús dotant-lo de la intel·ligència necessària que fins ara els sistemes no suportaven.

Les grans organitzacions son conscient del moment que vivim, i veuen en l'aprenentatge automàtic un impuls competitiu en molts sectors. Per tal de donar intel·ligència a les seves dades, no es poden permetre el luxe de programar totes les infinites regles i situacions que apareixen al món real. Es necessiten sistemes que siguin capaces d'auto-programar-se, és a dir que aprenguin sols. El Machine Learning s'ocupa d'això.

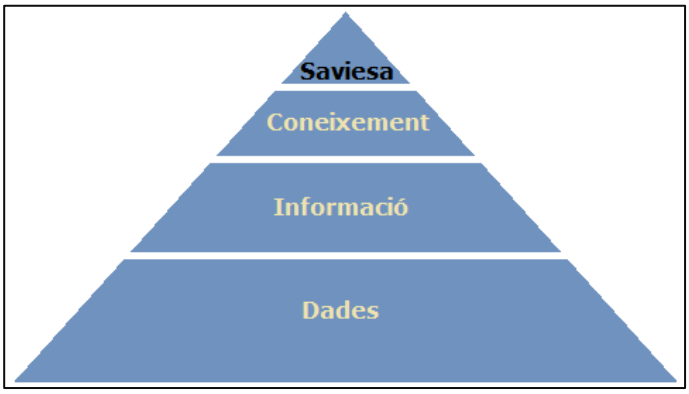


Figura 1 – Piràmide coneixement empresarial

Com es pot observar a la Figura 1, equivalent a la piràmide del coneixement empresarial, en l'actualitat les organitzacions estan gaudint d'un alt coneixement de la informació que provoquen les seves dades, però estem a un pas d'aprofitar aquest coneixement per obtenir saviesa, i això o pot aconseguir el Deep Learning.

2.3 - Xarxes neuronals

2.3.1 - Introducció

Encara que existeixen diferents models d'implementar Deep Learning, tal i com s'ha comentat al capítol anterior, una de les més comuns i en la que es basa el projecte, és utilitzant xarxes neuronals.

Una xarxa neuronal és una tècnica matemàtica que modela de forma molt simplificada el funcionament de les neurones del cervell. Es tracta d'un sistema d'interconnexions de neurones que col·laboren entre si per a produir un estímul de sortida.

De forma molt esquemàtica es podria veure com un processador de dades que rep una informació d'entrada, la qual és codificada amb números, i un cop realitzats una sèrie d'operacions s'obté un resultat final també codificada numèricament.

La següent imatge mostra simplificadament l'explicació anterior:

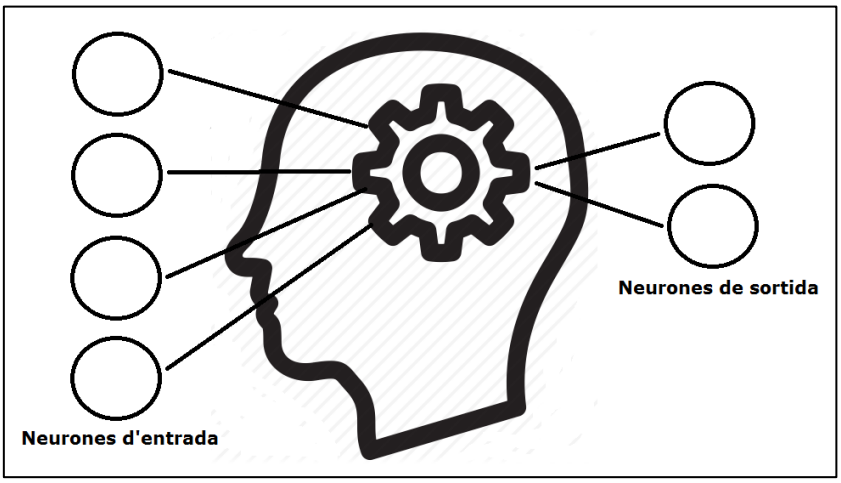


Figura 2 – Esquema bàsic Entrades i Sortides

2.3.2 - Arquitectura

Per contra del que pot semblar, el concepte que hi ha darrere de les xarxes neuronals no és gaire complexa. Un conjunt de neurones connectades entre si, sense que cada neurona tingui una tasca concreta, que assolint una experiència (entrenament), les neurones van creant i reforçant certes connexions per aprendre alguna cosa.

És a dir, donats uns paràmetres hi ha forma de combinar-los per a predir uns resultats. Per exemple, coneixent els píxels d'una imatge, existeix una forma de saber quin numero hi ha escrit en aquesta imatge.

Les xarxes neuronals son un model per trobar aquesta combinació de paràmetres per obtenir un resultat. Al present projecte es centra en l'entrenament d'una xarxa neuronal, ja que una xarxa neuronal entrenada es pot utilitzar per a obtenir prediccions o classificacions d'altres objectes nous.

A la següent figura podem observar l'arquitectura d'una xarxa de neurones:

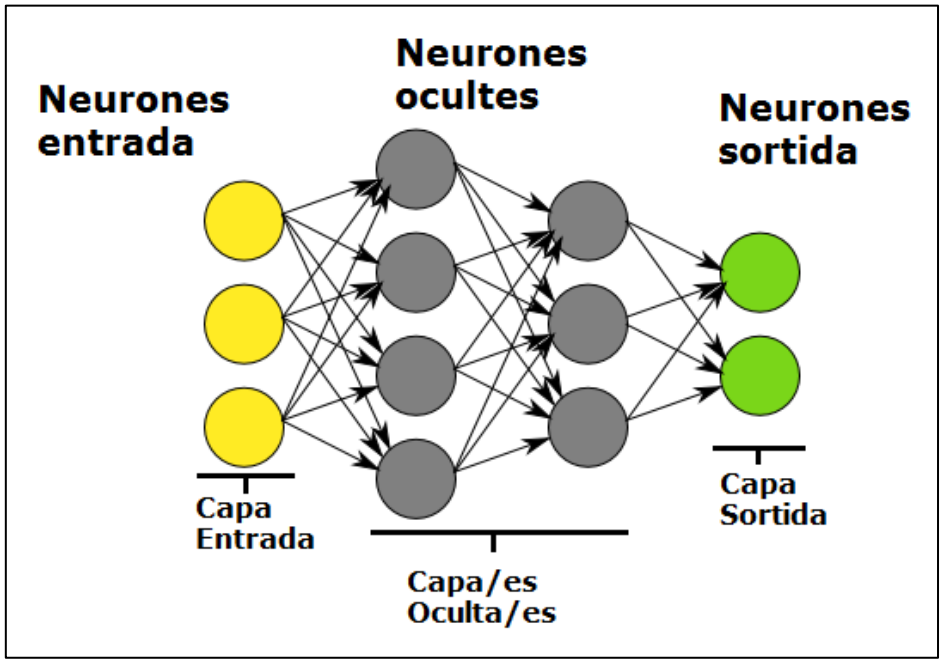


Figura 3 – Arquitectura bàsica Xarxa de neurones

Cada cercle representa l'equivalent a una neurona i aquestes neurones estan organitzades per capes, de tal forma que les neurones grogues son les neurones d'entrada i cada una rep el valor d'un numero del llistat de números entrants. Agafant un exemple, si l'objecte d'entrada sigues una imatge, cada neurona d'entrada equivaldria a cada píxel de la imatge, o tres en el cas que sigues una imatge en color.

Les neurones de color verd equivaldrien a les sortides, és a dir, el resultat final després de realitzar les diferents operacions matemàtiques, que també seria un llistat de números. Seguint amb l'exemple, imaginem que les imatges d'entrada son imatges de gats i gossos, on volem que el sistema detecti si a la imatge hi ha un gat o un gos. La neurona de sortida podria ser única, indicant amb un 1 si és un gat i un 2 si és un gos.

Les neurones de color gris son les neurones hidden (ocultes), que contenen els càlculs interns de la xarxa. És en aquets càlculs on amb la informació de les neurones d'entrada, s'aconsegueix un resultat representat en les neurones de sortida.

Totes les neurones de cada capa tenen una connexió amb cada neurona de la capa següent (normalment, depèn de l'arquitectura final). Aquetes relacions tenen associades un numero, anomenat pes. La principal operació que realitza la xarxa de neurones consisteix en multiplicar els valors de una neurona per els pesos de les seves connexions de sortida, i cada neurona de la capa següent rep el numero de varies connexions entrants, i el primer que fa és sumar-los.

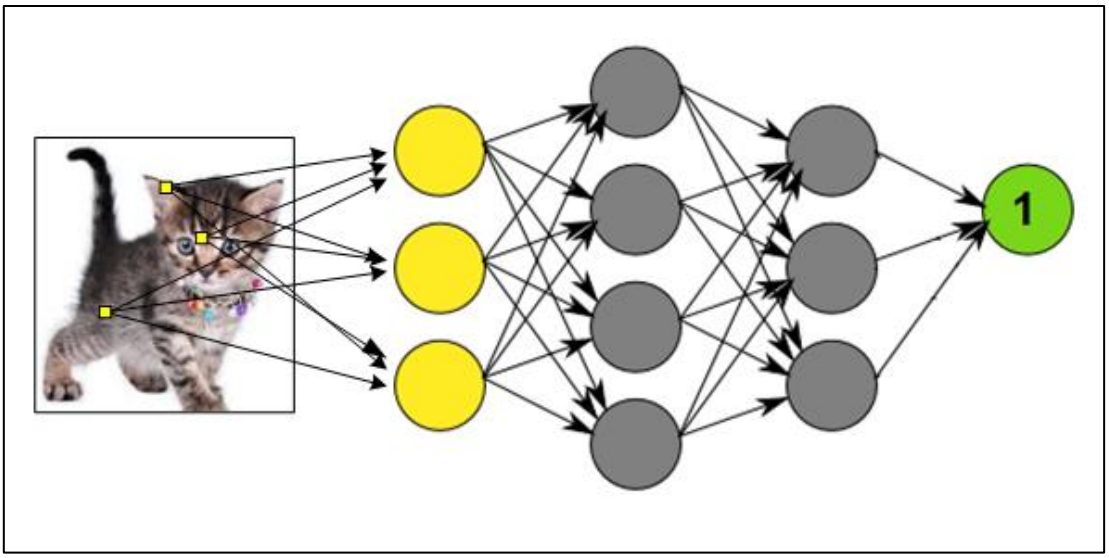


Figura 4 – Exemple Xarxa Neuronal

Com es pot observar, la figura 4 representa l'arquitectura de l'exemple comentat anteriorment, on l'objectiu és detectar si en una imatge hi ha un gat o un gos. La figura representa la xarxa neuronal amb 3 píxels, amb mode d'exemple, encara que en la realitat la xarxa neuronal serà molt més extensa degut a tenir que representar tots els píxels de la imatge com a neurones d'entrada.

Com ja s'ha explicat anteriorment, les neurones d'una capa tenen connexió amb les neurones de la capa següent. Cada neurona rep una sèrie d'entrades i emet una sortida. Aquesta sortida ve donada per tres funcions:

1- Funció de propagació, que de forma general consisteix en el sumatori de cada entrada multiplicat per al pes de la seva interconnexió. Si el seu pes és positiu, la connexió s'anomena excitadora, si és negatiu s'anomena inhibitòria.

2- Funció d'activació, que modifica l'anterior. Pot no existir, llavors la sortida serà la mateixa funció de propagació.

3- Funció de transferència, s'aplica al valor retornat per la funció d'activació. S'utilitza normalment per acotar la sortida de la neurona i generalment ve donada per la interpretació que es vol donar a la sortida. Exemples poden ser la funció sigmoide per obtenir valors entre 0 i 1 o la tangent hiperbòlica per obtenir resultats entre -1 i 1.

La funció sigmoide és una funció no lineal, això significa que si es dibuixés en una gràfica els valors d'entrada en un eix i la sortida en un altre eix, el dibuix no seria una línia. Això és molt important ja que si la funció d'activació i transferència és lineal, la xarxa neuronal tant sols estarà limitada a resoldre problemes lineals, és a dir, problemes simples.

Un altre concepte important que s'ha de tenir en compte en l'arquitectura de les xarxes neuronals son les Bias. Just abans d'aplicar la funció d'activació cada neurona afegeix la suma de productes a un nou terme constant, anomenat Bias. Una forma típica d'implementar la Bias consisteix en imaginar-se que s'estén la capa anterior amb una falsa neurona que sempre agafa com a valor un 1, i incorpora els pesos corresponents a aquesta falsa neurona de la matriu de pesos. Per veure de forma molt senzilla la seva utilitat imaginem una funció d'activació lineal, i una xarxa de tant sols 2 neurones, una d'entrada i una de sortida. En aquesta xarxa volem obtenir la conversió de graus Celsius i Fahrenheit. Llavors tenim un problema important ja que 0 graus Celsius correspon a 32 graus Fahrenheit i tant sols amb una multiplicació no es pot obtenir el valor esperat. Llavors, introduint el terme de Bias, tenim una multiplicació i després una suma, lo qual simplifica poder realitzar aquesta conversió.

2.3.3 - Altres arquitectures

És important conèixer que existeixen d'altres arquitectures de xarxa de neurones diferents per la implementació de tècniques Deep Learning. El tipus

d'arquitectura que s'utilitzarà pot venir donat pel problema que es vol resoldre. Veiem les següents arquitectures:

- **Xarxes de neurones recurrents:** aquest tipus d'arquitectura no té una estructura per capes, sinó que permeten connexions de forma arbitraria entre totes les neurones. Això permet incorporar a la xarxa el concepte de temporalitat i permet que la xarxa tingui memòria, ja que els paràmetres d'entrada que s'introdueixen en un moment donat en les neurones d'entrada, son transformats i continuen circulant per la xarxa, inclús després de canviar els valors d'entrada per d'altres diferents.

- **Xarxes de neurones convolutives:** aquest tipus d'arquitectura manté el concepte de capes, però cada neurona de una capa no rep connexions entrants de totes les neurones de la capa anterior, sinó tant sols d'algunes. Això fa que una neurona s'especialitzi en una regió de la llista de neurones de la capa anterior i redueix notablement el nombre de pesos i de càlculs. El més habitual és que dos neurones consecutives d'una capa mitjana s'especialitzi en regions solapades de la capa anterior.

2.3.4 – Entrenament i aprenentatge supervisat

Continuant amb l'exemple de la detecció de si en una imatge hi ha un gat o hi ha un gos, suposem que ja tenim ben identificades quantes neurones fan falta per l'entrada i quantes neurones fan falta per la sortida, ja que hem decidit com representar la informació en forma de llista de números.

Suposem que a la xarxa li entraran imatges de 150x150 píxels, llavors tenim que hi hauran un total de 22500 neurones d'entrada (una per cada píxel, imaginarem que son imatges en blanc i negre). I que hi haurà tant sols una neurona de sortida, on aquesta neurona indicarà 1 si és un gat i 2 si és un gos.

Encara resten varies coses per decidir per tal de poder implementar la xarxa neuronal:

- Quantes capes ocultes utilitzarem?
- Quantes neurones posarem a cada capa oculta?
- Quins pesos concrets utilitzarem en les connexions per a cada capa?

Normalment els dos primers punts es decideixen de forma subjectiva i mitjançant prova i error. És lògic que si el nombre de neurones és alt en les capes ocultes, la xarxa és més complexa i podria arribar a resoldre problemes més complexos, per contra també serà més costos la realització de tots els seus càlculs.

El tercer punt es resolt de forma automàtica, mitjançant un procés d'entrenament. Per entrenar aquesta xarxa de neurones necessitem recopilar exemples d'entrada i la sortida que desitgem que tingui cada exemple. Continuant amb l'exemple, es necessiten exemples d'imatges de gats que les etiquetarem amb un 1 i es necessiten exemples d' imatges de gossos que les etiquetarem amb un 2. Aquest procés d'entrenament és el que es coneix com a aprenentatge supervisat, ja que el sistema necessita d'un supervisor que li expliqui el que ha d'aprendre.

