Mètodes d'eficiència aplicats a les xarxes neuronals

David Bosch Sepulveda

Resum– L'objectiu d'aquest projecte es basa en estudiar l'aplicació de mètodes d'eficiència sobre les xarxes neurals, els quals busquen rebaixar els requeriments computacionals necessaris per a processar una xarxa. Concretament s'apliquen dos mètodes que abarquen diferents aspectes sobre la xarxa, el pruning o poda de connexions i la quantització dels paràmetres.
Amb l'aplicació d'aquests mètodes s'aconsegueix obtenir una reducció del tamany de la xarxa, per tal de poder processar xarxes neuronals sobre dispositius amb pocs recursos computacionals i energètics.

Paraules clau– Xarxes neuronals, Deep Learning, Fine-tuning, Poda, Quantització, Cost en Memòria, Cost Computacional

Abstract– The aim of the study relies on the research of the of known efficiency methods applied on a neural network that reduces the computational requirements for processing a network. Specifically, the pruning of connections and the quantization of the parameters. Those methods cover different aspects of a neural network. With the applications of the previous methods, it is expected to obtain a reduction of the network size, so the low-resources and low-power devices can process it.

Keywords– Neural network, Deep learning, Fine-tuning, Pruning, Quantization, On-memory cost, Computational cost

1 Introducció

El tema de xarxes neuronals cada dia està més present en tots els àmbits relacionats amb la computació, i avança a gran velocitat. Existeixen múltiples aplicacions d'aquesta tecnologia en problemes complexes. Un d'ells es la conducció autònoma, que permet diferenciar els elements captats a través d'una càmera, que un conductor es podria trobar, i actuar en concordància. Les xarxes neuronals també estàn presents en el camp de la salut, on darrerament s'ha conseguit avançar molt. Un cas concret té a veure amb la classificació de teixit cerebral segons si pateix alguna lesió o és saludable mitjançant fotografíes en 3 dimensions.
S'han esmentat alguns dels camps d'aplicació de les xarxes neuronals però aquesta tecnologia abarca aspectes tan diferents com l'anàlisi de comportament de persones, reconstrucció d'imatges, descripció d'imatges... Davant la gran utilitat que ofereixen les xarxes neuronals sobre aspectes tant diferents, surgeix la necessitat de poder aplicar-les de forma còmode i ràpida en dispositius de baixa capacitat computacional, com seria un dispositiu móvil, per tal de donar una resposta a un event desitjat de forma ràpida i eficient. La gran desavantatge de les xarxes neuronals actualment és la necessitat de màquines amb un nivell d'exigència computacional elevada, per tal de poder-les fer funcionar, i, és un tema que limita l'expansió i l'integració d'aquesta tecnologia dintre de la societat. Existeixen una sèrie de mètodes que busquen trobar solució a aquest problema, per permetre l'ús de les xarxes neuronals en dispositius casuals. Aquest treball explora l'ús de mètodes d'eficiència sobre les xarxes neuronals per poder donar sortida a aquesta tecnologia cap a dispositius amb capacitat computacional restrictiva.

2 Objectius

La necessitat doncs d'una solució sobre el problema que presenten el tamany i el cost computacional d'algunes xarxes neuronals, concretament sobre les *'deep neural networks'*, surgeix quan es vol executar una inferència (fase en la qual un cop entrenada la xarxa, es dòna una predicció en resposta a una certa entrada) sobre un model en un temps acotat en una màquina amb capacitat de càlcul i memòria limitades. La solució doncs a aquest problema pasa per l'aplicació de mètodes d'eficiència que simplifiquen les xarxes

• E-mail de contacte: daaviid1712@gmail.com
• Menció realitzada: Enginyeria de Computació
• Treball tutoritzat per: Joan Serrat Gual (Computer Science)
• Curs 2017/18

neuronals reduïnt el tamany en memòria aixi com el cost de càlcul a l'hora de fer la predicció. Els dos mètodes elegits són:

- Pruning
- Quantization (Ternary trained quantization)

L'elecció d'aquests mètodes es deguda a que el primer mètode *'prunning'* permet la poda de connexions innecesaries entre capes de una forma ràpida i amb bons resultats, disminuïnt en gran mesura el pes de la xarxa degut a la eliminació de paràmetres. D'altra banda el segon mètode *'ternary trained quantization'*, permet la quantització dels paràmetres de les capes elegides reduïnt aixi el nombre de bits necessaris per a l'expressió dels pesos, amb la conseqüència final d'un tamany en memòria reduït considerablement. L'objectiu principal d'aquest projecte doncs es l'aplicació d'aquests dos mètodes d'eficiència sobre una model. A continuació, s'expliquen amb més detall els objectius en els que el treball s'enfoca.

- Fer un fine-tuning de la xarxa per a detectar 40 senyals de trànsit i conseguir una precisió raonable per poder fer l'estudi previ
- Aplicar el prunning en les capes fully-connected del model per observar aspectes com la precisió i el nombre de paràmetres.
- Aplicar la quantització sobre les capes convolucionals del model per observar aspectes com la precisió i el nombre de paràmetres.

Durant aquest document es descriurà l'estat d'art d'aquestes tècniques citant projectes i articles, per després pasar a la fase principal que sería tot el process del treball realitzat. Aquesta fase principal primerament s'enfoca en la preparació de la xarxa neuronal personalitzada a estudiar, que es fa a partir duna VGG16, per a un cop desenvolupada aquesta part, passar a l'estudi dels diferents mètodes d'eficiència aplicats sobre la xarxa.

