Grau d'Enginyeria Informàtica

Aprenentatge Automàtic

Computer vision techniques

Joan Balaguer Llagostera
Jaume Adrover Fernández

5 de gener de 2024

Índex

1 Introducció

En aquesta pràctica el nostre objectiu és aconseguir diferentes tasques associades a *computer vision* sobre el dataset de Oxford-IIT dataset. Aquest dataset, està format per 37 races de cans o moixos amb 200 imatges per a cada una. En resum, els nostres objectius són:

- **1. Crear una xarxa neuronal pròpia**: amb els coneixements adquirits a classe, crearem un model nou amb Pytorch i l'entrenarem des de 0.
- **2. Importar model/s existent**: amb al gran capacitat de models que tenim creats a internet, n'agafarem alguns per a comparar el seu rendiment.
- **3. Comparar els dos models anteriors**: després d'haver implementat el nostre model propi i haver importat un ja creat, el nostre objectiu sirà evaluar aquests dos models per a veure quin funciona millor.
- **4. Classificar raça**: volem tenir un model que sigui capaç de distingir entre 37 races diferents de moixos i cans. A diferència dels dos models anteriors, tractam amb un problema de multiclasse.
- **5. Detecció posició del cap**: donades les coordenades del bounding box del cap a un XML, el nostre objectiu és crear aquest requadre amb la millor precisió possible.
- **6. Segmentació de l'animal**: finalment, ens agradaria, donades les segmentacions en fitxers TRIMAPS, poder segmentar els animals de la manera més exacta possible.

Com hem comentat anteriorment, fomentarem principalment l'ús de altres models creats i compararem amb un nostre creat.

1.1 Dataset

El dataset, com hem comentat anteriorment, té suposadament 200 imatges per 37 races de cans o moixos. Anem a veure la distribució exacta que segueix:

	Races	Imtages/carpeta	TOTAL
Cans	25	199.12	4978
Moixos	12	197.58	2371
TOTAL	37	198.62	7349

$$\alpha = \frac{4978}{2371} \approx 2.1$$

Com podem apreciar, tenim 2.1 vegades més informació de cans que de moixos, fet que ens influirà en la tasca de poder classificar entre aquests. Si seguim aquesta ditribució, el nostre model estirà *biased*, és a dir, tendrà un prejudici a classificar cap a la classe majoritària.Tmbé podem dir que els moixos tenen una mitja més baixa de imatges per carpeta ja que deuen faltar més imatges.Tot i això, el darrer comentat no és gaire problema al tenir mitges molt elevades i pròximes a 200.

Més endavant veurem com afrontam aquest problema i preparam les dades.

Respecte a l'etiquetament d'imatges, tenim una carpeta annotations que conté 3 solucions diferents:

- **1. Classificació**: per a obtenir de quin tipus d'animal/raça és cada imatge, tenim dos arxius *trainval.txt* i *test.txt*, que contenen files amb el següent format:

  $$\vec{v} = \{x_0, x_1, x_2, x_3 | \ (x_1 \ \in [1, 37]) \wedge (x_2 \in [1, 2]) \wedge (x_3 \in [1, 37])\}$$

  Concretament, cada variable significa això:

- x_0: nom de l'arxiu sense la extensió, per exemple *Abyssinian_ 100.*
- x_1: nombre enter que ens indica cada raça. Va de 1 a 37, ja que tenim aquest nombre de races
- x_2: nombre que pot tenir els valors 1,2. Si té el valor 1, ens trobam davant un moix, sinós és un ca.
- x_3: similar al nombre que representa les races, pareix un identificador per a cada. Això es informació repetida ja que tenim x_1.

Trainval conté 3680 files, mentre que test té 3669. Si sumam aquests dos nombres ens dona 7349, resultat correcte. Amb aquest tipus de solució podem resoldre el problema de **classificar** cans, moixos i races. Un exemple de fila siria el següent:

$$\vec{v} = [american\_bulldog\_203, 2, 2, 1]$$

Ens trobam davant un ca($x_2 = 2$), raça bulldog americà ($x_1 = 2$) amb l'identificador de $breed\_id(x_3)$ com a 1, que es podria suprimir ja que tenim x_1

- **2. Detecció**: xml amb bounding box. Tenim un arxiu xml que conté la bounding box amb les seves 4 coordenades. Les dades segueixen aquest format:

$$\vec{x} = [(x_0, y_0), (x_1, y_1)]$$

Anem a veure l'exemple sobre la foto *Abyssinian_ 1*:

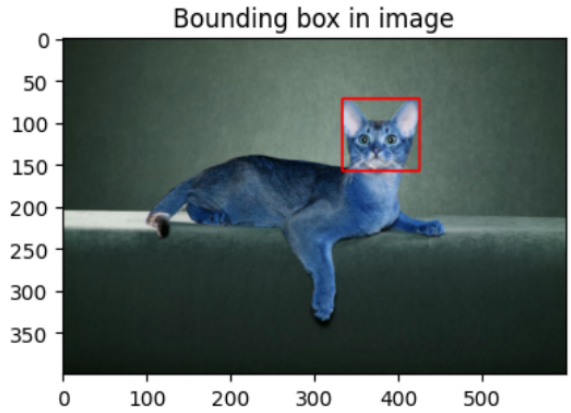

Figura 1: Imatge amb la bounding box

En aquest cas, el nostre vector siria així:

$$\vec{rect} = [(333, 72), (425, 158)]$$

- **3. Trimaps**: com podem apreciar, si obrim les imatges de la carpeta trimaps, només veurem tot negre, tenguent els següents valors:
  - **Valor '1'**: representa la mascota
  - **Valor '2'**: representa *background.*
  - **Valor '3'**: representa els bordes.

S'anomenen **trimaps** ja que estan compostes per 3 seccions: *foreground*,*background* i *unknown/transition*(bordes). Anem a veure que passa si substituim els valors 1,3 per blanc i el fons en negre:

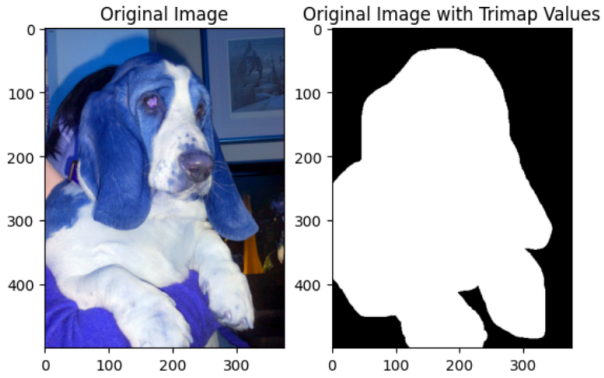

Figura 2: Imatge Original vs Modificada

Com podem apreciar, hi ha diferentes *labels* en funció de la tasca a seguir, fet que ens farà

el mateix nombre d'imatges per a entrenar cada classe, de manera que no hi haurà desigualtats a l'hora de classificar. Al test sí que hi ha un nombre diferent d'imatges per a cada classe, fet que no ens afecta en principi, ja que el model no serà entrenat amb aquestes dades.

2 Data Preprocessing

En aquesta etapa, ens encarregarem d'estructurar les imatges i les seves etiquetes de la millor manera possible. Hem obtingut un objecte dict amb totes les races i els seus id's corresponents dels .txt que hens han donat.A continuació, per a cada foto, ja podem generar la seva solució. Si comença en majúscula sirà un 1(moix), mentre que sinó sirà un 2(ca). D'aquesta manera, afegirem l'atribut class, que sirà el valor del diccionari corresponent.

Anem a veure per passes que hem fet:

- **1. Reestructurar imatges dins carpetes train, test**
  - **1.1 Agrupar fotos de races per carpetes**: cada carpeta contendrà només fotos de races. Agrupam per a evitar que si ho feim aleatòriament es creïn desbalancejos entre classes. D'aquesta manera, sempre agafarem el mateix nombre de races a train i test
  - **1.2 Agafam 80% de cada carpeta**: agafarem aquest percentatge per a entrenament i el 20 per a testejar.
- **2. Reestructurar etiquetes a csv**
  - **2.1 Escanejar informació actual**: hem escanejat i obtengiut una variable diccionari que obté l'identificador que té cada raça. Ho hem extret dels dos .txt continguts a *annotations*, carpeta inicial.
  - **2.2 Generar files csv**: a continuació, sabent ja els id's de cada raça,podem crear les files que contindrà el csv:

$$\vec{v} = \{x_0, x_1, x_2\}$$

$x_0 = filename$
$x_1 = breed \in [0, 36]$
$x_2 = specie \in [0, 1 \mid (Cat = 1) \ \wedge \ (Dog = 0)]$

- **3. Comprovació manual**: hem mirat si el nombre de imatges dins les carpetes *train*,*test*(properties) és igual al nombre de files del csv.