L'aprenentatge pot ser vist com el procés d'ajustament dels paràmetres de la xarxa neuronal. Partint d'una sèrie de pesos aleatoris, el procés d'aprenentatge intenta cercar un conjunt de pesos que permetin a la xarxa desenvolupar una determinada tasca. Aquest procés d'aprenentatge és un procés iteratiu, el qual va refinant la solució fins aconseguir un nivell d'operació suficientment alt.

La majoria dels mètodes d'entrenament utilitzats en les xarxes neuronals consisteixen en proposar una funció d'error que mesuri el rendiment actual de la xarxa en funció dels seus pesos. L'objectiu del mètode d'entrenament és trobar un conjunt de pesos que minimitzin o maximitzin la funció d'error. El mètode d'optimització proporciona una regla d'actualització dels pesos que en funció d'uns patrons d'entrada modifica de forma iterativa els pesos fins aconseguir un nivell òptim de la xarxa neuronal.

Uns dels mètodes més utilitzats per a l'entrenament supervisat de les xarxes i la reducció d'errors és el del propagació enrere (backpropagation). Un cop tenim un conjunt d'exemples, és fàcil avaluar la xarxa amb cada exemple i comprovar l'error entre la sortida desitjada i la sortida que esta produint la xarxa. Per a calcular l'error, en cada neurona de la capa de sortida, restarem el valor produït i el que esperàvem. Sabent quin és l'error per un determinat exemple, és pot corregir.

L'idea de corregir errors consisteix en buscar culpables que produeixen aquets errors. Entre les neurones de sortida és fàcil saber quines son les culpables, però cadascuna d'aquestes neurones esta connectada amb les neurones anteriors mitjançant pesos. Es poden utilitzar aquets pesos per a determinar quant contribueixen les neurones anteriors a l'error simplement propagant els errors cap enrere, del mateix mode que es propaga cap endavant, multiplicant i sumant. D'aquesta manera es pot saber quant contribueix a l'error cada neurona de la xarxa.

Sabent quant contribueix cada neurona a l'error, es pot intentar actualitzar pesos per a reduir aquest error. Primer es necessita saber com afecta a l'error els canvis produïts als pesos. Identificat això, el que cal és canviar els pesos de forma justa per a que l'error es redueixi a la major velocitat possible. Tot aquest procés s'inicia amb pesos generats a l'atzar, que es van auto-corregint amb l'algoritme corresponent.

L'algoritme de propagació enrere té un greu problema, i és que l'error es va diluint de forma exponencial a mida que es van passant capes fins al principi de la xarxa. En xarxes molt profundes, amb moltes capes ocultes, resulta un problema ja que tant sols les últimes capes son les que s'entrenen, en canvi les primeres no sofreixen canvis. Per això en moltes ocasions compensa utilitzar xarxes amb poques capes que continguin moltes neurones, envers a xarxes amb moltes capes que continguin menys neurones en cada capa.

2.3.5 - Aprenentatge no supervisat: Auto-codificadors

Un auto-codificadors és normalment una xarxa de neurones de tres capes que s'utilitza habitualment per implementar Deep Learning. Un auto-codificador el que fa és reproduir a la sortida exactament la mateixa informació que la que ha rebut d'entrada, amb això es pot deduir que la capa d'entrada i de sortida tenen el mateix nombre de neurones.

A simple vista pot semblar que això no pugui servir per res, però el secret està en la tercera capa, la capa oculta. Imaginem un auto-codificador que com a entrada té 100 neurones i com a conseqüència la sortida també té 100 neurones. Ara definim la capa oculta amb tant sons 50 neurones. Podem arribar a tenir una entrada de 100 neurones, representades en la capa oculta amb 50 neurones i capaç de tornar a reproduir-se amb 100 neurones de sortida. Donat que exigim a la xarxa que puguem representar la mateixa informació de l'entrada a la sortida, i aquesta informació ha de passar per una capa oculta de menys neurones, la xarxa es veu obligada a trobar una representació de la informació en la capa oculta utilitzant menys neurones. Llavors es pot deduir que la capa oculta tindrà una versió comprimida de la informació, però a més serà una versió comprimida que pot tornar a descomprimir-se.

És per això que aquest tipus de xarxes neuronals s'anomenen auto-codificadors, ja que son capaces de descobrir per si mateixes una forma alternativa de codificar la informació i no necessiten cap tipus de supervisió (no necessiten entrenament supervisat).

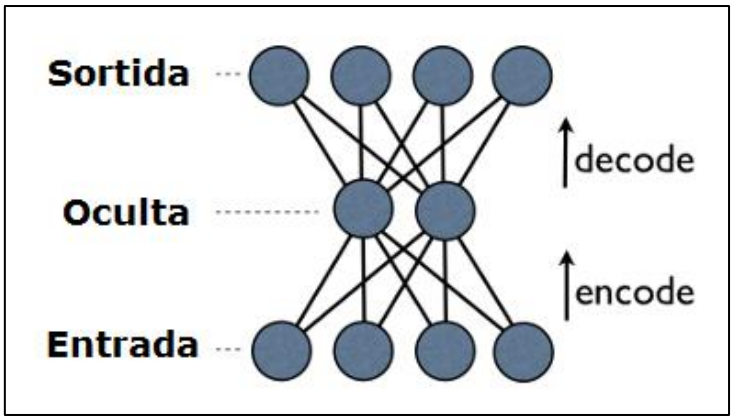


Figura 5 – Auto-codificador

Aplicant un sol auto-codificador es poden trobar característiques bàsiques de la informació d'entrada, les característiques més primitives i simples que es puguin extraure d'aquesta informació. Posant un exemple, amb un auto-codificador aplicat a una imatge, es pot extraure informació com a rectes, corbes, punts, etc.. però si l'objectiu és definir una xarxa que detecti característiques més complexes, com pot ser detectar si en una imatge hi ha un gat o un gos, ens fa falta més potència.

Per aconseguir aquesta potencia s'utilitza el que s'anomena auto-codificadors apilats. Partint de la informació d'entrada, per exemple píxels d'imatges, amb auto-codificadors som capaços de cercar característiques com poden ser formes simples (línies, corbes, punts...). Imaginem que al resultat codificat en aquesta capa oculta li apliquem un altre auto-codificador, pot arribar a identificar característiques més complexes. Si continuem aplicant auto-codificadors varies vegades, obtindrem una jerarquia de característiques cada cop més complexes, conjuntament amb una pila de auto-codificadors creats. Llavors si tenim el nombre d'imatges necessàries i la profunditat suficient, podem aconseguir que alguna neurona s'activi quan en una imatge hi ha un gat o un gos, sense la necessitat de que ningú li expliqui a la xarxa neuronal que és un gat o un gos.

La idea de Deep Learning amb xarxes neuronals és precisament aquesta. Utilitzar varis codificadors, entrenar-los un a un, utilitzant cada codificador entrenat per a entrenar al següent, això rep el nom d'algoritme voraç o greedy, i aquest és realment el gran avanç del Deep Learning avui dia.

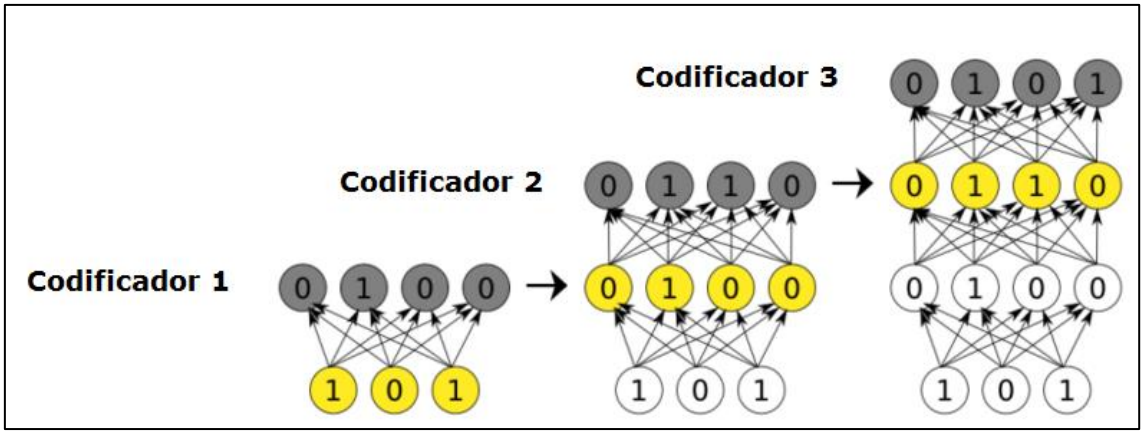


Figura 6 – Auto-codificador apilats

Una xarxa creada amb auto-codificadors apilats té les següents característiques, les quals són molt importants:

- És capaç d'aprendre sense supervisió, tant sols necessita dades i troba entre elles les característiques freqüents i les etiqueta.
- Es poden entrenar xarxes tot lo profundes que vulguem. Tal i com s'ha comentant en l'algoritme de propagació enrere, al entrenar xarxes molt profundes l'error es va diluint i les primeres capes casi no s'entrenen. Amb els auto-codificadors no hi ha aquets problemes.

Els auto-codificadors no son els únics mecanismes per a realitzar Deep Learning, existeixen d'altres com pot ser les xarxes de creença profunda (Deep Belief Networks - DBN). El seu funcionament és molt semblant, però en comptes d'utilitzar auto-codificadors, utilitza una Restricted Boltzmann Machine (RBM). Les xarxes de Boltzmann consisteixen en neurones connectades entre si que poden estar interconnectades de forma bidireccional.

3 - Context tecnològic

3.1 - Entorn de treball

Una de les primeres decisions que s'han pres per a l'execució del desenvolupament del prototip, és l'elecció de l'entorn de treball amb el que es treballarà. Existeixen diferents entorns que suporten l'execució de llibreries basades en Deep Learning, com per exemple:

- Java
- CPP
- Python
- .NET

Després de valorar les diferents alternatives, l'elecció de l'entorn ha estat Python, ja que és un entorn conegut a les assignatures d'Intel·ligència Artificial i disposa de multitud de llibreries per facilitar l'anàlisi de dades i la creació de xarxes neuronals per la implementació de Deep Learning.

El desenvolupament es realitza amb un ordinador portàtil amb sistema operatiu Windows 7 de 64 bits i 8GB de RAM, amb una targeta gràfica RADEON Graphics de memòria compartida.

Amb les característiques hardware disponibles es decideix utilitzar l'entorn de desenvolupament WinPython que conté Spyder.

Es treballa amb 2 entorns de desenvolupament Python, un per la versió 2 i un altre per la versió 3, per tal de simplificar els diferents desenvolupament i la incompatibilitat de llibreries que existeixen entre les diferents proves realitzades.

Les versions utilitzades són:

- WinPython 64bits 2.7.9.5
- WinPython 64bits 3.5.1.3

3.2 - FrameWorks analitzats

Partint de l'elecció de Python com l'entorn de treball pel desenvolupament del prototip del projecte, s'analitzen les diferents alternatives disponibles de FrameWorks i/o llibreries per la implementació de Deep Learning. Recordar tal i com s'ha comentat en capítols anteriors, que Deep Learning s'aprofita de tècniques ja conegudes des de fa temps, que fent ús de la seva unió pot arribar a resoldre problemes que fins fa uns anys eren impensables.

Avui dia el Machine Learning o el Deep Learning més concretament, esta a l'abast de tothom ja que disposem de les eines necessàries per la seva implantació d'una forma més o menys senzilla, que garanteix uns resultats més que òptims.

L'entorn i la comunitat Python tenen disponible un ventall de llibreries disponibles per la implantació de Deep Learning, les quals no s'analitzaran totes ja que no és l'objectiu del projecte. L'objectiu de l'anàlisi de diferents FrameWorks, és ser capaços d'identificar els pros i contres d'utilitzar un FrameWork o un altre, per a resoldre un problema en concret, i conèixer fins a on aquets FrameWorks poden arribar.

A més a més del FrameWork escollit per a l'execució de tècniques Deep Learning, s'utilitzaran d'altres llibreries que ens facilitarà el desenvolupament del prototip, però aquestes les veurem amb més detall al capítol 4 Implantació del prototip.

Les llibreries analitzades son:

- **Theano** : Segurament una de les llibreries més utilitzades per al desenvolupament de tasques matemàtiques i de Deep Learning ja que molts dels FrameWorks que existeixen es basen en aquest. És una llibreria de més baix nivell, això implica que dona molta versatilitat per resoldre qualsevol tipus de problema, però també la complexitat del seu desenvolupament és superior.

Desenvolupada per investigadors de la universitat de Montreal en Canada. És una llibreria molt potent que permet definir, optimitzar i avaluar expressions en les que es treballin amb matrius multidimensionals, sen molt útil per al desenvolupament i disseny de xarxes neuronals i obtenir una major eficiència als càlculs. És molt important saber que Theano permet un ús totalment transparent en la utilització de processadors GPU, amb la qual cosa millorarà considerablement els temps de càlculs dels processos desenvolupats.

- **Pylearn2** : Entrenar una xarxa neuronal basada en Theano requereix un alt nivell de complexitat, ja que s'ha de definir el comportament complert de la xarxa neuronal creada. Per això existeixen altres llibreries que ens permet obtenir un nivell més alt d'abstracció i simplificar els processo d'experimentació amb xarxes neuronals, i aquest és l'objectiu de Pylearn2. Pylearn2 esta basada en Machine Learning i usa internament Theano. Permet dividir el problema de Machine Learning en tres parts ben diferenciades, el DataSet, el model i els algoritmes d'entrenament.

- **Caffe:** FrameWork per al desenvolupament de Deep Learning basat en expressions, amb les característiques de que és ràpid i modular. Està desenvolupat per la Berkeley Vision and Learning Center (BVLC). Es basat en arquitectura expressiva que fomenta l'aplicació i la innovació, modelant models optimitzats. Suport tant de CPU com GPU, amb la capacitat de canviar el tipus d'execució de forma molt simple. És de codi extensible i això fomenta el desenvolupament actiu. Caffe destaca per ser un FrameWork molt ràpid per això és perfecte per a projectes d'investigació. És capaç de processar 60 milions d'imatges en un sol dia amb una sola GPU NVIDA K40. Els seus desenvolupadors comenten que és la xarxa de convolució (CNN) més ràpida fins ara disponible. Ara mateix s'està utilitzant per a projectes d'investigació i fins i tot industrials per temes de visió, parla o multimèdia.

- **Tensorflow:** FrameWork desenvolupat per Google per a projectes d'aprenentatge automàtic, especialitzant-se en el reconeixement de veu, textos i imatges. FrameWork encara una mica verd, però que Google ha alliberat el seu codi font per a que investigadors i desenvolupadors juntament amb Google puguin millorar el seu sistema d'intel·ligència artificial. Permet interacció amb la GPU i és d'un alt nivell d'abstracció per a simplificar el desenvolupament dels projectes. Google porta temps potenciant el món del Deep Learning, on el seu primer pas va ser DistBelief. DisBelief és el responsable de la millora del reconeixement de veu, la cerca d'imatges en Google fotos, etc.. El problema és que DistBelief tenia moltes limitacions ja que era difícil de configurar i estava molt lligat a la infraestructura de Google i per això apareix Tensorflow, que resolt tots aquets problemes.

- **Lasagne:** Lasagne és una de les múltiples llibreries basades en Theano per la construcció de xarxes neuronals. Al estar basat en Theano comparteix moltes de les seves característiques però sent una llibreria de més alt nivell, és a dir, el seu desenvolupament és més simple. Permet desenvolupar tan xarxes recurrents com de convolutives i la combinació d'ambdues. Lasagne permet arquitectures de múltiples entrades i múltiples sortides incloent classificadors auxiliars. Consta de molts mètodes d'optimització (Nesterov, RMSprop i ADAM), que millora el rendiment de la xarxa. Una diferenciació envers Theano és que porta la definició d'una funció de cost, sense la necessitat de derivar gradients, i al igual que Theano el seu suport de CPU i GPU és transparent.