3 ESTAT DE L'ART

Normalment totes les xarxes neuronals están sobreparametritzades, és a dir, contenen paràmetres que no col·laboren alhora de fer la inferència sobre una entrada, i que l'únic que consegueixen és augmentar el tamany d'aquesta amb el consegüent augment del cost computacional, de memòria i energètic. Així doncs es pot obtenir un model eficienment superior amb una precisió igual al model original. Generalment quan es parla sobre augmentar l'eficiència de les xarxes neuronals, es tenen dos camins:

- Reduir el nombre de paràmetres
- Tenir menys precisió sobre els paràmetres (utilitzar menys bits per a expressar-los)

3.1 Reducció del nombre de paràmetres

Un dels primers mètodes en abordar aquesta teoria és *'Optimal Brain Damage'* [8], que consisteix bàsicament, en adaptar el tamany de la xarxa neuronal i utilitzar la informació de segona derivada per a obtenir un model amb una complexitat i un error d'entrenament raonables. També hi han treballs que es basen en la descomposició de valors singulars per poder reduir el nombre de paràmetres de la xarxa neural [9]. Existeixen també arquitectures innovadores que permeten la reducció de paràmetres sustituint les FC (*'fully connected layers'*) que és on es concentren la majoria de paràmetres, per capes de convolucionals que realitzen la mateixa tasca de forma més eficient. També, el mètode desenvolupat per S. Han i coautors [4], que es on descansa la part de *prunning* d'aquest projecte, es basa en la poda dels pesos més petits conseguint també la reducció de paràmetres amb la no pèrdua de precisió del model.

3.2 Reducció de la precisió dels paràmetres

Pel que fa al mètode de reducció de precisió dels paràmetres, existeixen mètodes que utilitzen 8 bits per representar els pesos (vs 32 bits) [10]. També un mètode que consegueix reduir el nombre de bits per representar el pes agrupant connexions de manera aleatoria mijtançant una funció de hash [11]. El mètode conegut com a *'binay neural network'* [12] que consegueix reduir de 32 bits a 2-3 bits mitjançant la distribució de Bernouilli, obtenint així una reducció en el temps de la inferència sobre el model quantitzat. El mètode que s'ha seguit durant aquest projecte *'Trained ternary quantization'* [13], que permet expressar els pesos amb 3 valors (3 bits), els quals asigna la xarxa durant l'entrenament. Amb tots els mètodes esmentats anteriorment es pot reduir la precisió del pes (decimals del nombre) i cada pes pot ser representat amb un menor nombre de bits.

3.3 Deep Compression

Actualment es parla de *'Deep compression'* [14] com l'aplicació dels dos tipus de mètodes descrits anteriorment a les xarxes neuronals, conseguint així els beneficis de tenir un nombre reduït de paràmetres i poder-los expressar amb un menor nombre de bits, cosa que comporta una major reducció computacional i energètica. Facebook per exemple, ha fet servir aquesta tècnica que combina els dos mètodes, per implementar les DNNs (deep neural networks) sobre mòvils [15]. Baidu també ha adoptat aquest mecanisme per utilitzar models de DNN sobre aplicacions de móvil, com per exemple reconeixement facial.

4 METODOLOGIA

Per al desenvolupament del projecte, s'ha optat per fer ús de python. De entre totes les llibreries dedicades a les xarxes neuronals, s'ha triat la llibreria pytorch [1], perque cada cop està creixent més i permet el desenvolupament de projectes interessants. A més, compte amb el suport actiu dels desenvolupadors que permet la constant aportació de noves funcionalitats. Un altre de les raons per la qual s'ha utilitzat aquesta llibreria, és que no existeixen molts projectes desenvolupats amb aquesta eina referents a l'eficiència sobre xarxes neuronals i suponia un petit repte que s'ha volgut adoptar.
La xarxa neuronal escollida doncs per al estudi dels

mètodes d'eficiència de prunning i quantization, ha estat la VGG16 (1). És una xarxa convolucional que compte amb un nombre de capes i paràmetres adequats per a poder aplicar els estudis mencionats, a més de ser compatible amb el fine-tuning que permet que la xarxa detecti les senyals de trànsit.
S'han utilitzat projectes de suport de codi obert per al desenvolupament d'aquest treball sobre el prunning [6] i sobre la quantització ternària [16]

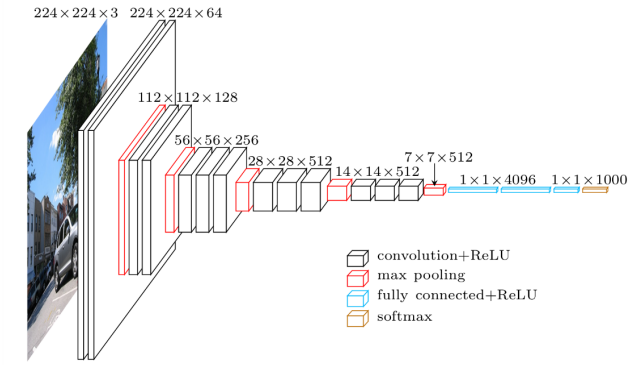

Fig. 1: Capes del model VGG16

4.1 Fine-tuning de la xarxa

Poques vegades realment s'entrena una xarxa neuronal convolucional com podria ser la VGG16, desde zero, és a dir, amb uns pesos inicials, ja que es bastant inusual que es tingui el volum de dades suficient per a poder obtenir uns bons resultats.
En canvi, és cada cop més comú utilitzar una xarxa neuronal (en el nostre cas convolucional) que ha estat previament entrenada sobre un volum de dades molt gran (Imagenet per exemple conté més de 1.2 millons d'imatges amb 1000 categories) com a inicialització. El que entenem doncs per *'fine-tuning'* o *'task transfer'* és l'utilització dels pesos d'una xarxa preentrada com a base per a desenvolupar la nostra tasca d'interés (2).
Existeixen dos grans escenaris que descriuen com efectuar aquesta transferència. La primera consisteix en inicialitzar la nostra xarxa amb la xarxa preentrenada com a base per a després efectuar el entrenament de forma habitual. La segona consisteix en fixar tots els pesos de les capes convolucionals, de manera que durant l'entrenament de la xarxa aquests pesos no patiran modificacions, a més de substituir les capes de classificació (les capes FC) per el conjunt desitjat. En aquest cas, l'única part de la xarxa que s'entrena és la part de classificació, obtenint així un temps molt més reduït d'entrenament amb un bons resultats.