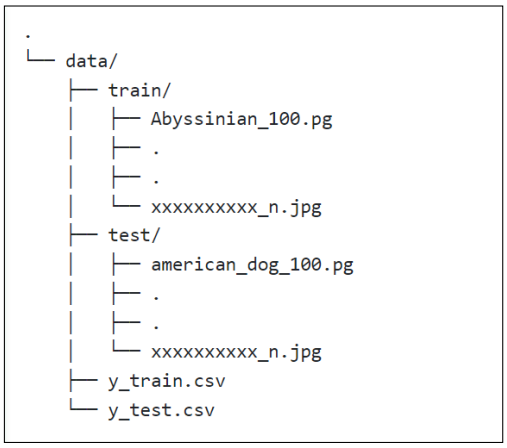

Figura 3: Estructura final

3 Workflow

En aquesta secció, veurem com funciona tot el nostre workflow de la pràctica. Bàsicament, podrem diferenciar dues fases:

- **1. Data Extraction**: en aquesta fase, ens encarregarem d'extreure/obtenir la informació segons quin problema és el que volem resoldre. No tendrem la mateixa label en cas de fer segmentació o classificació.
- **2. Model Creation**: en la etapa final crearem un model i l'entrenarem, testejant-lo a cada epoch. Així, finalment, tendrem un model creat amb una certa precisió.

3.1 Data Extraction

En aquesta primera i no menys important fase, ens encarregam d'obtenir les dades d'entrenament i test amb el format desitjat, que vendria a ser el següent:

$$\vec{v} = [\ (img_0, y_0), (img_1, y_1), ..., (img_n, y_n)\ ]$$

$img_i = M_{m \times n \times 3}$

$y_i \in ([0,1] \vee [0,36] \vee [x_{min}, x_{max}, y_{min}, y_{max}] \vee\ trimap)$

És a dir, volem obtenir duples de matriu imatge i y_i, que podrà ser binari(cans vs moixos), [0,37] (classificar races), bounding box(detecció del cap) o trimap(segmentació).

Això es consegueix amb el fluxe que podem areciar a continuació:
Dataset vendria a ser la classe en *python* que fa tot el tractament per dintre. Anem a veure com funciona per dedins aquesta caixa negra:

- **1. Selecciona una imatge**
- **2. Processa la imatge en forma de matriu**
- **3. Transforma la imatge(resize + normalització)**
- **4. Cerca la imatge dins el csv**: agafa la label desitjada.
- **5. Concatena amb forma de dupla**
- **6. Torna al primer pas si queden imatges**

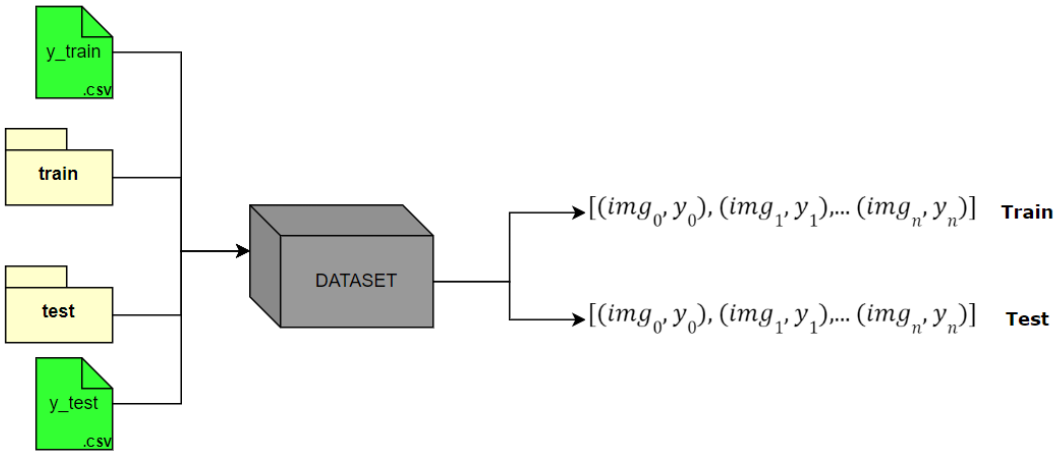

Figura 4: Procés d'extracció/format de dades

Això es fa tant per train com per test, de manera que s'instancia un objecte amb dos atributs que contenen dues llistes. D'aquesta manera, accedim amb *obj.train* i *obj.test* a l'informació corresponent.