  El seu desenvolupament és basa en sis principis:

  - Simplicitat: fàcil d'utilitzar, d'entrenar, d'estendre, etc...
  - Transparència: No cal conèixer la complexitat de Theano, s'utilitza amb tipus de dades de Python / Numpy.
  - Modularitat: Permet modular totes les parts del procés.
  - Pragmatisme: Fer coses d'ús comú, fàcils.

- Restricció: No dificultar a l'usuari amb característiques que no li cal utilitzar.
- Enfoc: Fer una cosa i fer-lo bé.

- **Nolearn:** Nolearn és un FrameWork que té com a objectiu dotar d'un nivell més alt d'abstracció a l'usuari en la utilització de tècniques de Deep Learning. Nolearn utilitza llibreries ja existents per al tractament de xarxes neuronals, sobre tot de Lasagne i Theano. Un punt fort d'aquest FrameWork, a part del seu alt nivell d'abstracció, és la seva alta compatibilitat amb la llibreria scikit-learn, llibreria molt utilitzada per al tractament de dades i el seu anàlisis, així com per Machine Learning. Nolearn incorpora moltes de les avantatges de Lasagne, simplificant encara més el seu ús.

A continuació es mostra un quadre comparatiu de diferents característiques dels FrameWorks que s'han valorat per la realització del projecte:

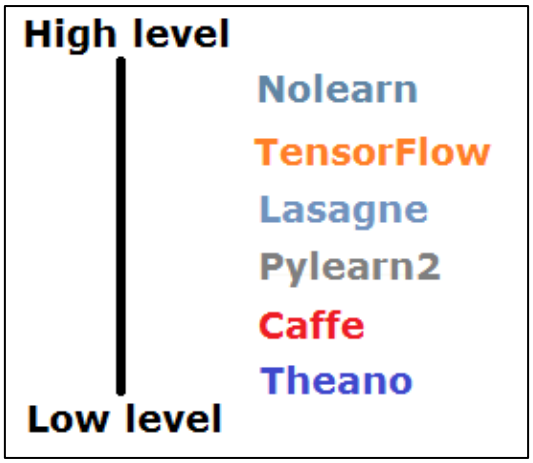
Llibreria	Nivell abstracció	Open Source	GPU	Rec. nets (RNN)	Conv. nets(CNN)	RBM/DBN
Theano	Baix	Sí	Sí	Sí	Sí	Sí
Pylearn2	Mig-Alt	Sí	Sí	Sí	Sí	Sí
Caffe	Mig	Sí	Sí	Sí	Sí	No
Tensorflow	Alt+	Sí	Sí	Sí	Sí	Sí
Lasagne	Alt	Sí	Sí	Sí	Sí	No
Nolearn	Alt+	Sí	Sí	No	Sí	Sí

Les diferents característiques que es valoren de cada FrameWork son:

- **Nivell d'abstracció**: Es valora el nivell d'abstracció per l'usuari en la implementació de xarxes neuronals.
- **Open Source**: Es valora si el FrameWork és de codi obert per a possibles millores de les eines per part de l'usuari o la comunitat que hi ha darrera.

- **GPU**: Es valora si el FrameWork permet la utilització de la GPU per la seva execució, punt molt important per millorar els temps d'execució.
- **RNN**: Es valora si permet la creació de xarxes amb arquitectures recurrents.
- **CNN**: Es valora si permet la creació de xarxes amb arquitectura convolutiva.
- **RBM/DBN**: Es valora si permet la creació de xarxes amb arquitectura Restricted Boltzmann Machines/Deep Belief Networks

A continuació mostrem els diferents FrameWorks analitzats per nivell d'abstracció:



FrameWorks – Nivell d'abstracció

Un cop estudiats els diferents FrameWorks disponibles en l'actualitat per al desenvolupament de Deep Learning en Python, la decisió ha estat la utilització del FrameWork **nolearn**.

Les decisions claus per l'elecció d'aquest FrameWork han estat principalment:

- El seu alt nivell d'abstracció respecte a d'altres FrameWorks analitzats. En aquest punt Tensorflow de Google també era una opció però nolearn és un FrameWork més madur i que el seu grau de desenvolupament, a dia d'avui, és major.

- La seva total integració amb la llibreria Sciki-learn de Python amb múltiples funcions per a resoldre problemes matemàtics i de Machine Learning.
- Per estar basat en Lasagne i Theano, dos dels FrameWorks més comuns en el desenvolupament de Deep Learning en Python.
- Per suportar l'execució en GPU que millorarà notablement els temps de càlculs de la càrrega, l'entrenament i el test de les diferents xarxes neuronals creades.
- Per al suport de la creació tant de xarxes CNN com xarxes DBN, que son les que s'utilitzaran al projecte.

3.3 - Preparació de l'entorn de treball

Decidits tant l'entorn de desenvolupament com el FrameWork a utilitzar, s'ha de preparar i configurar tots els programes necessaris.

Com s'ha comentat, s'utilitzaran tant la versió 2 com la versió 3 de Python, per aquest motiu el primer pas és instal·lar els 2 entorns de desenvolupament.

Els tenim disponibles a:

https://sourceforge.net/projects/winpython/files/

Les versions instal·lades son:

- WinPython 64bits 2.7.9.5 (Python 2)
- WinPython 64bits 3.5.1.3 (Python 3)

Instal·lats els 2 entorns, el següent pas és instal·lar GIT GUI. GIT GUI és un programa per a gestionar i accedir a repositoris de codi i llibreries. S'utilitzarà per a poder descarregar les darreres versions de nolearn.

La seva descarrega es pot fer des de:

https://git-scm.com/download/win

A continuació s'instal·len les llibreries Python necessàries:

- **WinPython 64bits 2.7.9.5 (Python 2):**
    - Des de WinPython Command Prompt, executar la comanda:
        - pip install nolearn

      Instal·larà la versió nolearn disponible al repositori Python. Amb aquesta versió es realitzarà el desenvolupament de nolearn.dbn, que s'explicarà al mòdul 4 de la memòria.

- **WinPython 64bits 3.5.1.3 (Python 3):**
    - Des de WinPython Command Prompt, executar les comandes:
        - pip install -r https:// raw.githubusercontent.com /dnouri/ nolearn/ master/ requirements.txt
        - pip install git+https:// github.com/dnouri/ nolearn.git@master#egg=nolearn==0.7.git

      La primera comanda instal·larà totes les llibreries necessàries per l'execució de nolearn.

      La segona comanda instal·larà amb GIT la versió 0.7 de nolearn disponible al repositori github.com/dnouri.

      Amb aquesta versió es realitzarà el desenvolupament de nolearn.lasagne que s'explicarà al mòdul 4 de la memòria.

També és necessari tenir disponible algun visor d'imatges compatible amb imatges .pgm. Es recomana utilitzar GIMP 2 per ser un editor d'imatges gratuït i bastant complert. Es pot descarregar a l'adreça:

https://www.gimp.org/downloads/

3.4 - DataSet : The mini-MIAS data base of mammograms

Com s'explica en profunditat al capítol 4.1 Cas d'estudi, la implementació del prototip va enfocada en la detecció de anomalies en imatges mèdiques de mamografies. S'han realitzat diverses cerques tant a la xarxa com a la meva organització per obtenir un DataSet que compleixi amb els requeriments necessaris per a la realització del cas d'estudi i poder generar un prototip software.

Finalment s'utilitza el DataSet "The mini-MIAS data base of mammograms" que es pot trobar a la url: http://peipa.essex.ac.uk/info/mias.html

La Mammographic Image Analysis Society (MIAS) és una organització de grups d'investigadors del Regne Unit amb l'objectiu de la comprensió i l'estudi de mamografies. Les imatges estan disponibles via el pilot europeu de processament d'arxivat d'imatges (PEIPA) en la universitat de Essex.

Aquest DataSet consta d'un total de 322 imatges en format .pgm (format en escala de grisos, que s'explica al capítol 3.5) dels quals estan informades en parell de imatges (bilateralitat) on el nombre parell representa la imatge esquerre i el nombre imparell la imatge dreta.

Les imatges tenen una mida de 1024x1024 píxels i totes aquestes estan centrades a la matriu.

Les imatges del DataSet estan etiquetades amb la següent informació que consta d'un total de 7 columnes on es defineix:

Columna	Descripció	Valors
1	ID de referència de la imatge	
2	Característica del teixit de fons	**F**- gras **G**- Glandulars grassos **D**- Glandulars dens

3	Tipus d'anomalia que es detecta	**CALC**- Calcificació. **CIRC**- Masses circumscrites ben definides. **SPIC**- Masses espiculades. **MISC**- Altres masses mal definides. **ARCH**- Distorsió de la arquitectura. **ASYM**- Asimetria. **NORM**- Normal.
4	Severitat de la anomalia	**B**- Benigne **M**- Maligne
5	Posició x de la anomalia a la imatge	Referència esquerre inferior en píxels
6	Posició y de la anomalia a la imatge	Referència esquerre inferior en píxels
7	Radi aproximat en píxels que engloba la anomalia	

L'ús d'aquest DataSet esta sota el llicenciament:

MAMMOGRAPHIC IMAGE ANALYSIS SOCIETY
MiniMammographic Database

LICENCE AGREEMENT

This is a legal agreement between you, the end user and the Mammographic Image Analysis Society ("MIAS"). Upon installing the MiniMammographic database (the "DATABASE") on your system you are agreeing to be bound by the terms of this Agreement.

1. GRANT OF LICENCE

MIAS grants you the right to use the DATABASE, for research purposes ONLY. For this purpose, you may edit, format, or otherwise modify the DATABASE provided that the unmodified portions of the DATABASE included in a modified work shall remain subject to the terms of this Agreement.

2. COPYRIGHT

The DATABASE is owned by MIAS and is protected by United Kingdom copyright laws, international treaty provisions and all other applicable national laws. Therefore you must treat the DATABASE like any other copyrighted material. If the DATABASE is used in any publications then reference must be made to the DATABASE within that publication.

3. OTHER RESTRICTIONS

You may not rent, lease or sell the DATABASE.

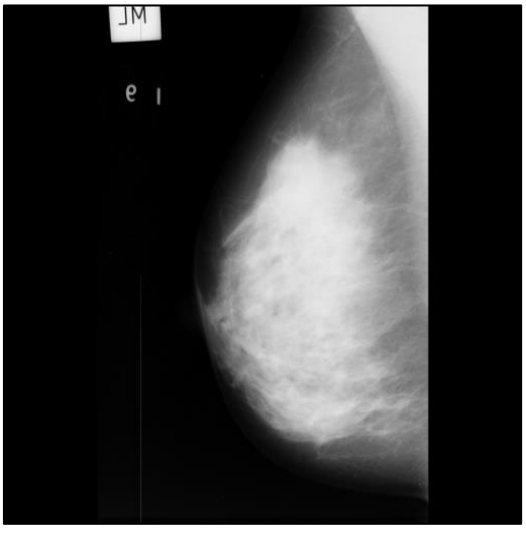
4. LIABILITY

To the maximum extent permitted by applicable law, MIAS shall not be liable for damages, other than death or personal injury, whatsoever (including without limitation, damages for negligence, loss of business, profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use this DATABASE, even if MIAS has been advised of the possibility of such damages. In any case, MIAS's entire liability under this Agreement shall be limited to the amount actually paid by you or your assignor, as the case may be, for the DATABASE.

On s'obliga a seguir la seva normativa afegint la seva referència en la memòria de qualsevol publicació que utilitzi les dades.

A continuació es mostren un parell d'exemples del contingut del DataSet:

Imatge mdb003.pgm



Columna 1: mdb003

Columna 2: **D**- Glandulars dens

Columna 3: **NORM**- Normal

Columna 4:

Columna 5:

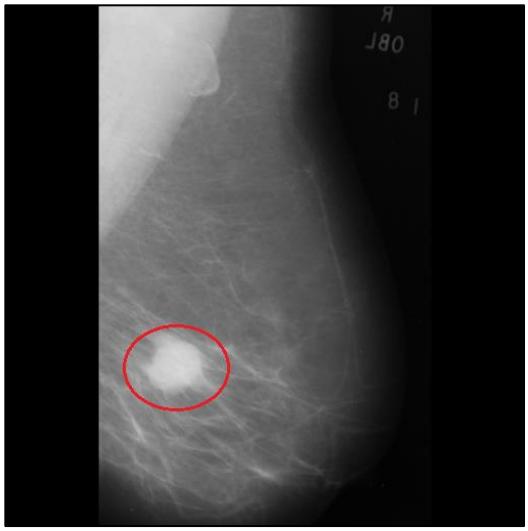
Columna 6:

Columna 7:

Aquesta imatge al ser amb nombre imparell (mbd003) indica que és de lateralitat dretana. Com es pot observar el seu teixit de fons es glandular dens i

no té cap anomalia ja que esta etiquetada com normal. Al no tenir cap anomalia la resta de camps estan buits.

Imatge mdb028.pgm



Columna 1: mdb028

Columna 2: **F**- gras

Columna 3: **CIRC**- Mas. circ. ben def.

Columna 4: **M**- Maligne

Columna 5: 338 píxels

Columna 6: 314 píxels

Columna 7: 56 píxels

Aquesta imatge al ser de nombre parell (mdb028), indica que és de lateralitat esquerrana. El seu teixit de fons és gras i com es pot observar té una anomalia de massa circular ben definida, està catalogat com maligne. Dins de la imatge s'ha encerclat en vermell on s'ubica la anomalia. Aquesta anomalia està a la posició 338 – 314 amb un radi de 56 píxels.

3.5 - Imatges .PGM

Per la realització de l'estudi del projecte utilitzarem imatges del tipus PGM (Portable Gray Map Format), ja que el DataSet utilitzat conté les imatges en aquest format.

El format PGM emmagatzema les imatges en escala de grisos i és àmpliament utilitzat per investigadors en projectes de processament d'imatges, degut a la seva simplicitat i eficiència.

Utilitza 8 bits per píxels si el seu valor màxim de gris és de 255 i 16 bits si el seu valor màxim va entre 255 i 65536.

Una imatge en format PGM conté text pla i pot ser modificat amb un simple processador de textos, encara que també existeix una versió en binari que no pot ser llegida per un processador de textos normal.

El format d'una imatge .PGM és el següent:

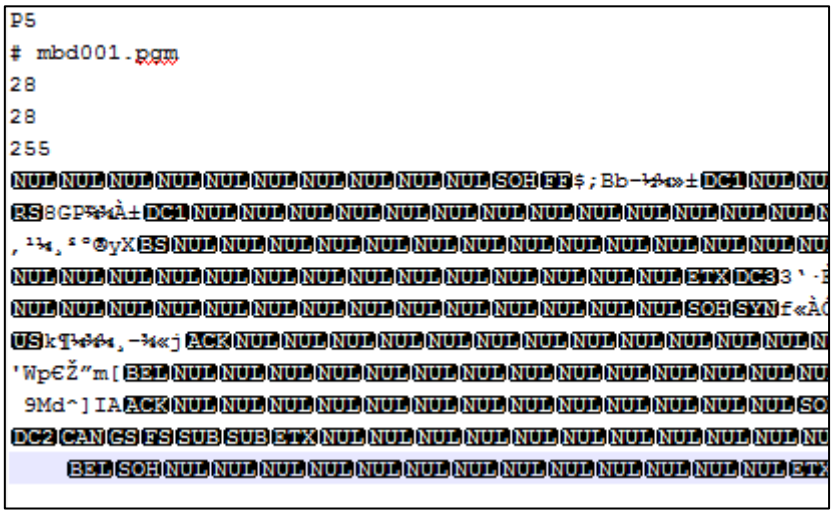
- Primera línia: apareix un valor que indica el format de la imatge.
    - P2: els valors dels píxels de la imatge venen en format ASCII (és a dir, xifres numèriques entre 0 i 255 o 0 i 65536)
    - P5: els valors dels píxels de la imatge venen en format binari (és a dir, la informació de cada píxel ve representada en un bytes).
- Segona línia: a partir d'aquesta línia poden venir una o varies línies que si porten al davant un #, son considerades comentaris.
- Després del darrer comentari: Apareixen 2 números enters separats per un espai que corresponen a la amplada i alçada de la imatge en píxels.
- Línia següent: apareix un numero enter que indica la màxima quantitat de nivells de grisos que suporta la imatge. Generalment 255 però pot arribar fins a 65536.
- A continuació: apareix la informació de cadascun dels píxels de la imatge. En el format P2 o P5 (definits en la primera línia). Venen tants nombres com mida de amplada per alçada.