Per aconseguir el fine-tuning de la xarxa en el nostre cas, primerament s'ha carregat la VGG16 a través de la llibreria *torchvision* que permet carregar els pesos preentrenats. En segon lloc, s'han dividit les imatges en un 70% d'entrenament, un 20% de validació, i la resta de test. Per a facilitar l'ingesta de les dades a la xarxa, s'ha utilitzat la mateixa llibreria *torchvision*, que permet d'una manera senzilla la gestió de l'ingesta de dades i tots els processaments necessaris, com per exemple normalitzar les dades, i escalar-les a un tamany adequat per a la xarxa.

Pel que fa a l'estructura de la pròpia xarxa, s'han substituit les 3 FC originals de la VGG16 per a dos FC ([2048,1024], [1024,40]) respectivament, aconseguint doncs un classificador capaç de discernir sobre 40 categories diferents, en el nostre cas les 40 senyals de trànsit diferents. Un altra aspecte important a mencionar, és que s'ha optat per congelar els pesos de totes les capes convolucionals (capes de extracció de característiques) de tal manera que només s'han hagut d'entrenar els paràmetres de les noves capes afegides (capes de classificació). Finalment amb l'estructura montada, s'ha passat a entrenar la xarxa durant 15 èpoques, vist que augmentant-ne el nombre no implicaba una millora en la precisió de la xarxa. Així doncs un cop entrenada, amb el set de test s'ha obtingut una precisió entorn al **85%**.

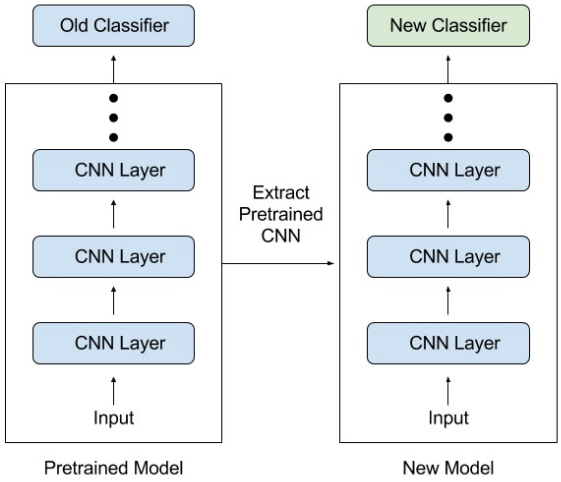

Fig. 2: Idea del Fine-tuning

4.2 Prunning

La poda de paràmetres, més coneguda com a prunning, és un mètode que es basa en això, en reduir el nombre de pesos d'una xarxa neuronal, més específicament de les seves capes, mitjançant algun indicador que informa sobre quins pesos són els redundants i que, per tant, no aporten cap benefici alhora d'aplicar l'inferència.

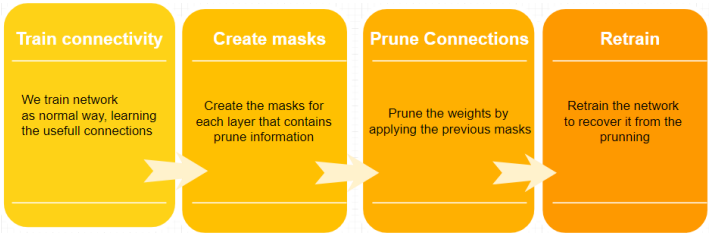

Fig. 3: Mètode general de pruning

Existeixen varies formes d'efectuar la poda com s'ha explicat en el punt de l'estat de l'art, però en aquest treball s'ha adoptat la metodologia proposada per Song Han en el seu article [4], que permet la poda de paràmetres sense afectar a la precisió original de la xarxa, sent possible també una millora en el model degut a la generalització.

En aquest article, Song Han proposa un mètode basat en 3 pasos (3). Aquest mètode comença per el entrenament natural de la xarxa, o en el nostre cas, en fer servir la xarxa preentrenada anteriorment quan s'efectuava la transferència d'aprenentatge, de tal manera que busquem aprendre quins són els pesos que aporten un major benefici alhora de fer la predicció, és a dir, localitzem aquells pesos que són més importants. En el training habitual, en comptes de buscar quins pesos aporten més durant la predicció, es busca quins pesos són els que funcionen bé de forma general, però de forma natural la xarxa apren els dos aspectes durant l'entrenament, per tant doncs s'entrena la xarxa de forma habitual. El segon pas té a veure en crear algún tipus d'indicador que ens informi sobre quins pesos són els importants i quins poden ser podats. En l'article, s'opta per un indicador molt senzill però eficaç, que penalitza aquells pesos amb valors més petits, és a dir, que a la pràctica els pesos per sota d'un llindar de valor mínim que imposem, són podats. Per posar un exemple pràctic, podriem podar el 60% calculant el llindar mínim adequat per aquest percentatge, i així amb qualsevol altre.
Un aspecte molt important d'aquest projecte és que entenem la poda del pes com a reduir el pes a podar a zero, que sería el mètode més equivalent a eliminar la connexió.
En aquest punt doncs tenim una xarxa amb un percentatge de pesos podats (4), a la que si efectuem una revisió de la precisió, observarem que ha disminuït. Això és degut a que falta l'últim pas del mètode que és crític, que consisteix en reentrenar aquells pesos que han quedat (és a dir els pesos que no han estat podats) per a que puguin adaptar-se al nou entorn degut a la poda.