3.2 Model Creation

Finalment, el que ens interessa després de tenir totes les dades preparades, és construir el model i entrenar-lo. Això ho aconseguim amb la classe *ModelTrainer.py*, que s'encarrega, donada una configuració, de entrenar i testejar un model.

Anem a veure com està construït aquest fluxe:

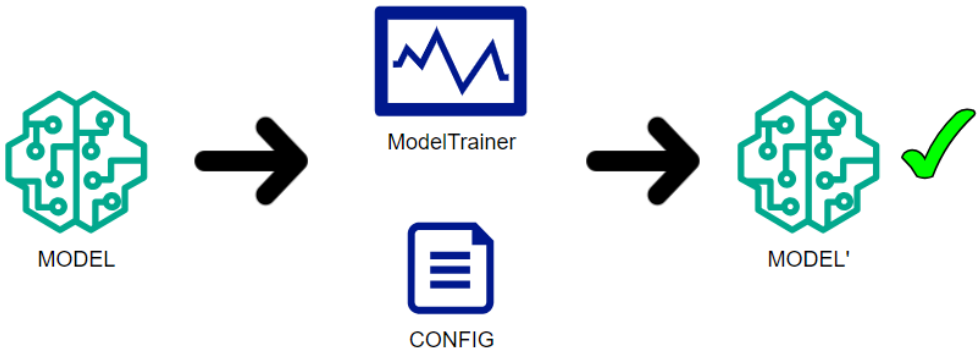

Figura 5: Procés de creació d'un model ja entrenat

Bàsicament, la configuració que rep el *ModelTrainer* és una funció de pèrdua, optimitzador, nombre de epochs etc. Aquest s'encarrega de unificar el procés d'entrenament per a que pugui ser aplicat a cada model amb la major abstracció possible. El resultat és un model suposadament entrenat i amb una precisió. Els models estan emmagatzemats a la carpeta **models/classifiers**, en format .ipynb. Els pesos, es troben en la carpeta weights, amb el nom del model al que pertanyen.

4 Classificació binària(cans vs moixos)

Aquest primer problema a resoldre consisteix en, donada una imatge, detectar si es tracta d'un moix o un ca. Utilitzarem un model ja preentrenat com sirà *AlexNet*, comparat amb un creat per nosaltres anomenat *AdroBalaNet*.

4.1 BinaryAlexNet

Anem a veure com funciona la nostra custom AlexNet. Primer ensenyarem un poc els aspectes tècnics i després com ha estat el procés d'entrenament.

4.1.1 Aspectes tècnics

Anem a veure coses rellevant del nostre model:

- **1. Nº paràmetres entrenables**: 54.011.394
- **2. Input size**: $227 \times 227 \times 3$.
- **3. Format de la sortida**: la sortida segueix aquest format:

$$\vec{v} = \{x_0, x_1 \mid (x_0 + x_1 = 1)\}$$

4.1.2 Entrenament del Model

Pel que fa a l'entrenament del model, hem volgut optar per un procés de *transfer learning*, degut a la escassa potència computacional que tenim. Així, congelarem tota la part de les característiques de l'AlexNet i després implementarem un classificador manual.

Hem obtingut els següents valors de pèrdua i percentatge de precisió:

Època	Pèrdua mitjana	Precisió (%)
1	0.0038	93.40
2	0.0037	94.69
3	0.0040	91.97
4	0.0037	94.83
5	0.0042	89.66
6	0.0038	93.95
7	0.0038	94.35
8	0.0040	92.18
9	0.0037	94.97
10	0.0038	93.67

Taula 1: Pèrdua i precisió mitjana per època

Amb aquests valors, mostram la corresponent funció de pèrdua durant l'entrenament realitzat i el test i per guardar els pesos del model, hem guardat els de l'època que millor precisió ha tingut:

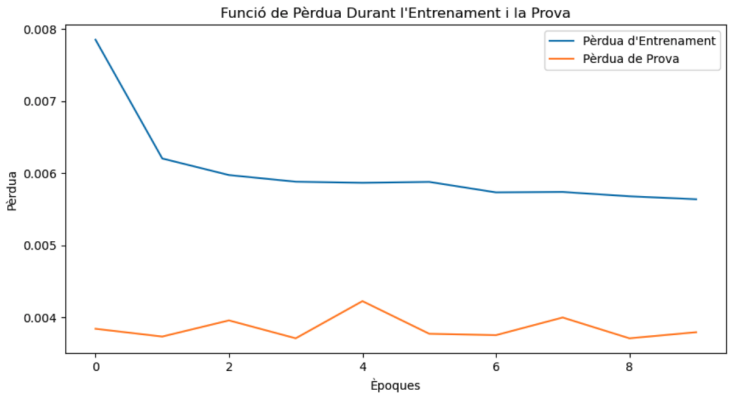

Figura 6: Funció de pèrdua per a l'entrenament de l'AlexNet

Amb els resultats obtinguts, podem extreure les següents conclusions:

- **Pèrdua durant l'Entrenament i la Prova:** La gràfica de pèrdua mostra una disminució pronunciada de la pèrdua d'entrenament (línia blava) en les primeres èpoques, estabilitzant-se a mesura que progressa l'entrenament. La pèrdua de prova (línia taronja) es manté relativament estable després d'una inicial disminució, suggerint que el model no està excessivament entrenat amb les dades d'entrenament.
- **Estabilitat en la Precisió:** Observant els resultats de precisió, podem veure que el model manté una precisió d'aproximadament el 94% a partir de la segona època, amb algunes variacions menors d'una època a l'altra. Això podria indicar que el model ha après característiques generals eficaçment i no necessita moltes èpoques per a consolidar el seu aprenentatge en aquest conjunt de dades.
- **Variabilitat entre les Èpoques:** Hi ha una certa variabilitat en la precisió i la pèrdua entre les èpoques, especialment notable en la cinquena i sisena èpoques on la precisió disminueix abans de recuperar-se. Això pot ser una indicació de la presència de característiques específiques en el conjunt de dades de prova que el model troba més difícils de generalitzar.

Com a conclusió, podem dir que el model **BinaryAlexNet** mostra un bon rendiment en la tasca de classificació entre gossos i gats. La ràpida estabilització de la pèrdua i l'alta precisió obtinguda són indicatius d'un model robust.

4.2 BinaryAdroBalaNet

En aquest apartat explicarem la composició de la xarxa sense pre entrenament que hem creat per la classificació binària entre cans i moixos. La xarxa es compon de les següents parts:

- **Capa de Característiques (Features):** Aquesta secció de la xarxa utilitza diverses capes convolucionals, les quals són responsables de l'extracció automàtica de característiques visuals des de les imatges d'entrada. Cada capa convolucional aprèn a identificar diferents atributs de les imatges, com ara formes i textures, que són crucials per a la distinció entre les classes. En aquest cas, s'utilitzen les capes de característiques d'AlexNet, excepte l'última capa convolucional, per adaptar la xarxa a la tasca específica de classificació binària.
- **Pooling Adaptatiu (Adaptive Pooling):** Després de passar per les capes convolucionals, s'aplica un AdaptiveAvgPool2d per reduir les dimensions espacials de les representacions de les imatges a una mida fixa. Això permet que la xarxa manegi imatges d'entrada de diferents dimensions i normalitza l'output per a la següent etapa de classificació.
- **Classificador**: La informació processada és aplanada en un vector unidimensional i passada a través d'una sèrie de capes fully connected. La primera capa lineal redueix la dimensió d'un vector llarg de característiques a una representació més compacta de 128 elements. Aquesta etapa també inclou l'ús

de la funció d'activació ReLU per introduir no-linealitats i la tècnica de Dropout per minimitzar el risc d'overfitting. Finalment, una última capa lineal converteix aquestes 128 característiques en dos outputs, corresponents a les dues classes (gos i moix).

4.2.1 Aspectes Tècnics

Anem a veure els aspectes més rellevants del nostre model:

- **1. Nº paràmetres entrenables**: 3.649.730
- **2. Input size**: $227 \times 227 \times 3$.
- **3. Format de la sortida**: la sortida segueix aquest format:

$$\vec{v} = \{x_0, x_1 \mid (x_0 + x_1 = 1)\}$$

4.2.2 Entrenament del model

En aquest apartat veurem els resultats que hem obtingut de l'entrenament de la nostra xarxa per a la classificació binària i intentarem interpretar els resultats obtinguts. Per començar, aquests han estat la pèrdua i precisió mitjanes durant l'entrenament del model:

Època	Pèrdua mitjana	Precisió (%)
1	0.0065	67.35
2	0.0062	67.35
3	0.0059	67.35
4	0.0063	67.62
5	0.0057	67.76
6	0.0056	73.95
7	0.0053	73.20
8	0.0053	71.02
9	0.0054	75.65
10	0.0052	75.92