Exemples de imatge .pgm:

```
P2
# feep.pgm
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
3 0 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
3 0 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
3 0 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Com es pot observar aquesta imatge és de tipus P2 o també anomenat PGM pla. La següent línia conté un comentari amb el nom de la imatge. La línia següent indica que la imatge és de mida de 24x7 píxels i la següent el nombre màxim del valor del gris. A continuació és defineixen tots els píxels de la imatge amb valors entre el 0 i el 15 separats per un espai.

L'exemple següent equivaldria a una imatge en format .PGM però de tipus P5. La imatge té un comentari i una mida de 28x28 píxels, amb una valor màxim d'escala de grisos de 255. A continuació es poden veure tots els píxels en format binari.



Com es pot observar aquest tipus de format ens serà molt útil per a realitzar l'estudi del projecte ja que serà fàcilment adaptable a Deep Learning i com una matriu de números d'entrada a la xarxa neuronal que definim.

4 - Implementació del prototip

4.1 - Cas d'estudi

Coneguts tos els elements necessaris, tan teòrics com d'eines de programació, aquest capítol es centra en al desenvolupament d'uns prototips basats en Python i Nolearn per l'aplicació de tècniques Deep Learning en imatges, i més concretament en imatges mèdiques com són les mamografies.

Amb l'objectiu d'introduir i entendre com funciona Deep Learning en l'entorn Python, es realitzen una sèrie de proves de desenvolupament sobre el DataSet MNIST que s'explica al capítol 4.2.1 de la memòria. Aquestes primeres proves serviran per realitzar la comprensió de forma més detalla del desenvolupament d'una xarxa neuronal, que posteriorment serà entrenada per a predir possibles resultats.

El desenvolupament del prototip es realitzarà amb 2 tècniques:

- Nolearn.DBN
- Nolearn.Lasagne

Realitzades les primeres proves, el projecte es centra en l'aplicació de Deep Learning en imatges mediques per intentar detectar-hi característiques i/o anomalies.

S'estableixen 3 casos d'estudi per tal d'analitzar els seus resultats:

- **Identificar la característica del teixit de fons de la mama**.
  L'estudi es basa en la columna 2 del DataSet on s'identifiquen els valors:
    - F – Gras
    - G – Glandulars grassos
    - D – Glandulars dens

- **Detectar l'aparició d'anomalies en una mamografia.**
  L'estudi es basa en la columna 3 del DataSet on s'indica el tipus d'anomalia que es detecta a la imatge:
    - NORM – Normal
    - ANORMAL – La resta (CALC, CIRC, SPI, MISC, ARCH, ASYM)

- **Identificar l'anomalia de masses circumscrites ben definides**.
  L'estudi es basa en la columna 3 del DataSet on s'indica el tipus d'anomalia. Ens centrem en la anomalia CIRC ja que és la mes fàcil de detectar de forma visual degut a que és una estructura ben definida.
    - CIRC – Amb anomalia de masses circumscrites ben definides.
    - No – La resta (CALC, SPIC, MISC, ARCH, ASYM, NORM)

4.2 - Desenvolupament

4.2.1- Proves realitzades

Nolearn DBN amb DataSet MNIST en Python 2

Per a la realització del primer cas pràctic s'utilitzarà un DataSet anomenat MNIST. Aquest DataSet està composat per aproximadament unes 66.000 imatges d'una mida de 28x28 píxels, on cada imatge porta un dígit numèric manuscrit. La idea és poder detectar quin dígit manuscrit hi ha en cada imatge.

Per la creació de la xarxa neuronal identifiquem:

- Tipologia: Es fa servir una xarxa tipus DBN (Deep Belief Network) amb Python 2.

- Neurones d'entrada: Una per a cada píxel (al ser imatges en blanc i negre), 28x28 = 784 píxels.
- Neurones de sortida: Una per a cada resultat valor, és a dir, 10 neurones. Dels dígits de 0 a 9.
- Capes de neurones: Es defineix segons prova/error.
- Nombre de neurones per capes: Es defineix segons prova/error.

Definida l'estructura de la xarxa neuronal que es vol crear, es comença el seu desenvolupament en Python2 utilitzant nolearn.dbn.

Primer de tot s'importen totes les llibreries necessàries per al desenvolupament. Com veiem s'importen les següents llibreries:

- **train_test_split:** Funció de la llibreria sklearn que s'utilitza per generar les jocs de dades tant d'entrenament com de test.
- **classification_report:** Funció de la llibreria sklearn que s'utilitza per a mostrar per pantalla un resum dels resultats de la xarxa creada.
- **datasets:** Classe de la llibreria sklearn que s'utilitza per a la descarrega i la creació del DataSet que utilitzarem per entrenar i provar la xarxa neuronal creada.
- **DBN:** Classe de la llibreria nolearn encarregada de la creació de la xarxa neuronal. Engloba les diferents parts de la xarxa, com son la definició, la creació, l'entrenament, i la prova.
- **np:** Llibreria numpy que s'utilitza per a mostrar els resultats obtinguts.
- **cv2:** Llibreria cv2 que s'utilitza per a mostrar imatges i esperar lectures de teclat.

```
# Importació de totes les llibreries necessàries
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report
from sklearn import datasets
from nolearn.dbn import DBN
import numpy as np
import cv2
# Fi importació llibreries
```

El següent pas és la creació del DataSet que s'utilitza. El DataSet el descarregarem amb la funció .fech_mldata(), i s'anomena MNIST Original.

```
# Descarrega de les dades MNIST, Dataset amb imatges del 0 al 9
print ( "######## Descarregant imatges MNIST...")
dataset = datasets.fetch_mldata("MNIST Original")
print ( "######## Descarrega finalitzada...")
# Fi de la descarrega de dades
```

Un cop tenim disponibles les dades, es generen les dades d'entrenament i de test. Les dades de test suposaran un 33% de les dades generals (definit amb test_size=0.33). Les dades d'entrenament i de test es generen a partir de dos estructures, dataset.data i dataset.target. Dataset.data conté la informació de totes les imatges, i l'estructura dataset.target conté la classe a la que correspon cada imatge (els nombres de 0 a 9).

Es generen les següents estructures:

- **trainX:** Estructura amb totes les imatges d'entrenament.
- **testX:** Estructura amb totes les imatges de test.
- **trainY:** Estructura amb la classificació de l'estructura d'entrenament.
- **testY:** Estructura amb la classificació de l'estructura de test.

```
# Creació dels datasets d'entrenament i els dataset de test
# L'entrenament es crea amb el 66% de les dades i el test amb la resta.
# S'utilitza funció ja predefinida de sklearn
print ( "######## Generant DS d'entrenament i test...")
(trainX, testX, trainY, testY) = train_test_split(
     dataset.data / 255.0, dataset.target.astype("int0"), test_size = 0.33)
print ( "######## Fi de la generació...")
# Fi de la creació de l'entrenament
```

El següent pas és la definició de l'estructura de la xarxa neuronal que es vol entrenar. Per la definició de la xarxa s'utilitza DBN importada de la llibreria nolearn, en la que es defineixen les següents dades:

- **Estructura de capes i neurones de la xarxa:** Es defineixen 4 capes. La primera, la capa d'entrada, es defineix amb tantes neurones com píxels té la imatge (784). Es defineixen dos capes ocultes, una de 100 neurones i una altre de 50 neurones. Finalment la capa de sortida es defineix amb 10 neurones, una per a cada dígit possible (de 0 a 9).
- **Learn_rates:** Especifica la velocitat d'aprenentatge, que és essencialment la mida dels passos al descens del gradient. A mides més

petites segurament la precisió sigui major però també el temps de càlcul serà superior. Es defineix amb valor 0.2.

- **Learn_rate_decays:** Especifica el factor de la tassa d'aprenentatge inicial que es multiplicarà després de cada iteració de formació. Es defineix amb valor 0.9.
- **Epochs:** Especifica el nombre de vegades que una xarxa iterarà sobre tots els exemples d'entrenament per a minimitzar la funció de cost. Un valor més alt millora la precisió però requereix de més temps de càlcul. Es defineix amb valor 2.
- **Verbose:** Especifica com volem visualitzar el progres d'entrenament de la xarxa. Es defineix amb valor 1.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1], 100,50, 10],
      learn_rates = 0.2,
      learn_rate_decays = 0.9,
      epochs = 2,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

Definida com serà la xarxa neuronal el següent pas és entrenar-la. Això s'aconsegueix amb la funció dbn.fit(), on se l'indica les dades d'entrenament (trainX – dades, trainY - Classificació).

Un cop entrenada la xarxa neuronal ja esta disponible per a fer prediccions, i això s'aconsegueix amb la funció dbn.predict(), on s'especifica per paràmetre les dades de test a predir i retorna la classificació de les prediccions que ha realitzat.

```
# Inici de l'entrenament de la xarxa DBN
print ( "######## Entrenant la xarxa DBN...")
dbn.fit(trainX, trainY)
print ( "######## Fi de l'entrenament...")
# Fi de l'entrenament de la xarxa DBN

# Classificació de les dades de test un cop la xarxa ha estat entrenada
print ( "######## Classificant el DS de test...")
preds = dbn.predict(testX)
print ( "######## Fi de la classificació...")
# Fi de la classificació
```

A continuació es visualitzen els resultats obtinguts de la predicció de la xarxa neuronal que s'ha entrenat. La funció classification_report de la llibreria sklearn, mostra un resum de les prediccions obtingudes i les compara amb la realitat.

Per veure exemples concrets, s'agafen 10 imatges de test a l'atzar per mostrar-les per pantalla, veient la seva predicció i la seva classificació real.

```
# Generació del report dels elements de test classificats
# Funció ja definida al paquet sklearn
print classification_report(testY, preds)
# Fi del report

# Es mostren 10 imatges a l'atzar per veure el seu resultat
for i in np.random.choice(np.arange(0, len(testY)), size = (10,)):

      pred = dbn.predict(np.atleast_2d(testX[i]))
      image = (testX[i] * 255).reshape((28, 28)).astype("uint8")
      print "L'imatge és:  {0}, i la classifiació ha estat:  {1}".format(testY[i], pred[0])
      cv2.imshow("Imatge", image)
      cv2.waitKey(0)
```

Com es pot observar, la funció classification_report ens mostra el resum que veiem a continuació:

	precision	recall	f1-score	support
0	0.95	0.98	0.96	2256
1	0.98	0.98	0.98	2643
2	0.95	0.93	0.94	2257
3	0.95	0.94	0.94	2364
4	0.93	0.96	0.95	2223
5	0.97	0.92	0.95	2103
6	0.98	0.95	0.97	2287
7	0.96	0.95	0.95	2424
8	0.93	0.92	0.93	2227
9	0.90	0.95	0.92	2316
avg / total	0.95	0.95	0.95	23100

Analitzant els resultats es pot observar que obtenim una precisió global d'un 95%, resultat obtingut de la classificació del test de 23100 imatges. De forma individual la predicció del numero 1 té un encert d'un 98%, per contra el numero 9 el seu grau d'encert és del 90%.

A continuació es visualitzen diferents exemples classificats i quina ha estat la seva predicció. Si analitzem la línia 5, es veu com una imatge amb nombre 2 s'ha classificat com a 9.

```
L'imatge és:  7, i la classifiació ha estat:  7
L'imatge és:  7, i la classifiació ha estat:  7
L'imatge és:  5, i la classifiació ha estat:  5
L'imatge és:  2, i la classifiació ha estat:  9
L'imatge és:  1, i la classifiació ha estat:  1
L'imatge és:  7, i la classifiació ha estat:  7
L'imatge és:  8, i la classifiació ha estat:  8
```

Un cop obtinguts els primers resultats, l'objectiu és millorar la precisió de la xarxa neuronal creada. Per tal de augmentar el grau de precisió hi ha diferents alternatives, però a mode d'exemple en primer lloc ens centrarem en l'augment del nombre de iteracions (epochs), per tal de que es pugui reduir la funció de cost. Als paràmetres d'entrada de la xarxa s'especificarà que el nombre d'epochs és 10.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1], 100,50, 10],
      learn_rates = 0.2,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

Amb aquesta configuració, es pot observar que s'ha passat d'un 95% d'encert a un 97%, això reafirma que, en aquest cas, la funció de cost s'ha pogut reduir gràcies a l'augment de les iteracions aplicades sobre les dades d'entrenament.

	precision	recall	f1-score	support
0	0.98	0.98	0.98	2232
1	0.99	0.99	0.99	2586
2	0.96	0.97	0.96	2301
3	0.97	0.96	0.97	2371
4	0.98	0.96	0.97	2308
5	0.97	0.96	0.97	2087
6	0.97	0.98	0.98	2182
7	0.97	0.97	0.97	2387
8	0.96	0.96	0.96	2300
9	0.95	0.96	0.96	2346
avg / total	0.97	0.97	0.97	23100

Partint del primer exemple on el nombre d'iteracions era igual a dos, l'objectiu es millorar el grau de precisió modificant l'estructura en nombre de neurones de la xarxa que s'ha definit. Les capes d'entrada i de sortida no podran variar, però

si les capes ocultes. Es defineix una nova estructura amb 2 capes ocultes una de 500 i una altre de 50 neurones.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1], 500,50, 10],
      learn_rates = 0.2,
      learn_rate_decays = 0.9,
      epochs = 2,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

S'observa que els resultats també han millorat respecte a la primera configuració, passant d'un 95% de precisió fins a un 96%. Aquesta millora és deguda a que amb un nombre superior de neurones en les capes ocultes es pot arribar a analitzar més detall de la imatge d'entrenament.

	precision	recall	f1-score	support
0	0.99	0.96	0.97	2302
1	0.97	0.99	0.98	2524
2	0.98	0.92	0.95	2364
3	0.93	0.96	0.94	2325
4	0.96	0.96	0.96	2247
5	0.96	0.95	0.96	2084
6	0.96	0.98	0.97	2268
7	0.97	0.97	0.97	2393
8	0.94	0.95	0.94	2289
9	0.95	0.95	0.95	2304
avg / total	0.96	0.96	0.96	23100

Per últim, es fusionen aquestes dos configuracions de millora. Es modifica l'estructura de neurones de la xarxa i s'augmenta el nombre d'iteracions.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1], 500,50, 10],
      learn_rates = 0.2,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

Tal i com es veu, s'augmenta encara més la precisió d'encert de la xarxa que s'està entrenant, fins a arribar al 98 % de precisió.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2249
1	0.99	0.98	0.99	2579
2	0.97	0.98	0.98	2305
3	0.98	0.98	0.98	2358
4	0.99	0.97	0.98	2290
5	0.98	0.98	0.98	2086
6	0.99	0.99	0.99	2203
7	0.98	0.98	0.98	2451
8	0.97	0.97	0.97	2245
9	0.95	0.98	0.97	2334
avg / total	0.98	0.98	0.98	23100

És obvi que aquets punts de millora no son els únics que es poden aplicar a la xarxa que s'està entrenant. Tal com s'explica al capítol 2 de la memòria, la configuració òptima d'una xarxa neuronal per a Deep Learning es basa, en moltes ocasions, en exercicis de prova i error intentant cercar la xarxa més òptima per al problema que es vol resoldre.

Nolearn Lasagne amb DataSet MNIST en Python 3

Un cop realitzades les primeres proves amb nolearn.dbn, s'explica la utilització d'un altre mètode també disponible al conjunt de llibreries nolearn com és nolearn.lasagne. Nolearn recomana utilitzar aquest mètode envers a DBN ja que en les properes versions de nolearn deixarà d'estar suportat. Nolearn conté una sèrie d'abstraccions entorn a llibreries de xarxes neuronals i especialment de lasagne, això simplificarà el seu desenvolupament envers a utilitzar lasagne directament.

Per l'execució d'aquest exemple també es farà us del DataSet explicat en l'exemple anterior, MNIST. Per la creació de la xarxa neuronal identifiquem:

- Tipologia: Es fa servir una xarxa tipus full connection (es defineix en el tipus de capa com veurem més endavant) amb Python 3.
- Neurones d'entrada: Una per a cada píxel (al ser imatges en blanc i negre), 28x28 = 784 píxels.
- Neurones de sortida: Una per a cada resultat, és a dir, 10 neurones. Dels dígits de 0 a 9.

- Capes de neurones: Es realitzen les mateixes proves que a l'exemple DBN, per a la seva comparativa.
- Nombre de neurones per capes: Es realitzen les mateixes proves que a l'exemple DBN, per a la seva comparativa.

Un cop definida l'estructura de la xarxa neuronal, es desenvolupa el seu funcionament en Python 3 utilitzant nolearn.lasagne.

Com a l'exemple anterior s'importen les llibreries necessàries per al desenvolupament de la xarxa que s'ha definit. S'incorporen les següents llibreries i s'expliquen les novetats respecte a DBN.

- **NeuralNet:** Classe de la llibreria nolearn.lasagne encarregada de la creació de la xarxa neuronal. Engloba les diferents parts de la xarxa, com són la definició, la creació, l'entrenament i la prova.
- **Softmax:** Funció no lineal escollida per a fer que la capa no sigui lineal.
- **InputLayer:** Del paquet lasagne.layers importem la capa InputLayer, utilitzada a la definició de la xarxa com a capa d'entrada de dades.
- **DenseLayer:** Del paquet lasagne.layers importem la capa DenseLayer, utilitzada per a la definició de la resta de capes del nostre exemple. DenseLayer és un tipus de capa que indica que totes les neurones tenen connexions amb la resta de neurones de la següent capa. Com es veurà més endavant hi ha diferents formes de definir cada capa.
- La resta de llibreries de sklearn ja es coneixen de l'exemple anterior.