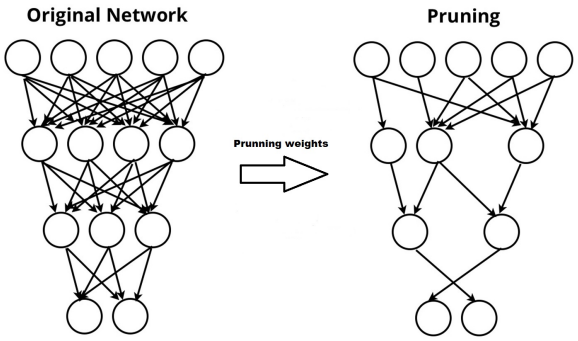

Fig. 4: Descripció gràfica de la poda de paràmetres

La idea de efectuar el prunning és a la pràctica la igualació a zero a nivell de pes per al paràmetre a podar i conseguir evitar que alhora de fer el backpropagation (que és el procés que conté la part d'actualització dels pesos que és la que ens interesa) el pes indicat com a podat no s'actualitzi. En aquest projecte es farà el prunning a la part de les capes de FC, degut a que són les capes que contenen la major densitat de paràmetres.
La part doncs, que condiciona més el procés de prunning, té a veure amb la propia estructura de la xarxa, ja que ha de ser capaç de admetre tot aquest procés esmentat anteriorment.
Per a entendre correctament tot aquest mètode, és important conèixer com tracta les variables i funcions la llibreria pytorch. De forma resumida, pytorch fa ús de grafs computacionals dinàmics necessaris per a poder soportar tots els càlculs i procediments que una xarxa neuronal pot necessitar. Un graf computacional és un graf acíclic on els nodes representen les variables (tensors, variables) i les connexions representen les operacions (sumes, multiplicacions...). És a dir, cada cop que declarem una variable o efectuem una operació amb ella, guardem tota aquesta informació en el graf. Llavors, durant la optimització, combinant la regla de la cadena i el graf de computació, obtenim la derivada de la sortida, respecte les variables (nodes) que es volen aprendre (en les quals el flag de gradient està activat), per a finalment poder actualitzar aquestes variables per acostar-nos més a la solució. Pytorch permet especificar si s'ha d'efectuar el gradient a nivell de variable. A la pràctica, les variables són el conjunt de pesos per a cada capa de la xarxa neuronal que s'han d'entrenar per a aconseguir l'efecte desitjat. En el nostre cas, no podem fer ús d'aquest flag que permet controlar si s'efectua el gradient de les variables, ja que aquest afecta a nivell de conjunt de pesos de cada capa i no de forma individual per cada pes. Degut a això, hem de fer ús d'una màscara que conté informació sobre quins pesos s'han de podar per tal de reduir a zero a nivell de paràmetre de cada capa FC.

4.3 Quantització ternària

El concepte de quantització és conegut com el processament i mapeig de diferents entrades d'una espai molt gran (normalment un espai continu), cap a valors de sortida d'un espai molt més petit (valors discrets). Truncar valors o arrodonir-los són exemples típics de quantització. En el cas de les xarxes neuronals, concretament les deep learning, l'espai de valors que els diferents paràmetres poden adoptar és molt gran (normalment 32 bits per representar els pesos), i comporta una exigència de recursos tant de memòria, computacional. La quantització doncs busca resoldre aquest problema consideran un espai molt més reduït pels diferents valors que els paràmetres poden prendre.
De forma experimental s'ha observat que en els diferents mètodes que existeixen, alguns descrits anteriorment, presenten una degradació de la precisió del model degut a la compressió dels pesos, on en casos específics d'alguns models, aquesta degradació és més gran. El mètode proposat per S.Han i altres coautors [13], està enfocat en intentar solucionar la contradicció que apareix entre la compressió dels pesos i la pèrdua de precisió del model. Aquest mètode és l'adoptat per aquest treball i bàsicament, aplica una transformació als paràmetres de la xarxa neuronal per a poder expressar-los amb 3 valors, els quals són entrenats per la xarxa, obtenint així una millora en la precisió del model respecte altres mètodes.

L'article que segueix aquest treball, proposa un mètode basat en una sèrie de pasos aplicats sobre una xarxa neuronal per aprendre els valors de quantització i a la vegada obtenir una precisió del model acceptable (5). En el nostre cas, tots els passos de la quantització ternària, s'apliquen sobre la xarxa prèviament preentrenada a la que se li ha aplicat un fine-tuning.

Com a primer pas en la quantització ternària, s'aplica una

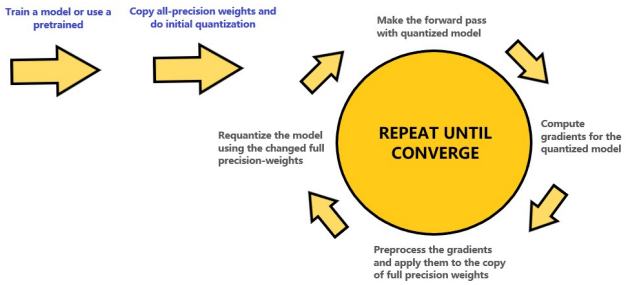

Fig. 5: Pasos de la quantització ternària

normalització dels pesos de la xarxa dividint cada pes pel pes amb valor màxim que conté la xarxa obtenint així un rang de valors per cada pes entre [-1,+1]. El següent pas de la quantització s'encarrega de preparar l'estructura que soporta la quantització dels pesos. Per a això, es creen 3 variables diferents:

- Variable amb els pesos que utilitza la xarxa per a fer la predicció. És aqui on s'asignaran els valors dels pesos quantitzats, un cop calculats.
- Variable que conté una còpia dels pesos amb total precisió (pesos abans d'aplicar quantització)
- Variable que conté els factors d'escala, és a dir, aquells valors que prenen els pesos de la xarxa quan s'efectua la quantització.