```
# Importació de totes les llibreries necessàries
from lasagne.layers import DenseLayer
from lasagne.layers import InputLayer
from lasagne.nonlinearities import softmax
from nolearn.lasagne import NeuralNet

from sklearn.metrics import classification_report
from sklearn import datasets
from sklearn.cross_validation import train_test_split
# Fi importació llibreries
```

Un cop importades les llibreries necessàries, s'obtenen les dades i es generen els jocs d'entrenament i de test. Aquets passos son exactament igual que en l'exemple DBN.

```
# Descarrega de les dades MNIST, Dataset amb imatges del 0 al 9
print ( "######## Descarregant imatges MNIST...")
dataset = datasets.fetch_mldata("MNIST Original")
print ( "######## Descarrega finalitzada...")
# Fi de la descarrega de dades

# Creació dels datasets d'entrenament i els dataset de test
# L'entrenament es crea amb el 66% de les dades i el test amb la resta.
# S'utilitza funció ja predefinida de sklearn
print ( "######## Generant DS d'entrenament i test...")
(trainX, testX, trainY, testY) = train_test_split(
      dataset.data / 255.0, dataset.target.astype("int32"), test_size = 0.33)
print ( "######## Fi de la generació...")
# Fi de la creació de l'entrenament
```

Continuem per la definició de les diferents capes de les que estarà composada la xarxa neuronal. Si ens fixem en la definició de LayersCNN, observem que tenim quatre capes, la d'entrada (InputLayer), dos d'ocultes (DenseLayer), i la capa de sortida (DenseLayer) amb una funció de sortida no lineal softmax.

Tal i com es veu en la definició, totes les capes excepte la d'entrada estan definides com a DenseLayer, amb la qual cosa no s'està utilitzant la definició de convolució, ja que totes les neurones entre capes tenen interconnexió entre elles.

Cada capa conté una sèrie de variables. Si ens fixem la capa del tipus InputLayer, conté una variable del tipus shape, on s'indica com a segon paràmetre el nombre de neurones d'entrada. A les capes del tipus DenseLayer, s'indica amb la variable num_units el nombre de neurones per la capa en qüestió. A més a més, a l'exemple, se li esta indicant que la capa de sortida conté una funció no lineal com és softmax.

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
    (InputLayer, {'shape': (None, trainX.shape[1])}),
    (DenseLayer, {'num_units': 100}),
    (DenseLayer, {'num_units': 50}),
    (DenseLayer, {'num_units': 10, 'nonlinearity': softmax}),
]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes
```

Existeixen diversos tipus de capes i cada tipus disposa de diverses variables i característiques per a la seva parametrització. En la documentació oficial del projecte de lasagne es pot consultar tota la seva informació:

http://lasagne.readthedocs.io/en/latest/modules/layers.html

A continuació s'anomenen els tipus de capes més rellevants:

- **InputLayer:** Aquesta capa és utilitzada per a definir la capa d'entrada de les dades.
- **DenseLayer**: Capa amb total connexió de les seves neurones amb les capes posteriors.
- **NINLayer**: Capa igual que DenseLayer amb l'objectiu de fer convolucions entre diferents dimensions.
- **Conv1DLayer**: Aquesta capa realitza una convolució d'1 dimensió en les seves neurones d'entrada, opcionalment es pot aplicar una BIA i aplicar una funció no lineal a la sortida.
- **Conv2DLayer**: Igual que l'anterior però per a dades de 2 dimensions.
- **TransposedConv2DLayer**: Igual que l'anterior però ho realitza de forma transposada.
- **DilatedConv2Layer**: Igual que Conv2DLayer però aplica filtres dilatats.
- **MaxPool1DLayer**: Capa que aplica max-pooling d'1 dimensió sobre una entrada de 3 dimensions.
- **MaxPool2DLayer**: Capa que plica max-polling de 2 dimensions sobre una entrada de 4 dimensions.
- **DropoutLayer**: Capa que estableix a valor 0 neurones amb una certa probabilitat (p).

Continuant amb les proves, el següent pas és la definició de la xarxa neuronal. Per la definició de la xarxa s'utilitza la funció NeuroalNet importada de la llibreria nolearn.lasagne, en la que es defineixen les següents dades:

- **layers**: Defineix les capes de la xarxa neuronal. A l'exemple desenvolupat correspon a la variable layersCNN.
- **max_epochs**: Especifica el nombre de vegades que una xarxa iterarà sobre tots els exemples d'entrenament per a minimitzar la funció de cost. Un valor més alt millora la precisió però requereix de més temps de càlcul. Es defineix amb valor 2.

- **update_learning_rate**: Especifica la velocitat d'aprenentatge, que essencialment és la mida dels passos al descens del gradient. A mides més petites segurament la precisió sigui major però també el temps de càlcul serà superior. Es defineix amb valor 0.2
- **verbose**: Especifica com volem visualitzar el progrés d'entrenament de la xarxa. Es defineix amb valor 2.

```
# Configuració de la xarxa CNN
print ( "######## Configurant CNN LASAGNE...")
net0 = NeuralNet(
    layers=layersCNN,
    max_epochs=2,
    update_learning_rate=0.2,
    verbose=2,
)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa CNN
```

Un cop definida la xarxa, el següent pas és entrenar-la. Això s'aconsegueix amb la funció .fit(), on se l'hi indica les dades d'entrenament trainX(dades) i trainY(classificació).

```
# Inici de l'entrenament de la xarxa CNN
print ( "######## Entrenant la xarxa CNN LASAGNE...")
net0.fit(trainX, trainY)
print ( "######## Fi de l'entrenament...")
# Fi de l'entrenament de la xarxa CNN
```

Finalitzat l'entrenament, ja tenim la xarxa disponible per a realitzar prediccions, i això es realitza amb la funció .predict(), on s'especifica per paràmetre les dades de testX a predir i retorna la classificació de les prediccions realitzades.

Per veure els resultats de les prediccions utilitzarem la funció classification_report del paquet sklearn, on se l'hi passarà les prediccions obtingudes amb les dades reals.

```
# Classificació de les dades de test un cop la xarxa ha estat entrenada
print ( "######## Classificant el DS de test...")
preds = net0.predict(testX)
print ( "######## Fi de la classificació...")
# Fi de la classificació

# Generació del report dels elements de test classificats
# Funció ja definida al paquet sklearn
print (classification_report(testY, preds))
# Fi del report
```

Observem els resultats següents:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	2259
1	0.95	0.99	0.97	2605
2	0.94	0.96	0.95	2302
3	0.97	0.94	0.95	2382
4	0.97	0.94	0.96	2262
5	0.97	0.92	0.94	2115
6	0.97	0.95	0.96	2286
7	0.98	0.95	0.96	2327
8	0.91	0.96	0.93	2272
9	0.94	0.95	0.94	2290
avg / total	0.96	0.96	0.96	23100

S'obté un percentatge de precisió del 96% (pràcticament igual que DBN on es va obtenir un 95%), classificant un total de 23100 imatges. De forma individual el nombre 7 obté una precisió del 98% i per contra el 8 obté la precisió més baixa amb un 91%.

Al igual que amb l'exemple DBN es proven diferents configuracions per veure si obtenim millors resultats.

Realitzant la mateixa definició de la xarxa neuronal, tant sols augmentant el nombre de iteracions (max_epochs=10) per a tal d'intentar reduir la funció de cost.

```
# Definició de les capes de la xarxa
print ( "######### Definició de les capes...")
layersCNN = [
    (InputLayer, {'shape': (None, trainX.shape[1])}),
    (DenseLayer, {'num_units': 100}),
    (DenseLayer, {'num_units': 50}),
    (DenseLayer, {'num_units': 10, 'nonlinearity': softmax}),
]
print ( "######### Fi de la definició de les capes...")
# Fi de la definició de les capes

# Configuració de la xarxa CNN
print ( "######### Configurant CNN LASAGNE...")
net0 = NeuralNet(
    layers=layersCNN,
    max_epochs=10,
    update_learning_rate=0.2,
    verbose=2,
)
print ( "######### Fi de la configuració...")
# Fi de la configuració de la xarxa CNN
```

S'obté una predicció del 95%, pràcticament igual que amb 2 iteracions que era d'un 96% (aquesta variabilitat pot estar condicionada per les dades d'entrenament obtingudes), amb la qual cosa amb aquesta xarxa creada, la funció de cost no es redueix augmentant les iteracions.

	precision	recall	f1-score	support
0	0.98	0.97	0.97	2309
1	0.99	0.97	0.98	2641
2	0.95	0.96	0.95	2320
3	0.94	0.94	0.94	2365
4	0.96	0.94	0.95	2176
5	0.95	0.94	0.94	2120
6	0.96	0.97	0.96	2259
7	0.98	0.92	0.95	2413
8	0.90	0.97	0.93	2198
9	0.92	0.95	0.93	2299
avg / total	0.95	0.95	0.95	23100

Per últim es realitza una altre prova. En aquest cas modificant la definició de la xarxa creada, i augmentant a 500 neurones la primera capa oculta.

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
    (InputLayer, {'shape': (None, trainX.shape[1])}),
    (DenseLayer, {'num_units': 500}),
    (DenseLayer, {'num_units': 50}),
    (DenseLayer, {'num_units': 10, 'nonlinearity': softmax}),
]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes

# Configuració de la xarxa CNN
print ( "######## Configurant CNN LASAGNE...")
net0 = NeuralNet(
    layers=layersCNN,
    max_epochs=2,
    update_learning_rate=0.2,
    verbose=2,
)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa CNN
```

Els resultats obtinguts de forma global, tampoc milloren la primera configuració d'escrita (encara que si en alguns valors com ara pot ser al nombre 8), i s'arriba fins a un total d'un 96% d'encert.

	precision	recall	f1-score	support
0	0.97	0.99	0.98	2336
1	0.97	0.98	0.98	2583
2	0.96	0.95	0.96	2277
3	0.97	0.93	0.95	2300
4	0.94	0.97	0.95	2198
5	0.98	0.92	0.95	2073
6	0.95	0.99	0.97	2287
7	0.97	0.96	0.97	2376
8	0.93	0.96	0.94	2326
9	0.96	0.93	0.95	2344
avg / total	0.96	0.96	0.96	23100

A banda d'aquestes possibles configuracions existeixen d'infinites possibilitats, que segur que poden millorar el % d'encert. La idea seria poder combinar noves tipus de capes, amb d'altres nombres de capes i neurones per a poder reduir la funció de cost.

4.2.2- Desenvolupament del prototip

Vistes les primeres proves, es continua amb el desenvolupament del prototip destinat a aconseguir predir característiques i anomalies d'imatges mèdiques, més concretament de mamografies del DataSet MIAS explicat al capítol 3 de la memòria. En aquest capítol veurem tots els passos necessaris per al desenvolupament del prototip, sense passar a analitzar els resultats obtinguts ja que s'explicaran en profunditat al capítol 5 de la memòria.

Per la creació de la xarxa neuronal identificarem:

- Tipologia: Com s'ha fet a les proves anteriors es realitzen 2 prototips, un desenvolupat amb nolearn.dbn en Python 2 i un altre amb nolearn.lasagne en Python 3.
- Neurones d'entrada: Serà una per a cada píxel de las imatges de mamografies. Al ser les imatges en format .pgm (escala de grisos), es requereix una neurona per píxel. El nombre de píxels d'entrada variarà segons les diferents proves realitzades com es veurà al capítol 5.

- Neurones de sortida: Una neurona per a cada possible resultat de la classificació de les imatges. Els exemples que es realitzaran seran de 2 o 3 neurones segons la informació a classificar.
- Capes de neurones: És defineix segons prova/error.
- Nombre de neurones per capes: És defineix segons prova/error.

Nolearn DBN amb DataSet MIAS en Python 2

El primer pas és importar totes les llibreries necessàries per al desenvolupament, la majoria ja vistes a l'exemple MNIST, excepte:

- **csv**: Llibreria per al tractament de fitxers csv (Comma separated values), amb la que es llegiran els fitxers csv que contenen la informació de les imatges i la seva classificació.
- **re**: Llibreria per operacions d'expressions regular, utilitzada per a poder llegir les imatges .pgm per transformar-les en un array.

```
# Importació de totes les llibreries necessàries
import csv
import re
import numpy
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report
from nolearn.dbn import DBN
import numpy as np
import cv2
# Fi importació llibreries
```

Es crea una funció anomenada read_pgm, que serà l'encarregada de poder llegir fitxers del tipus .pgm i fer la seva transformació en array d'informació per a poder tractar-la de forma correcta.

```
# Funció per la lectura d'una imatge .pgm i passar-la a numpy array
def read_pgm(filename, byteorder='>'):
    with open(filename, 'rb') as f:
        buffer = f.read()
    try:
        header, width, height, maxval = re.search(
            b"(^P5\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
    except AttributeError:
        raise ValueError("Not a raw PGM file: '%s'" % filename)
    return numpy.frombuffer(buffer,
                            dtype='u1' if int(maxval) < 256 else byteorder+'u2',
                            count=int(width)*int(height),
                            offset=len(header)
                            ).reshape((int(height), int(width)))

# Fi de la funció
```

Tot seguit es defineixen una sèrie de variables per a poder realitzar diferents jocs de proves que veurem al capítol 5. Aquesta parametrització de variables simplificarà totes les proves que es realitzin.

- **cvs_file**: Variable que indica el nom del fitxer cvs on estan ubicades les dades a carregar.
- **img_src**: Variable que indica la carpeta on estan ubicades les imatges que volem carregar.
- **img_pix**: Variable que indica la mida X en píxels de la imatge (imatges quadrades)
- **img_add**: Variable que indica si el joc de proves conté imatges addicionals. Si és S , carregarà les imatges addicionals.
- **img_add_csv**: Variable que indica el fitxer cvs de les imatges addicionals (tant sols si img_add='S').
- **img_add_src**: Variable que indica la carpeta on estan ubicades les imatges addicionals que volem carregar (tant sols si img_add='S').
- **neu_out**: Nombre de neurones que tindrà la capa de sortida de la xarxa neuronal creada.

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_massa.csv' # cvs de les dades a carregar
img_src='Imatges512/' # ruta de les imatges a carregar
img_pix = 512 # mida en pixels de les imatges (imatges quadrades)

img_add = 'N' # si tenim disponible imatges addicionals
img_add_csv = '' # csv de les imatges adiccionals
img_add_src ='/' # ruta de les imatges addicionals

neu_out= 3 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Definides les variables d'entrada, és defineixen 2 estructures, images i target, on s'emmagatzemaran les dades a carregar a la xarxa neuronal: imatges i classe a la que pertany.

Seguidament es llegeix el fitxer cvs definit prèviament a la variable cvs_file, i es carreguen totes les imatges a la variable images (row[0]) i la seva classificació a la variable target (row[1]). Observem que la ruta on estan ubicades les imatges és la definida a la variable img_src.

L'estructura del fitxer .cvs és molt simple:

Imatge1,classificació

Imatge2,classificació

Imatge3,classificació

...

ImatgeN,classificació.

Com s'explicarà al capítol 5 es realitzaran diferents tipus de proves, i és per això que tenim disponibles diferents fitxers .csv, segons el cas d'estudi que es vulgui executar.

```
# Definició de les 2 estructures per carregar les imatges i la classificació
target = []
images = []
# Fi de la definició de les variables target i images

# Lectura del fitxer .csv per a la carrega de les imatges
reader = csv.reader(open(cvs_file, 'rb'))
# Fi de la lectura del fitxer

# Carrega de les imatges i la seva classificació en les estructures target i images
print ( "######## Carregant dades MIAS...")
for index,row in enumerate(reader):
    image = read_pgm( img_src + row[0]+".pgm", byteorder='<')
    target.append(row[1])

    A = np.asarray(image).reshape(-1)
    x = np.array(A, np.uint8)

    images.append(x)
print ( "######## Carrega finalitzada...")
# Fi de la carrega de les imatges i la seva classificació
```

Addicionalment a la càrrega de les dades principals, s'ha definit la possibilitat de poder carregar dades extres a la xarxa neuronal. Això s'ha fet, tal i com s'explicarà al capítol 5, per a poder obtenir uns millors resultats a les diferents prediccions, afegint més imatges a l'entrenament de la xarxa. Com s'observa, s'ha parametritzat amb la variable img_add on si és igual a S, es realitzarà la carrega addicional d'imatges del cvs definit a img_add_cvs i on la ruta de les imatges és img_add_src.

```
# Si definime imatges adiccionals, s'afegeixen
if img_add == 'S':
    print ( "######## Carregant dades addicionals MIAS...")
    reader = csv.reader(open(img_add_csv, 'rb'))
    for index,row in enumerate(reader):
        image = read_pgm( img_add_src + row[0]+".pgm", byteorder='<')
        target.append(row[1])

        A = np.asarray(image).reshape(-1)
        x = np.array(A, np.uint8)

        images.append(x)
    print ( "######## Carrega finalitzada...")
# Fi Carrega d'imatges addicionals
```

Un cop tenim les dades carregades a les 2 variables, a l'igual que als exemples de proves, és generen les dades de test i d'entrenament amb la funció traint_test_split del paquet sklearn. Es continua generant el paquet de test amb el 33% de les dades disponibles.

```
# Creació de variables i i t amb array d'imatges i de classificacions
t = np.array(target)
i = np.array(images)
# Fi declaració de variables

# Creació dels datasets d'entrenament i els dataset de test
# L'entrenament es crea amb el 66% de les dades i el test amb la resta.
# S'utilitza funció ja predefinida de sklearn
print ( "######## Generant DS d'entrenament i test...")
(trainX, testX, trainY, testY) = train_test_split(
                    i, t.astype("int0"), test_size = 0.33)
print ( "######## Fi de la generació...")
# Fi de la creació de l'entrenament
```

Amb les dades d'entrenament i de test disponibles, es passa a definir la xarxa neuronal amb la funció DBN de nolearn. En l'exemple, es defineix una xarxa de 3 capes (entrada, oculta i sortida). La capa d'entrada té tantes neurones com píxels de la imatge, la capa oculta té un total de 500 neurones i la capa de sortida té tantes neurones com les definides en la variable neu_out (en aquest cas 3).

Es realitzen un total de 10 iteracions amb un learn_rates de 0.2 (conceptes ja explicats anteriorment).

Amb la funció .fit(), es realitza l'entrenament de la xarxa.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],500,neu_out],
      learn_rates = 0.2,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN

# Inici de l'entrenament de la xarxa DBN
print ( "######## Entrenant la xarxa DBN...")
dbn.fit(trainX, trainY)
print ( "######## Fi de l'entrenament...")
# Fi de l'entrenament de la xarxa DBN
```

Un cop la xarxa esta entrenada, la tenim disponible per a poder realitzar les prediccions. Aquesta és la tasca de la funció .predict(), on retorna un array de prediccions de les dades introduïdes per paràmetre.

Per últim, s'utilitza la funció classification_report, del paquet sklearn, per a mostrar un resum dels resultats obtinguts.

```
# Classificació de les dades de test un cop la xarxa ha estat entrenada
print ( "######## Classificant el DS de test...")
preds = dbn.predict(testX)
print ( "######## Fi de la classificació...")
# Fi de la classificació

# Generació del report dels elements de test classificats
# Funció ja definida al paquet sklearn
print classification_report(testY, preds)
# Fi del report
```

De forma addicional s'ha afegit un petit control, per a veure els resultats de 10 imatges a l'atzar, amb la seva classificació real i la classificació que s'ha predit.

```
# Es mostren 10 imatges a l'atzar per veure el seu resultat
for i in np.random.choice(np.arange(0, len(testY)), size = (10,)):
      pred = dbn.predict(np.atleast_2d(testX[i]))
      image = (testX[i]).reshape((img_pix, img_pix)).astype("uint8")
      print "La imatge actual és del tipus {0}, i la predicció és {1}".format(testY[i], p
      cv2.imshow("MIAS", image)
      cv2.waitKey(0)
```

Nolearn Lasagne amb DataSet MIAS en Python 3

Com a la resta de desenvolupaments, el primer pas és importar totes les llibreries necessàries. Totes les llibreries utilitzades ja son conegudes i han estat explicades en apartats anteriors.

```
# Importació de totes les llibreries necessàries
import csv
import re
from nolearn.lasagne import NeuralNet
from lasagne.layers import DenseLayer
from lasagne.layers import InputLayer
from lasagne.nonlinearities import softmax
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report
import numpy as np
import numpy
# Fi importació llibreries
```

Es defineix la funció read_pgm, que serà l'encarregada de poder llegir fitxers del tipus .pgm i fer la seva transformació en array d'informació per a poder tractar-la de forma correcta.

```
# Funció per la lectura d'una imatge .pgm i passar-la a numpy array
def read_pgm(filename, byteorder='>'):
    with open(filename, 'rb') as f:
        buffer = f.read()
    try:
        header, width, height, maxval = re.search(
            b"(^P5\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
    except AttributeError:
        raise ValueError("Not a raw PGM file: '%s'" % filename)
    return numpy.frombuffer(buffer,
                            dtype='u1' if int(maxval) < 256 else byteorder+'u2',
                            count=int(width)*int(height),
                            offset=len(header)
                            ).reshape((int(height), int(width)))
# Fi de la funció
```

Igual que en DBN, es defineixen una sèrie de variables per a dur a terme els diferents cassos d'estudi que s'analitzaran al capítol 5. Aquestes variables ja han sigut descrites amb anterioritat.

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_massa.csv' # cvs de les dades a carregar
img_src='Imatges512/' # ruta de les imatges a carregar
img_pix = 512 # mida en pixels de les imatges (imatges quadrades)

img_add = 'N' # si tenim disponible imatges addicionals
img_add_csv = '' # csv de les imatges adiccionals
img_add_src ='/' # ruta de les imatges addicionals

neu_out= 3 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Definides les variables d'entrada, és defineixen les estructures necessàries per a emmagatzemar les imatges i la classe a la que pertanyen.

Seguidament es llegeix el fitxer cvs que conté tota la informació i es carreguen les dades a analitzar a les estructures prèviament definides: images (row[0]) i target (row[1]). Observem que tant la ruta del fitxer cvs com la ruta de les imatges estan definides a les variables cvs_file i img_src respectivament.

```
# Definició de les 2 estructures per carregar les imatges i la classificació
target = []
images = []
# Fi de la definició de les variables target i images

# Lectura del fitxer .csv per a la carrega de les imatges
reader = csv.reader(open(cvs_file, 'r'))
# Fi de la lectura del fitxer

# Carrega de les imatges i la seva classificació en les estructures target i images
print ( "######## Carregant dades MIAS...")
for index,row in enumerate(reader):
    image = read_pgm( img_src + row[0]+".pgm", byteorder='<')
    target.append(row[1])

    A = np.asarray(image).reshape(-1)
    x = np.array(A, np.uint8)

    images.append(x)
print ( "######## Carrega finalitzada...")
# Fi de la carrega de les imatges i la seva classificació
```

En aquest cas també s'ha desenvolupat una carrega addicional d'informació que es parametritza amb la variable img_add. Amb aquesta càrrega addicional, tal i com s'explicarà al capítol 5, s'intentarà millorar els diferents resultats obtinguts en les prediccions realitzades, permetent entrenar la xarxa amb més imatges. Aquestes noves imatges seran creades a partir d'imatges ja existents.

```
# Si definime imatges adiccionals, s'afegeixen
if img_add == 'S':
    print ( "######## Carregant dades addicionals MIAS...")
    reader = csv.reader(open(img_add_csv, 'r'))
    for index,row in enumerate(reader):
        image = read_pgm( img_add_src + row[0]+".pgm", byteorder='<')
        target.append(row[1])

        A = np.asarray(image).reshape(-1)
        x = np.array(A, np.uint8)

        images.append(x)
    print ( "######## Carrega finalitzada...")
# Fi Carrega d'imatges addicionals
```

Amb les dades carregades, es generen les dades d'entrenament i de test amb la funció traint_test_split del paquet sklearn. Les dades de test, seran d'un 33% del total de les dades disponibles.

```
# Creació de variables i i t amb array d'imatges i de classificacions
t = np.array(target,np.int32)
i = np.array(images)
# Fi declaració de variables

# Creació dels datasets d'entrenament i els dataset de test
# L'entrenament es crea amb el 66% de les dades i el test amb la resta.
# S'utilitza funció ja predefinida de sklearn
print ( "######## Generant DS d'entrenament i test...")
(trainX, testX, trainY, testY) = train_test_split(
      i, t.astype("int32"), test_size = 0.33)
print ( "######## Fi de la generació...")
# Fi de la creació de l'entrenament
```

A continuació es defineix l'estructura de les capes de las que es composarà la xarxa neuronal. Es defineix la variable layersCNN, composada de 4 capes, una d'entrada (amb X neurones d'entrada), dos d'ocultes (100 i 50 neurones) i una de sortida (en aquest cas amb 3 neurones de sortida definides a la variable neu_out), a més de la funció no lineal softmax.

Totes les capes estan definides com a DenseLayer, excepte la capa d'entrada que es defineix com InputLayer. Això indica que totes les neurones de la xarxa tindran connexió amb les neurones de les capes posteriors, i no utilitzem la convolució.

Aquesta configuració de l'estructura de la xarxa pot ser modificada per tal de millorar les possibles prediccions que es realitzin.

Com s'ha comentat al desenvolupament de les proves amb MNIST, podem obtenir tota la informació i la definició de cada capa a la pàgina oficial de lasagne:

http://lasagne.readthedocs.io/en/latest/modules/layers.html

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
        (InputLayer, {'shape': (None, trainX.shape[1])}),
        (DenseLayer, {'num_units': 100}),
        (DenseLayer, {'num_units': 50}),
        (DenseLayer, {'num_units': neu_out, 'nonlinearity': softmax}),
    ]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes
```

Amb l'estructura de les capes de la xarxa ja creada, el següent pas és la definició de la xarxa neuronal. Per la definició s'utilitza la funció NeuralNet del paquet nolearn.lasagne.

Es defineix amb l'estructura de capes layersCNN, 2 iteracions de les dades i un learnint_rate del 0.2.

```
# Configuració de la xarxa
print ( "######## Configurant NOLEARN.LASAGNE...")
net1 = NeuralNet(
        layers=layersCNN,
        max_epochs=2,
        update_learning_rate=0.2,
        verbose=1,
    )
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa
```

Els passos següents son l'entrenament de la xarxa amb la funció .fit() passant per paràmetre les dades d'entrenament, i la realització de les prediccions de les dades de test amb la funció .predict().

Per últim, s'utilitza la funció classification_report del paquet sklearn per a visualitzar un resum dels resultats obtinguts.

```
# Inici de l'entrenament de la xarxa
print ( "######## Entrenant la xarxa NOLEARN.LASAGNE...")
net1.fit(trainX, trainY)
print ( "######## Fi de l'entrenament...")
# Fi de l'entrenament de la xarxa

# Classificació de les dades de test un cop la xarxa ha estat entrenada
print ( "######## Classificant el DS de test...")
preds = net1.predict(testX)
print ( "######## Fi de la classificació...")
# Fi de la classificació

# Generació del report dels elements de test classificats
# Funció ja definida al paquet sklearn
print (classification_report(testY, preds))
# Fi del report
```

5 - Resultats del prototip

A les primeres proves unitàries realitzades amb el DataSet MIAS el temps de processament era molt elevat degut a la mida de la imatge, per aquest motiu es va prendre la decisió de reduir-les i així també reduir el nombre de neurones d'entrada passant de 1024x1024 neurones a 512x512 neurones. Amb aquesta tasca s'aconsegueix reduir el temps de processament per l'entrenament de la xarxa neuronal ja que es disminueix el nombre de càlculs dels pesos de cada interconnexió.

Imatge 1024x1024 píxels

Imatge 512x512 píxels

Realitzada aquesta tasca de redimensionament, s'analitzen els diferents resultats obtinguts segons els diferents cassos d'estudi ja identificat, cadascun dels quals s'executarà tant amb el model nolearn.dbn com nolearn.lasagne.

5.1 - Identificar la característica del teixit de fons de la mama

El primer cas d'estudi es basa en la columna 2 del DataSet on s'indica el tipus de teixit de fons de la mama, amb les diferents característiques :

- F – Gras (que per l'estudi serà un 2), aproximadament menys del 25% de teixit glandular
- G – Glandulars grassos (que per l'estudi serà un 0), aproximadament entre de un 25% a 75% de teixit glandular.
- D – Glandulars dens (que per l'estudi serà un 1), aproximadament més d'un 75% de teixit glandular.

Per l'anàlisi de les dades s'utilitza:

- Fitxer .cvs : target_imatges_massa.csv
- Carpeta Imatges: \Imatges512

Per aquest cas d'estudi s'utilitzen les imatges senceres de 512x512 píxels ja que l'objectiu és identificar el % de teixit glandular del que esta composat el total de la mama.

La definició de les dades d'entrada de la xarxa neuronal és la següent:

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_massa.csv' # cvs de les dades a carregar
img_src='Imatges512/' # ruta de les imatges a carregar
img_pix = 512 # mida en pixels de les imatges (imatges quadrades)

img_add = 'N' # si tenim disponible imatges addicionals
img_add_csv = '' # csv de les imatges adiccionals
img_add_src ='' # ruta de les imatges addicionals

neu_out= 3 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Resultats amb nolearn.dbn

Com a resultats final de l'execució de diverses configuracions de la xarxa neuronal amb DBN, ens centrarem en les configuracions on els resultats han estat més òptims.

Comencem amb una xarxa de 3 capes, amb una capa oculta de 200 neurones, un learn_rates de 0,05 i 10 iteracions sobre la xarxa.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],200,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

S'observa que els resultats obtinguts no son gaire positius, ja que la xarxa no és capaç predir més d'un 42% dels casos de forma correcta. Observem que del tipus D (1) no ha pogut predir cap cas, amb el que es fa evident que els resultats no son bons.