Durant la primera iteració del mètode, els factors d'escala de la quantització prenen els valors $[w_l^p = 1, 0, w_l^p = -1]$, procés el qual es pot entendre com a la inicialització dels valors de quantització o factors d'escala. L'assignació dels factos d'escala adequats per a cada pes es basa en un llindar, descrit en la fòrmula (1).

$$w_t^l = \begin{cases} W_l^p : w_l' > \triangle_l \\ 0 : |w_l'| \leq \triangle_l \\ -W_l^n : w_l' < -\triangle_l \end{cases} \quad (1)$$

$\triangle_l$, és l'indicador llindar utilitzat per a assignar el valor a cada pes específicat en la fòrmula (1), on el hyperparàmetre *t* és global per totes les capes de la xarxa per tal de reduir l'espai de cerca. Aquesta fòrmula es defineix com:

$$\triangle_l = t \times \max(|w'|)$$

Un cop s'han obtingut els valors de pesos quantitzats sobre el model, s'efectua la inferència o predicció del model sobre unes entrades. En aquest pas doncs la precisió del model haurà decaigut enormement degut a que els valors quantitzats són els de inicialització i no han passat encara per l'entrenament. A continuació, ve la part més important d'aquest mètode, que busca l'actualització dels pesos quantitzats per a aprendre aquells factors d'escala (w_l^p, w_l^n) que donen una precisió al model acceptable. Durant aquest procés, s'utilitzen la còpia dels pesos amb total precisió (els pesos anteriors a la quantització), i els factors d'escala actuals utilitzats per a la quantització, per a efectuar els gradients d'ambdós, mitjançant de nou la fòrmula (1). Un cop obtinguts aquests gradients, s'actualitzen els pesos amb total precisió i els factors d'escala amb els gradients prèviament calculats. Notar que pels pesos quantitzats anteriorment que han servit per a fer el forward o predicció, no se'ls hi efectua el gradient ni l'actualització, ja que aquests pesos quantitzats només s'utilitzen alhora de predir i la seva actualització no ve donada pel seu gradient, sinó pels gradients o actualitzacions dels factors d'escala i els pesos de precisió total.
Finalment, un cop efectuada l'actualització de les variables pertinents amb els gradients, es torna a calcular la quantització amb (1), és a dir, calculant si cada pes (del conjunt de pesos amb total precisió després de ser actualitzats), supera o no el llindar específicat pels nous factors d'escala. A partir d'aquest punt es torna a calcular la predicció del model mitjançant els nous pesos quantitzats i, de forma iterativa, es repeteix la metodologia descrita anteriorment fins que el model convergeixi i doni una precisió raonable. Un cop el model convergeix, s'eliminen els pesos amb total precisió i es fixen els pesos quantitzats a la xarxa, que serviràn per a futures prediccions del model.

5 DATASET

Per a desenvolupar aquest treball, s'ha escollit un dataset oferit per l'institut Tsinghua, que consta amb més de 100 classes de senyals de trànsit [1]. S'ha elegit aquest dataset ja que per a un problema típic de classificació consta amb prou mostres i classes diferents.
Inicialment aquest conjunt de dataset està format per imatges senceres juntament amb un arxiu json que descriu les coordenades i el tipus de cada senyal. Així doncs, previament s'ha hagut de fer un preprocessat de cada imatge sencera per obtenir cada senyal. Un cop filtrat, s'ha obtingut una carpeta que conté totes les classes possibles de senyals i dins de cada classe el conjunt de imatges retallades a la que pertany cada senyal de trànsit.

D'altra banda aquest dataset conta amb alguns problemes que s'han hagut de solucionar mitjançant la creació d'una classe. Aquesta classe doncs, a part de solucionar els problemes que presenta el dataset, es fa servir també per crear el dataset final que la xarxa neuronal fa servir per entrenar. Alguns dels problemes més importants que presenta aquest dataset són:

- Existeixen classes sense cap imatge que s'han hagut d'obviar.
- Algunes classes contenen imatges etiquetades incorrectament. En aquest cas s'han recolocat manualment les imatges mal etiquetades.
- Algunes classes contenen imatges distorsionades d'altres classes, per tant tampoc s'han tingut en compte.

Degut a que aquest dataset no està balancejat (6) (diferència de nombre d'imatges per classe), s'ha optat per elegir les classes amb més imatges, per a poder conseguir el millor resultat possible. En el nostre cas s'han elegit les 40 senyals de trànsit amb més densitat d'imatges. Aquesta funcionalitat i d'altres com aconseguir separar el parell classes-imatges en sets per el entrenament, la validació i el testeig, estàn implementades dins d'aquesta classe.

TAULA 1: IMATGES DE LES DIFERENTS CLASSES

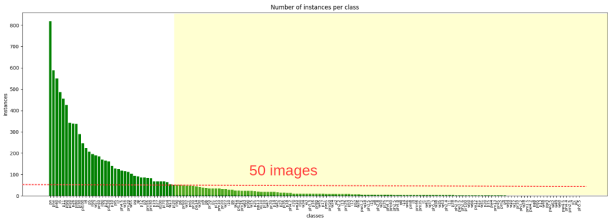

Fig. 6: Nombre de instancies per classe

6 IMPLEMENTACIÓ

6.1 Mètode de Prunning

Com s'ha explicat anteriorment, la poda de paràmetres s'efectua sobre les capes FC, i degut a que no existeix un flag per controlar el gradient a nivell de pes, s'ha creat un nou tipus de capa FC que permet el control a nivell de paràmetre. La classe hereda de la pròpia classe *LinearLayer*, i s'ha creat una nova funció que permet aplicar una màscara als pesos contingut dins de la capa. D'aquest manera mitjançant la màscara podem elegir quins pesos es volen podar colocantlos a 0.

Un altre punt important a tenir en compte de la poda és l'actualització dels pesos durant el backpropagation. Degut a que realment amb l'aplicació de la màscara no conseguim desactivar cap gradient a nivell individual (es segueix efectuant el gradient dels pesos ja que formen part d'una variable dins del graf amb el gradient actiu) sinó que simplemente posem a zero els pesos que ens interesa podar, durant el backpropagation tots els pesos s'actualitzen i per tant deixen de valdre zero. Per a resoldre aquest problema, s'utilitza un hook durant el backward [7]. Un hook sobre el backward o backpropagation, és una funció que es crida justament quan s'efectua aquest procediment, i en el nostre cas utilitzem aquesta funció (una per cada capa FC) per reduir a zero el gradient dels pesos podats, multiplicant el gradient de cada pes per la mateixa màscara que conté la informació sobre la poda. D'aquesta forma quan arriba l'hora d'actualitzar tots els pesos, aquells pesos elegits com a podats sel's hi suma zero com a gradient aconseguint així mantenir la poda inicial.

Per calcular la màscara, donat un percentatge de poda desitjat, el que es fa és calcular el límit adequat de entre el total dels pesos per a podar els que estiguin per sota d'aquest límit, és a dir, podar el percentatge dels pesos més petits. Degut a que l'estructura de la xarxa ha cambiat (les capes de FC han cambiat), es reentrena tot de nou fent el fine-tuning de la mateixa forma que s'ha explicat en un dels punts anteriors. Un cop hem obtingut la xarxa entrenada amb una precisió raonable, calculem la màscara per a cada capa FC. Un cop obtinguda aquesta màscara s'aplica sobre la xarxa inicial obtenint així els pesos podats i mantenint aquells que no ho han d'estar. Seguidament es torna a entrenar la xarxa per poder recuperar el coneixement perdut durant la poda reentrenant els pesos que no han estat podats, tenint en compte que durant el backpropagation, s'aplica la funció que permet controlar l'actualització dels pesos per a poder mantenir a zero el gradient d'aquells pesos que han sigut podats.

6.2 Mètode de Quantització ternària

Durant la quantització ternària s'utilitza el model modificat VGG16 que s'ha utilitzat al prunning, però qualsevol model habitual serveix com a estructura per aquest mètode. Primer de tot, es prepara el model de tal manera que es passen els pesos a quantitzar i aquells que seràn entrenats de forma habitual (sense patir quantització) al optimitzador, que s'ocuparà de l'actualització dels pesos. En el nostre cas els pesos a quantitzar són el de totes les capes convolucionals menys la primera, degut a que és la primera capa que entra en contacte amb les dades d'entrada del model, de tal manera que facilitem l'aprenentatge dels factors d'escala. Aquells pesos que no es quantitzaràn són els de les capes FC. Després, s'efectua una normalització de tots els pesos de la xarxa que han de ser quantitzats.

Un cop tenim tenim la normalització dels pesos, efectuem una còpia dels pesos a quantitzar continguts dins de l'optimitzador (en aquest punt aquests pesos són els de total precisió) i els agrupen dins d'un nou optimitzador. Seguidament efectuem la quantització sobre els pesos a quantitzar de l'optimitzador general, utilitzant els factors d'escala inicials [-1, 0, +1] i la còpia dels pesos amb total precisió per asignar a cada pes el seu valor quantitzat adequat segons la fòrmula (1). Per acabar aquest pas, agrupem els factors d'escala inicials en un nou optimitzador.

Un cop hem obtingut l'inicialització de quantització, entrem en la fase d'entrenament on s'aprendràn els factors d'escala adequats pel model. Durant aquesta fase es calcularà la pèrdua del model (loss) mitjançant els pesos quantitzats i es fara ús de tres optimitzadors:

- L'optimitzador principal que conté tots els pesos, tant els que es volen quantitzar com els que no.

- L'optimitzador que conté una còpia dels pesos a quantitzar (els pesos amb total precisió).
- L'optimitzador que conté els factors d'escala.

Llavors per a cada kernel o capa convolucional es calculen els gradients dels factors d'escala i de la còpia dels pesos amb total precisió. A continuació s'actualitzen els pesos que no es volen quantitzar (agrupats en l'optimitzador general), i els dels factors d'escala i la còpia dels pesos amb total precisió (aquests dos últims mitjaçant el gradient previament calculat). Notar que pels pesos quantitzats no s'efectua cap gradient ni actualització. Per últim es tornen a calcular els valors dels pesos quantitzats mitjançant (1).
Finalment iterant sobre aquesta metodologia, quan el model dóna una precisió acceptable, s'utilitzen els pesos quantitzats com a definitius per la xarxa.