```
             precision    recall  f1-score   support

          0       0.29      1.00      0.45        30
          1       0.00      0.00      0.00        42
          2       1.00      0.19      0.32        37

avg / total       0.42      0.34      0.23       109

La imatge actual és del tipus 2, i la predicció és 0
La imatge actual és del tipus 1, i la predicció és 0
La imatge actual és del tipus 0, i la predicció és 0
```

Vist els resultats negatius, s'augmenta el nombre de neurones de la capa oculta de 200 a 250 neurones.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],250,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa DBN
```

Amb la nova definició de la xarxa és on s'obtenen els millors resultats. A continuació es pot observar 2 execucions diferents d'aquesta definició. Als primers resultats es van obtenir una precisió total del 79% d'encert, on els casos G(0) i F(2) veiem fins al 100% d'encert.

	precision	recall	f1-score	support
0	1.00	0.03	0.05	38
1	0.30	1.00	0.46	32
2	1.00	0.03	0.05	39
avg / total	0.79	0.31	0.17	109

Als segons resultats el grau d'encert disminueix d'un 79% fins al 50%, amb la conclusió que els resultats no son del tot estables pero si es pot intuir que amb configuracions més potents i amb més possibles valors d'entrenament de la xarxa, es podrien obtenir resultats molt més positius i constats que els actuals.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	32
1	0.31	1.00	0.47	33
2	1.00	0.07	0.13	44
avg / total	0.50	0.33	0.20	109

Resultats amb nolearn.lasagne

A continuació analitzem els resultats obtinguts amb el model nolearn.lasagne. Ens centrarem en la definició de la xarxa amb característiques semblants que nolearn.dbn per a poder comparar resultats.

Es defineix la xarxa neuronal amb 3 capes, una d'entrada amb el nombre de píxels de la imatge, una d'oculta amb 250 neurones i una capa de sortida amb un total de 3 neurones, una per a cada valor possible. La xarxa es configurada amb un total de 10 iteracions i un learning_rate de 0.05.

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
        (InputLayer, {'shape': (None, trainX.shape[1])}),
        (DenseLayer, {'num_units': 250}),
        (DenseLayer, {'num_units': neu_out, 'nonlinearity': softmax}),
    ]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes

# Configuració de la xarxa
print ( "######## Configurant NOLEARN.LASAGNE...")
net1 = NeuralNet(
        layers=layersCNN,
        max_epochs=10,
        update_learning_rate=0.05,
        verbose=1,
    )
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa
```

Els resultats obtinguts son molt semblants que amb el desenvolupament norlearn.dbn, arribant a un % d'encert entorn al 50% dels casos. Resulta interesant saber que l'execució de nolearn.lasagne és molt més lenta que la implementació amb nolearn.dbn, per aquest motiu s'ha tingut certa limitació en al processament de xarxes amb més nombre de neurones.

	precision	recall	f1-score	support
0	0.37	1.00	0.54	38
1	0.00	0.00	0.00	32
2	1.00	0.18	0.30	39
avg / total	0.49	0.41	0.30	109

Conclusions

Amb els resultats del primer cas d'estudi podem observar que els percentatge d'encert varia segons l'execució independentment que es modifiqui la configuració de la xarxa neuronal. Amb això podem dir que els resultats no són constants, poden arribar al 79% d'encert però també arribar a un percentatge molt més baix. Això pot ser degut a l'entrenament es carregui en cada execució, intuint que les dades d'entrenament per aquesta xarxa no són suficients i no són prou robusts.

Segurament amb la disponibilitat de més imatges per l'entrenament, els resultats serien molt més constants i el seu % d'encert augmentaria.

5.2 - Detectar l'aparició d'anomalies en una mamografia

El següent cas d'estudi es basa en la columna 3 del DataSet on s'indica el tipus d'anomalia que es detecta a la imatge. S'identifiquen les característiques:

- NORM – Normal (que per l'estudi serà un 0)
- ANORMAL – La resta (CALC, CIRC, SPI, MISC, ARCH, ASYM, que per l'estudi serà un 1)

Per l'anàlisi de les dades s'utilitza:

- Fitxer .cvs : target_imatges_anomalia.csv
- Carpeta Imatges: \Imatges512

Per aquest cas d'estudi es comença utilitzant les imatges senceres de 512x512 píxels, però tal i com s'explicarà més endavant es realitzarà una tasca de divisió de les imatges per tal de reduir la complexitat del problema i així intentar millorar els resultats obtinguts.

La definició de les dades d'entrada de la xarxa neuronal és la següent:

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_anomalia.csv' # cvs de les dades a carregar
img_src='Imatges512/' # ruta de les imatges a carregar
img_pix = 512 # mida en pixels de les imatges (imatges quadrades)

img_add = 'N' # si tenim disponible imatges addicionals
img_add_csv = '' # csv de les imatges adiccionals
img_add_src ='' # ruta de les imatges addicionals

neu_out= 2 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Resultats amb nolearn.dbn

Es pot veure que amb les diferents proves i configuracions de la xarxa neuronal realitzades els resultats obtinguts no son prou bons, ja que amb les imatges de 512x512 píxels la xarxa no és capaç de classificar de forma correcta més del 45% dels casos.

A continuació es mostren les configuracions i resultats de diferents proves, les quals han estat incapaces de resoldre la complexitat del problema:

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],350,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la conf
# Fi de la configuració de la x
```

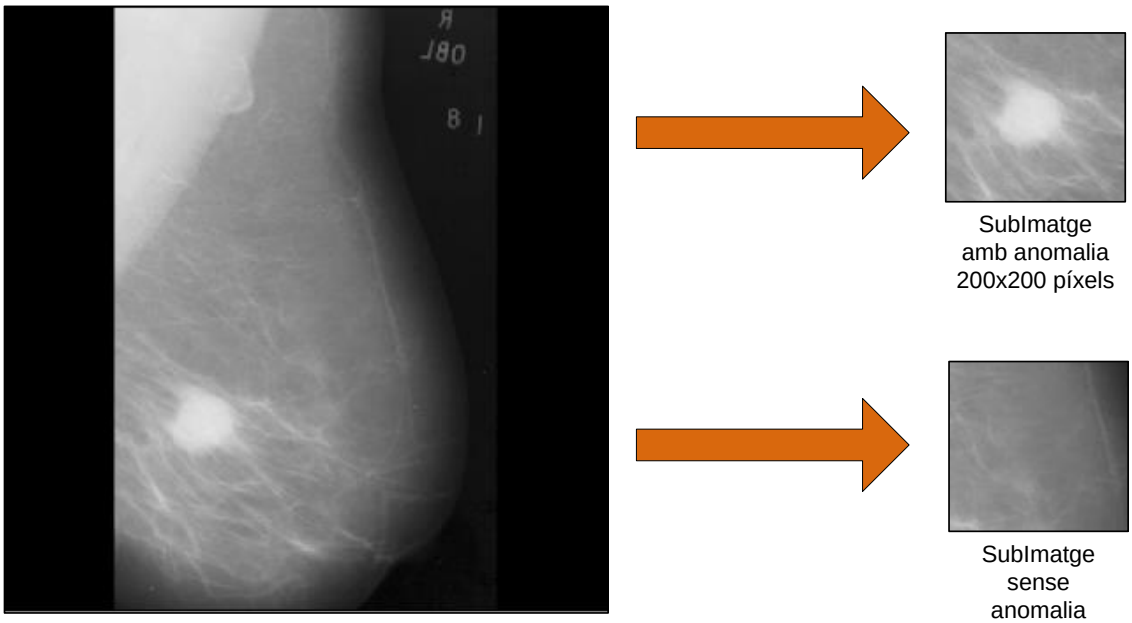
	precision	recall	f1-score	support
0	0.64	0.97	0.77	70
1	0.00	0.00	0.00	39
avg / total	0.41	0.62	0.49	109

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],250,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la confi
# Fi de la configuració de la x
```

	precision	recall	f1-score	support
0	0.67	0.99	0.80	73
1	0.00	0.00	0.00	36
avg / total	0.45	0.66	0.53	109

Obtinguts resultats poc positius per aquest estudi, es realitza una tasca per a reduir la complexitat del problema i intentar millorar el % d'encert. El que seguidament es busca és reduir la mida de les imatges i obtenir més imatges d'entrenament de la xarxa. Per aconseguir això es fa una tasca de divisió de les imatges originals en imatges més petites de parts de la imatge origen, és a dir, per a cada imatges podem obtenir N imatges d'una mida més petita identificant

en cada nova imatge quina d'aquelles conté una anomalia i quina no conté cap tipus d'anomalia, observem l'exemple següent:



Imatge real amb anomalia 512x512 píxels

Amb la tasca de subdivisió de les imatges originals, és realitza la següent definició de les variables d'entrada de la xarxa a entrenar. S'utilitzen les imatges de 200x200 píxels, a més a més de les imatges addicionals que s'han generat des de les imatges originals.

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_anomalia.csv' # cvs de les dades a carregar
img_src='Imatges200/' # ruta de les imatges a carregar
img_pix = 200 # mida en pixels de les imatges (imatges quadrades)

img_add = 'S' # si tenim disponible imatges addicionals
img_add_csv = 'target_imatges_addicionals.csv' # csv de les imatges adiccionals
img_add_src ='Imatges200addicionals/' # ruta de les imatges addicionals

neu_out= 2 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Amb aquestes noves dades d'entrada s'implementa la configuració de 2 xarxes diferents (la primera amb una capa oculta de 450 neurones, i la segona amb una capa oculta de 3000 neurones), observant que el % d'encert millora

considerablement envers les proves anteriors. S'obtenen uns resultats del 69% i 64% respectivament. Es pot veure que el percentatge d'encert en imatges sense anomalia és d'un 77% i les imatges amb anomalia d'un 40% com a màxim.

Gràcies a la tasca de segmentació de les imatges els resultats han millorat considerablement, es pot deduir que si disposéssim de més imatges d'entrenament i s'entrenessin xarxes més complexes els resultats podrien continuar millorant.

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],450,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la confi
# Fi de la configuració de la xa
```

	precision	recall	f1-score	support
0	0.77	0.98	0.86	137
1	0.40	0.05	0.09	41
avg / total	0.69	0.76	0.69	178

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],3000,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la config
# Fi de la configuració de la xar
```

	precision	recall	f1-score	support
0	0.76	0.98	0.85	135
1	0.25	0.02	0.04	43
avg / total	0.64	0.75	0.66	178

Resultats amb nolearn.lasagne

Millorats els resultats gràcies a l'entrenament de la xarxa amb les imatges més petites, s'utilitza el desenvolupament nolearn.lasagne per acabar de validar els resultats obtinguts.

És defineix una xarxa neuronal amb una capa d'entrada, una capa oculta de 450 neurones i la capa de sortida de 2 neurones. Es continua amb un learning_rate de 0,05 i 10 iteracions. Veiem la definició:

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
        (InputLayer, {'shape': (None, trainX.shape[1])}),
        (DenseLayer, {'num_units': 450}),
        (DenseLayer, {'num_units': neu_out, 'nonlinearity': softmax}),
    ]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes

# Configuració de la xarxa
print ( "######## Configurant NOLEARN.LASAGNE...")
net1 = NeuralNet(
        layers=layersCNN,
        max_epochs=10,
        update_learning_rate=0.05,
        verbose=1,
    )
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa
```

Es reafirma la millora de resultats treballant amb imatges més petites i més volum de dades d'entrenament. Amb aquesta configuració s'ha arribat a uns resultats prou òptims (un 83% d'encert total, amb un 100% del tipus 1 i un 77% del tipus 0) per evidenciar que la tècnica Deep Learning pot funcionar per resoldre problemes d'aquest tipus.

	precision	recall	f1-score	support
0	0.77	1.00	0.87	137
1	1.00	0.02	0.05	41
avg / total	0.83	0.78	0.68	178

Conclusions

Amb el segons cas d'estudi es reafirmen les conclusions obtingudes amb el primer cas. Els resultats no són prou constants degut a la falta de dades d'entrenament, però s'ha pogut millorar els resultats amb les imatges de mida més petita i amb el major nombre d'imatges d'entrenament. Amb aquestes

noves dades d'entrenament, els resultats han millorat de forma considerable, però encara no són del tot constants en les diferents execucions realitzades.

Es pot dir que els resultats han estat prou satisfactoris, i confirmar que amb les dades d'entrenament necessàries i els recursos hardware adients és possible l'aprofitament d'aquesta tècnica per a resoldre problemes d'aquesta complexitat.

5.3 - Identificar l'anomalia de masses circumscrites ben definides

El darrer cas d'estudi es basa en la columna 3 del DataSet on s'indica el tipus d'anomalia que es detecta a la imatge, amb l'objectiu d'identificar les anomalies de tipus CIRC ja que són les visualment més fàcil de detectar per representar una estructura ben definida. La classificació es defineix:

- CIRC – Amb anomalia de masses circumscrites ben definides (quer per l'estudi serà un 1).
- No – La resta (CALC, SPIC, MISC, ARCH, ASYM, NORM, que per l'estudi serà un 0)

Per l'anàlisi de les dades s'utilitza:

- Fitxer .cvs : target_imatges_massa_circ.csv
- Carpeta Imatges: \Imatges512

Per aquest estudi es comença utilitzant les imatges senceres de 512x512 píxels, però al igual que en l'estudi anterior, s'utilitzarà les imatges de mida més petita obtingudes d'imatges originals amb l'objectiu de millorar els resultats.

Es defineixen les dades d'entrada de la xarxa neuronal de la forma següent:

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_massa_circ.csv' # cvs de les dades a carregar
img_src='Imatges512/' # ruta de les imatges a carregar
img_pix = 512 # mida en pixels de les imatges (imatges quadrades)

img_add = 'N' # si tenim disponible imatges addicionals
img_add_csv = 'target_imatges_addicionals.csv' # csv de les imatges adiccionals
img_add_src ='Imatges200addicionals/' # ruta de les imatges addicionals

neu_out= 2 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Resultats amb nolearn.dbn

Tal i com s'observa els resultats amb imatges senceres de 512x512 píxels no son prou positius. Encara que podem veure que l'encert és d'un 88%, no és del tot real ja que el volum d'imatges sense anomalia és molt superior, amb la qual cosa si es classifica tot com a 0 la precisió seria molt elevada sense detectar cap imatge sense anomalia. Veiem l'exemple següent:

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],350,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la co
# Fi de la configuració de la
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	102
1	0.00	0.00	0.00	7
avg / total	0.88	0.94	0.90	109

Com s'ha fet amb l'estudi anterior, ens centrarem en les imatges creades de 200x200 píxels, per tal de reduir la complexitat del problema a resoldre.

Seguidament es modifica la definició de les dades d'entrada de la xarxa neuronal, i queda de la següent forma:

```
# Definició de variables d'entrada de dades
print ( "######## Carregant parametres d'entrada...")
cvs_file='target_imatges_massa_circ.csv' # cvs de les dades a carregar
img_src='Imatges200/' # ruta de les imatges a carregar
img_pix = 200 # mida en pixels de les imatges (imatges quadrades)

img_add = 'S' # si tenim disponible imatges addicionals
img_add_csv = 'target_imatges_addicionals.csv' # csv de les imatges adiccionals
img_add_src ='Imatges200addicionals/' # ruta de les imatges addicionals

neu_out= 2 # Neurones de sortida de la xarxa
print ( "######## Fi de la carrega dels parametres d'entrada...")
# Fi definició de les variables d'entrada
```

Observem que a l'igual que al cas d'estudi anterior els resultats han millorant de forma considerable arribant fins a un 93% d'encert global, però el més rellevant és que és capaç de detectar fins al 50% de les imatges amb anomalia CIRC (1).

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],3000,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la
# Fi de la configuració de
```

	precision	recall	f1-score	support
0	0.96	0.98	0.97	170
1	0.20	0.12	0.15	8
avg / total	0.93	0.94	0.93	178

```
# Configuració de la xarxa DBN
print ( "######## Configurant DBN...")
dbn = DBN(
      [trainX.shape[1],4000,neu_out],
      learn_rates = 0.05,
      learn_rate_decays = 0.9,
      epochs = 10,
      verbose = 1)
print ( "######## Fi de la conf
# Fi de la configuració de la x
```

	precision	recall	f1-score	support
0	0.95	0.99	0.97	168
1	0.50	0.10	0.17	10
avg / total	0.92	0.94	0.93	178

Resultats amb nolearn.lasagne

Finalment s'analitzen els resultats amb el desenvolupament nolearn.lasagne, i ens centrarem amb els resultats del joc d'entrenament i de test de les imatges de mida de 200 píxels.

Es configura la xarxa neuronal amb una xarxa oculta de 4000 neurones i un total de 5 iteracions.

```
# Definició de les capes de la xarxa
print ( "######## Definició de les capes...")
layersCNN = [
        (InputLayer, {'shape': (None, trainX.shape[1])}),
        (DenseLayer, {'num_units': 4000}),
        (DenseLayer, {'num_units': neu_out, 'nonlinearity': softmax}),
    ]
print ( "######## Fi de la definició de les capes...")
# Fi de la definició de les capes

# Configuració de la xarxa
print ( "######## Configurant NOLEARN.LASAGNE...")
net1 = NeuralNet(
        layers=layersCNN,
        max_epochs=5,
        update_learning_rate=0.05,
        verbose=1,
    )
print ( "######## Fi de la configuració...")
# Fi de la configuració de la xarxa
```

Amb aquesta configuració s'han obtingut resultats semblants que amb el desenvolupament nolearn.dbn. La precisió total és d'un 92%, però això no és rellevant ja el volum de dades del tipus 0 és molt superior que el de tipus 1. L'important és la capacitat de detectar el 25% dels casos amb anomalia. No és gaire, però amb configuracions més potents i més joc de dades per entrenar la xarxa es pot millorar, com s'ha pogut veure en l'exemple nolearn.dbn on s'ha obtingut fins a un 50% d'encert d'imatges amb massa circumscrites ben definides.

	precision	recall	f1-score	support
0	0.95	0.98	0.97	169
1	0.25	0.11	0.15	9
avg / total	0.92	0.94	0.93	178

Conclusions

Al darrer cas d'estudi s'ha jugat amb la dificultat de tenir poques imatges del tipus de classe 1, encara que a priori és el tipus d'anomalia més fàcil de detectar de forma visual. Això ha fet molt més difícil l'entrenament de la xarxa i l'obtenció de resultats positius. Malgrat això amb les imatges de 200x200 píxels el percentatge de classificació ha estat prou satisfactori com per a reafirmar els conclusions dels casos d'estudi anterior, on s'indica que Deep Learning pot resoldre problemes d'aquesta complexitat si es disposen de les dades necessàries d'entrenament i el hardware adient.

6 - Dedicació i recursos

La realització del Treball Final de Màster ha tingut una durada de tres mesos i la seva dedicació es detallarà al present capítol, conjuntament amb els recursos tant software com hardware necessaris per la seva execució.

Recursos Hardware

- Ordinador Portàtil HP I5 de 64 bits amb 8 GB de RAM i tarja gràfica ATI Radeon.
- Ordinador sobre taula HP Core 2 Quad a 2,6 GHz de 64 bits amb 4 GB de RAM amb tarja gràfica Intel integrada.
- Connexió a Internet.

Recursos Software

- 2 llicencies de Windows 7, incloses en cada ordinador utilitzat.
- 1 versió WinPython 64bits 2.7.9.5
- 1 versió WinPython 64 bits 3.5.1.3
- Llibreries nolearn versió 0.6 i 0.7 amb les seves dependències
- Gimp 2 per la visualització d'imatges PGM i el retoc d'imatges

Recursos Humans i dedicació

- 1 Recurs Humà d'Enginyer Tècnic en Informàtica de gestió sènior.

A continuació es mostra un resum de les hores dedicades:

Id. tasca	Descripció tasca	Hores dedicades
1	Cercar idees sobre possibles treballs a realitzar	6
2	Cercar i estudi d'informació sobre Deep Learning	16
3	Cercar i estudi d'Informació sobre Xarxes Neuronals	9
4	Cercar i estudi d'Informació sobre FrameWorks desenvolupament Deep Learning en Python	18
5	Cercar DataSet per l'execució del projecte: Imatge mèdica	4
6	Preparació de les dades del DataSet per l'estudi	3
7	Preparació dels entorns de treball per la implementació i execució de Deep Learning - Nolearn	30
8	Definició de l'estructura de la memòria	3
9	Desenvolupament de les proves amb Deep Learning: DataSet MNIST	36
10	Anàlisi i jocs de proves sobre dades MNIST	12
11	Tractament sobre les imatges MIAS (redimensionament i fragmentació)	7
12	Desenvolupament i anàlisi del prototip amb Deep Learning: DataSet MIAS	60
13	Anàlisi i jocs de proves sobre dades MIAS	25
14	Redacció de la memòria part teòrica	50
15	Redacció de la memòria part pràctica	35
16	Maquetació de la memòria	4
17	Creació de la presentació	6
18	Creació del vídeo de la presentació	2

Total hores dedicades	326

7 - Conclusions i línies futures

7.1 – Conclusions

Degut a l'evolució tecnològica cada cop es presenten nous reptes al tractament de les dades dels diferents Sistemes d'Informació. Les grans empreses tecnològiques són les pioneres en això, i veiem com Google, Apple o Facebook inverteixen milions de dòlars al camp de la investigació per al tractament de les seves dades.

Als darrers anys, unes d'aquestes branques d'investigació ha estat el que anomenem Deep Learning. Amb Deep Learning el que es pretén és simular el funcionament del cervell humà i és per això que s'han pogut aplicar tècniques en funcions que fins ara eren especifiques del esser humà, com poden ser la visió , el processament del llenguatge o la comprensió de textos.

Conegut el potencial del Deep Learning i els seus bon resultats envers d'altres tècniques que intenten simular els mateixos objectius, apareix la necessitat d'aplicar els seus beneficis en àrees més especifiques com pot ser la medicina.

La imatge mèdica és una d'aquestes àrees com s'ha pogut aprofundir en aquesta memòria, però com resulta obvi no és l'única. Tant sols cal imaginar el potencial d'un Sistema d'Informació Sanitari, on s'apliqués aquest tipus de tècniques no tant sols a les imatges, sinó també a dades analítiques, informació d'antecedents, registres del cursos clínics dels pacients, informes d'alta de les diferents patologies, etc.. El sistema podria ser capaç d'aprendre de la seva pròpia informació i tindria la capacitat dotar als diferents perfils d'usuari d'informació i d'intel·ligència no vista fins ara.

Com diuen els grans experts, el gran repte de futur és el disseny del que s'anomenaria la maquina humana, un sistema amb la capacitat d'aprendre i entendre com ho faria l'esser humà. Amb aquest objectiu en ment, podem imaginar els beneficis que això pot aportar en l'àmbit de la medicina i com pot beneficiar a la societat.

7.2 – Dificultats

En l'execució del projecte han sorgit diverses dificultats de les quals destaquem les següents:

- Incompatibilitat de Llibreries: Degut a la implementació de nolearn.dbn i nolearn.lasagne, han aparegut diverses incompatibilitats sobre les llibreries requerides, que finalment s'ha solucionat utilitzant versions diferents de Python.
- Baix rendiment degut als recursos Hardware: Al no disposar un ordinador amb una bona tarja gràfica, el temps d'entrenament de les xarxes neuronals ha sigut en ocasions molt elevats. Sempre és recomanable utilitzar la GPU envers la CPU ja que permet reduir el temps de processament de forma considerable. Es recomana com a mínim una tarja NVIDIA de mitja categoria.
- Tractament d'imatges: El recursos hardware ha limitat la creació i l'entrenament de les xarxes neuronals. Per tal de reduir la complexitats s'han fet diversos tractament en les imatges. El primer ha sigut redimensionar totes les imatges de 1024x1024 píxels a 512x512 píxels. Encara així, per certs estudis, s'ha realitzat una tasca de segmentació de les imatges, creant imatges de mida de 200x200 a partir de la imatge original, la qual cosa ha millorat els resultats de forma considerable.
- Limitació del nombre d'imatges disponibles: El nombre total d'imatges disponibles no és l'òptim per a resoldre problemes d'aquesta complexitat, per aquest motiu els resultats no han sigut del tot constants en les diferents execucions realitzades. Amb un conjunt d'entrenament més ampli els resultats haguessin millorat.

7.3 - Objectius assolits

Els tres principals objectius marcats des de l'inici del projecte han estat assolits de forma satisfactòria.

S'ha aconseguit aprofundir i estudiar, dins de l'àrea d'intel·ligència artificial, la tècnica Deep Learning desenvolupada amb arquitectura de xarxes neuronals. Les tasques de recerca, comprensió i estudi d'aquesta tècnica m'han fet assolir uns coneixements en aquesta àrea desconeguda per a mi fins fa uns mesos. Gràcies a aquests nous coneixements he pogut reflexionar sobre possibles noves aplicacions d'aquest tipus de tècniques en projectes reals, que podrien millorar de forma considerable tasques diàries arribant a ser totalment automatitzades, o inclús tasques de coneixement podent optimitzar el temps en la seva resolució.

El segon dels objectius plantejats és ser capaços de la creació d'un prototip per el tractament de dades reals com poden ser imatges mèdiques. Després de l'assimilació del conceptes teòrics necessaris, estem en disposició de fer la recerca més pràctica per a fer un desenvolupament amb Python utilitzant una de les llibreries disponibles per implementar Deep Learning. En aquest punt s'ha après la implementació amb dos tècniques diferents, utilitzant DBN i utilitzant LASAGNE, ambdues amb resultats molt semblants.

Finalment, el darrer dels objectius era demostrar que amb l'aplicació d'aquesta tècnica de forma correcta es poden obtenir resultats positius en estudis d'aquest tipus. S'ha pogut demostrar que amb el tractament adequat de les imatges i amb els recursos dels que es disposaven, els resultats obtinguts en cadascun dels 3 cassos d'estudi han estat prou satisfactoris.

8 - Annexos

S'annexa amb la memòria les carpetes següents amb el contingut que s'explica:

DL-MNIST-DBN : Carpeta amb els fitxers necessaris per l'execució en Python 2 de les proves amb MNIST amb nolearn.dbn, el seu contingut és:

- DeepLearning-DBN-MNIST.py : Codi Python.
- cv2.pyd: Llibreria Python cv2.

DL-MNIST-LASAGNE: Carpeta amb els fitxers necessaris per l'execució en Python 3 de les proves amb MNIST amb nolearn.lasagne, el seu contingut és:

- DeepLearning-LASAGNE-MNIST.py: Codi Python.

DL-MIAS-DBN: Carpeta amb els fitxers necessaris per l'execució en Python 2 del desenvolupament de MIAS amb nolearn.dbn, el seu contingut és:

- Imatges512: Imatges per l'estudi amb mida 512x512 píxels.
- Imatges200: Extracció d'imatges de mida de 200x200 píxels.
- Imatges200addiconals: Extracció d'imatges addicionals en mida 200x200 píxels.
- target_imatges_massa.csv: Fitxer csv amb la classificació de les imatges per tipus de massa.
- target_imatges_anomalia.csv: Fitxer csv amb la classificació de les imatges per tipus d'anomalia.
- target_imatges_massa_circ.csv: Fitxer csv amb la classificació de les imatges per anomalia de masses circumscrites ben definides.
- target_imatges_addicionals.csv: Fitxer csv per la carrega d'imatges addicionals.
- DadesImatges.txt: Fitxer txt amb la informació de totes les imatges

- DeepLearning-DBN-MIAS.py: Codi Python.
- cv2.pyd: Llibreria Python cv2.

DL-MIAS-LASAGNE: Carpeta amb els fitxers necessaris per l'execució en Python 3 del desenvolupament de MIAS amb nolearn.lasagne, el seu contingut és:

- Imatges512: Imatges per l'estudi amb mida 512x512 píxels.
- Imatges200: Extracció d'imatges de mida de 200x200 píxels.
- Imatges200addiconals: Extracció d'imatges addicionals en mida 200x200 píxels.
- target_imatges_massa.csv: Fitxer csv amb la classificació de les imatges per tipus de massa.
- target_imatges_anomalia.csv: Fitxer csv amb la classificació de les imatges per tipus d'anomalia.
- target_imatges_massa_circ.csv: Fitxer csv amb la classificació de les imatges per anomalia de masses circumscrites ben definides.
- target_imatges_addicionals.csv: Fitxer csv per la carrega d'imatges addicionals.
- DadesImatges.txt: Fitxer txt amb la informació de totes les imatges.
- DeepLearning-LASAGNE-MIAS.py: Codi Python.

9 - Fonts de dades i bibliografia

[1] https://pythonhosted.org/nolearn/

[2] http://deeplearning.net/software/theano/

[3] http://lasagne.readthedocs.io/en/latest/user/tutorial.html

[4] https://www.tensorflow.org/

[5] http://www.jorditorres.org/libro-hello-world-en-tensorflow/

[6] http://deeplearning.net/software/pylearn2/

[7] http://caffe.berkeleyvision.org/

[8] http://www.pyimagesearch.com/2014/09/22/getting-started-deep-learning-python/

[9] http://www.teglor.com/b/deep-learning-libraries-language-cm569/

[10] http://www.aidanf.net/posts/a-list-of-python-software-for-deep-learning

[11] http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python/

[12] https://www.cbinsights.com/blog/python-tools-machine-learning/

[13] http://es.slideshare.net/roelofp/python-for-image-understanding-deep-learning-with-convolutional-neural-nets

[14] http://nbviewer.jupyter.org/github/dnouri/nolearn/blob/master/docs/notebooks/CNN_tutorial.ipynb

[15] http://deeplearning.net/

[16] http://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial

[17] https://es.wikipedia.org/wiki/Aprendizaje_profundo

[18] http://www.eldiario.es/hojaderouter/tecnologia/software/moda-deep_learning-algoritmo-inteligencia_artificial_0_275772610.html

[19] https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/

[20] http://mediatelecom.com.mx/~mediacom/index.php/tecnologia/usos-sociales/item/75625-enlitic-deep-learning-para-la-detecci%C3%B3n-de-enfermedades-a-trav%C3%A9s-de-im%C3%A1genes

[21] https://es.wikipedia.org/wiki/Red_neuronal_artificial

[22] http://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

[23] http://neuralnetworksanddeeplearning.com/chap1.html

[24] http://peipa.essex.ac.uk/info/mias.html

[25] https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass

[26] http://netpbm.sourceforge.net/doc/pgm.html

[27] https://en.wikipedia.org/wiki/Netpbm_format