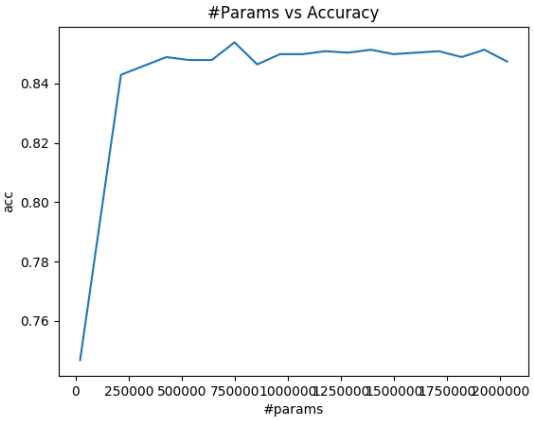

Fig. 8: Percentatge de prunning vs nombre de paràmetres

7 RESULTATS

Per al desenvolupament de la pràctica, s'ha utlitzat una GPU GTX 1080 disponible gràcies al CVC. S'han aplicat totes les proves sobre una xarxa deep learning VGG16 preentrenada i modificada per a detectar 40 senyals de trànsit. Inicialment en el model preentrenat després d'aplicar el fine-tuning, obtenim una precisió del **85%**

7.1 Prunning

Com s'ha explicat anteriorment, durant el prunning, s'utilitza una variable que conté el percentatge dels pesos de la xarxa neuronal que es vol prunar. En la figura (7), es mostren diferents porcentatges de prunning i la precisió del model en cada una. Observant la gràfica, es pot veure que la majoria de pesos que hi han a la xarxa són redundants, ja que només quan el percentatge és superior al 90% la precisió del model es comença a veure afectada.

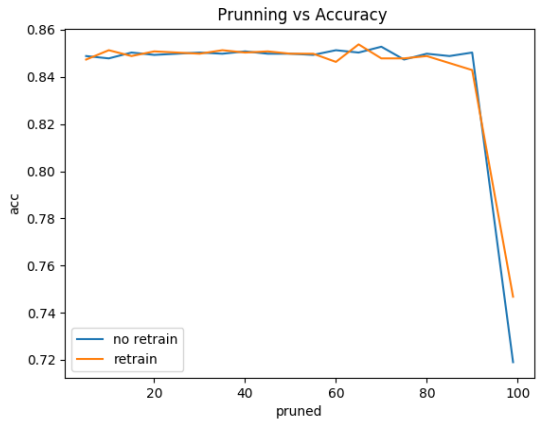

Fig. 7: Percentatge de prunning vs precisió del model

Un atra forma de veure la poda és desde el punt de vista del nombre de paràmetres (8), en la qual treiem les mateixes conclusions que la figura anterior.

Un altra aspecte important de la poda és el tamany dels pesos que obtenim quan fem la poda i el tamany en memòria que aquests ocupen. En aquest aspecte, cada pes es representa amb 32 bits, mentres que els pesos podats es poden representar amb un únic bit, ja que tots els pesos podats tenen un valor de 0. En la figura (9), es defineix de manera aproximada el tamany en memòria que té la xarxa a mesura que anem aplicant un procentatge de poda diferent.

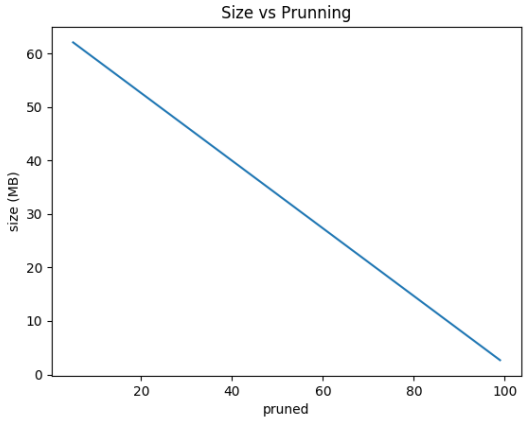

Fig. 9: Percentatge de prunning vs tamany

7.2 Quantització ternària

En la figura (10), observem la precisió del model a mesura que es van aprenen els factors d'escala adequats. Es pot observar, que en les primeres iteracions o èpoques, el model no és capaç de millorar, i no és fins a certa iteració on el model es desencalla i consegueix anar millorant la precisió. Finalment, observem que degut a la quantització, la precisió del model decau gairebé un 30%.

En la figura (11), observem l'error del model a mesura que es van entrenant els factors d'escala.

Degut a la quantització ternària utilitza tres valors per descriure els pesos, podem arribar a obtenir un tamany en memòria x16 vegades més petit que el mateix model amb els pesos amb total precisió.
Podem observar els paràmetres diferents de zero per cada capa convolucional abans (12) i despres de quantitzar (13). On realment obtenim una gran diferència, és en les últimes capes convolucionals, que contenen la major part de paràmetres i degut a la quantització conseguim reduir aquest nombre enormement.

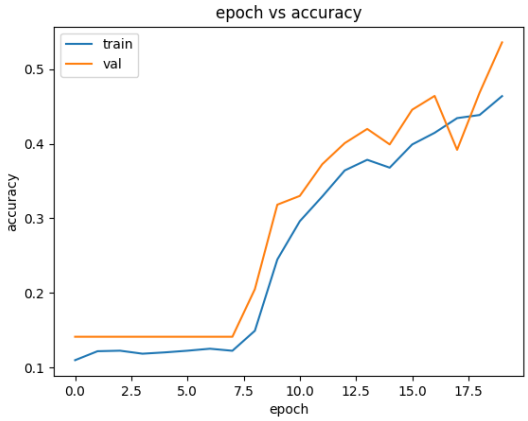

Fig. 10: Nombre d'iteració vs precisió del model

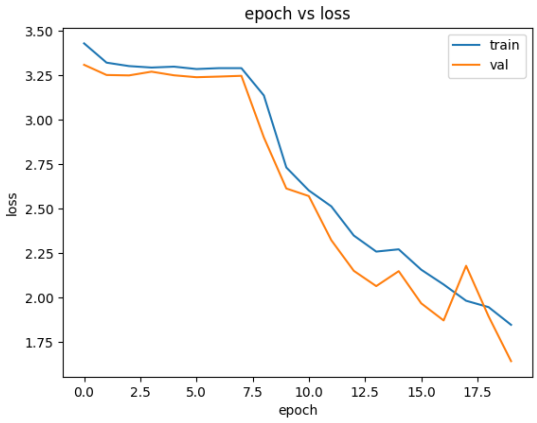

Fig. 11: Nombre d'iteració vs error del model

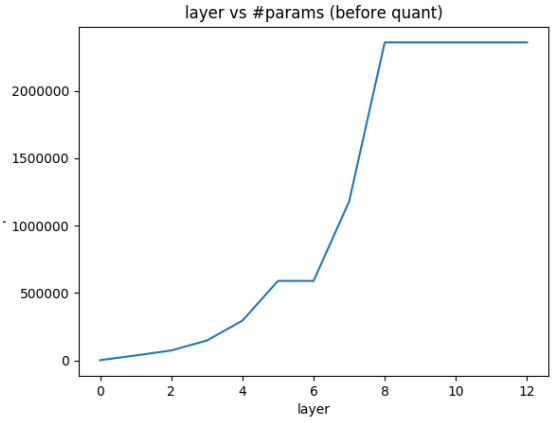

Fig. 12: Nombre de paràmetres per capa convolucional abans de quantitzar

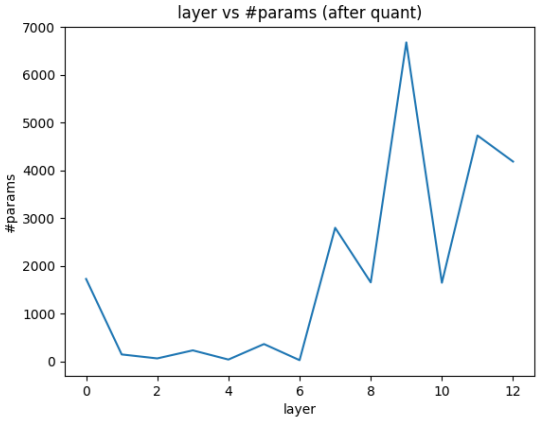

Fig. 13: Nombre de paràmetres per capa convolucional despres de quantitzar

8 Conclusions

En aquest treball, s'han utilitzat dos mètodes que abarquen diferents aspectes sobre l'eficiència en les xarxes neuronals. La poda de paràmetres, que es centra en la reducció del nombre de paràmetres i la quantització ternària que ho fa en la reducció de la precisió dels paràmetres.
Concluir que durant aquest treball s'han pogut implementar aquests mètodes de forma satisfactòria, conseguint una reducció computacional del grans models (en el nostre cas VGG16), fent possible el primer pas per a habilitar als dispositius més simples al processament d'aquest tipus de xarxes.
Les incidències més destacables dins del treball han estat sobretot per la falta de coneixement sobre el framework pytorch i la falta d'informació en alguns punts sobre els mètodes d'eficiència utilitzats en aquest treball.
Com a línies futures, es podria millorar l'aspecte del mètode de prunning, aconseguint que aquest mètode en comptes de deshabilitar la connexió (reduïnt a zero el pes) permeti la eliminació directe de la connexió. Un altra línea futura sería l'ús dels dos mètodes d'eficiència de forma simultànea (conegut com a Deep Compression), aconseguint així una reducció computacional encara més elevada.

Agraïments

Agraïr a la meva família i amics que han estat recolzant-me durant tot el desenvolupament del treball. Agraïr també al meu tutor Joan Serrat per la paciència i els bons consells i guiar-me durant aquest treball. I agraïr al CVC per prestar els recursos necessaris sense els quals aquest projecte no hagués sigut possible.

Referències

[1] Pytorch, `http://pytorch.org/`

[2] Fine-tuning, `http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`

[3] Song Han, *Efficient methods and hardware for deep learning*, págs. 19–60, 2015.

[4] Song Han, Jeff Pool, John Tran i William J. Dally, *Learning both Weights and Connections for Efficient Neural Networks*, 2015.

[5] Abigail See, Minh-Thang Luong i Christopher D. Manning, *Compression of Neural Machine Translation Models via Pruning*, Stanford, CA 94305.

[6] Projectes referència per prunning, https://github.com/wanglouis49/pytorch-weights_pruning, https://jacobgil.github.io/deeplearning/pruning-deep-learning

[7] Backward hook, URL: http://pytorch.org/tutorials/beginner/former_torchies/nn_tutorial.html

[8] YANN LE CUN, JOHN S. DENKER, i SARA A. SOLLA, *Optimal Brain Damage*, Holmdel, N. J. 07733.

[9] EMILY L DENTON, WOJCIECH ZAREMBA, JOAN BRUNA, YANN LECUN i ROB FERGUS, *Exploiting linear structure within convolutional networks for efficient evaluation*, NIPS.

[10] VINCENT VANHOUCKE, ANDREW SENIOR i MARK Z MAO, *Improving the speed of neural networkson cpus*, NIPS Workshop 2011.

[11] WENLIN CHEN, JAMES T. WILSON, STEPHEN TYREE, KILIAN Q. WEINBERGER i YIXIN CHEN, *Compressing neural networks with the hashing trick*, 2015.

[12] FENGFU LI i BIN LIU, *Ternary weight networks*, 2016

[13] CHENZHUO ZHU, SONG HAN, HUIZI MAO i WILLIAM J DALLY, *Trained ternary quantization*, 2016.

[14] SONG HAN, HUIZI MAO i WILLIAM J DALLY, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*, 2016.

[15] Facebook Developer Conference 2017. Delivering Real-Time AI In the Palm of Your Hand. https://developers.facebook.com/videos/f8-2017/delivering-real-time-ai-in-the-palm-of-your-hand

[16] Projecte referència per quantització ternària, https://github.com/TropComplique/trained-ternary-quantization