Taula 2: Pèrdua i precisió mitjana per època

Amb aquests valors, mostram la corresponent funció de pèrdua durant l'entrenament realitzat i el test i per guardar els pesos del model, hem guardat els de l'època que millor precisió ha tingut:

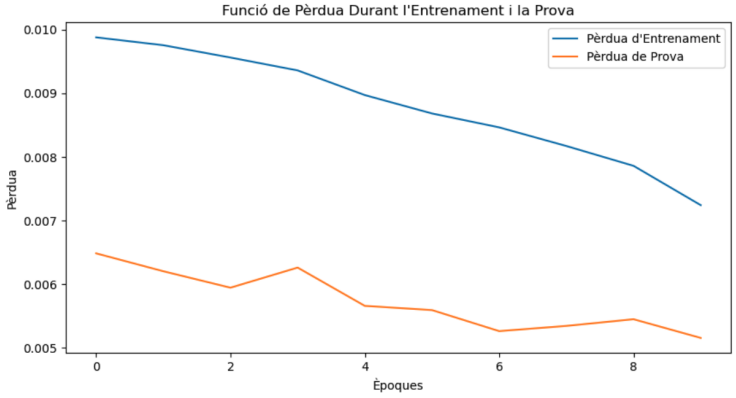

Figura 7: Funció de pèrdua per a l'entrenament de l'AdroBalaNet

Observant aquests resultats obtinguts, podem extreure algunes reflexions:

- **Tendències de Pèrdua:** La gràfica de pèrdua mostra que la pèrdua durant l'entrenament (línia blava) disminueix de manera consistent a mesura que augmenten les èpoques, la qual cosa és un indicador que

el model està aprenent adequadament. La pèrdua de prova (línia taronja) també disminueix, encara que amb algunes fluctuacions. Això pot indicar que el model generalitza raonablement bé a dades no vistes durant l'entrenament, però podria beneficiar-se d'una regularització addicional per suavitzar aquestes fluctuacions.

- **Precisió del Model:** La precisió del model en el conjunt de proves comença en un 67.35% i mostra una millora significativa a l'entorn de la sisena època, arribant a un 73.95%. A partir d'aquest punt, la precisió segueix augmentant fins a un 75.92% a l'última època. Això indica que el model millora la seva capacitat de distingir entre fotos de gats i gossos a mesura que s'entrena amb més dades.
- **Possible *overfitting*:** Tot i que no es disposa de dades suficients per fer una afirmació definitiva, la presència de fluctuacions en la pèrdua de prova podria ser un senyal d'overfitting. Això es podria mitigar amb tècniques com l'augmentació de dades o l'aplicació de més dropout (tècniques que no es realitzaran en aquesta pràctica per falta de temps).

En conclusió, el model **BinaryAdroBalaNet** demostra una capacitat d'aprenentatge efectiva en la tasca de classificació binària. Però, com en qualsevol model d'aprenentatge automàtic, hi ha espai per a la millora. La implementació de tècniques addicionals per evitar l'overfitting i l'optimització dels hiperparàmetres podrien contribuir a millorar la precisió i estabilitat de la pèrdua en el conjunt de proves.

5 Classificació Multiclasse (Detecció races)

Al tenir un altre problema de classificació però amb més classes, el més important és canviar el nombre de sortides del model, de manera que enlloc de tenir dues sortides per cans o moixos, tendrem tantes sortides com classes haguem de prediure. És a dir, els nostres models hauran de tenir 37 sortides diferents, una per a la probabilitat de que sigui de cada raça.

5.1 MultiAlexNet

El nostre model triat és similar al binari, basat en AlexNet, però canviant les sortides. Tenim dues parts:

- **1. Backbone**: vendria a ser la part de *features* de la pròpia AlexNet.
- **2. Classificador**: aquesta darrera secció té diversos blocs amb capes linears, ReLU i Dropout, de manera que acaba en 37 sortides. Més detallat al fitxer **MultiAlexNet.py**

5.1.1 Aspectes tècnics

Anem a veure els aspectes més rellevants del nostre model:

- **1. Nº paràmetres entrenables**: 54.047.269
- **2. Input size**: $227 \times 227 \times 3$.
- **3. Format de la sortida**: la sortida segueix aquest format:

$$\vec{v} = \{x_0, x_1, ..., x_{36} \mid (\sum_{i=0}^{n} x_i = 1)\}$$

5.1.2 Entrenament del model

Per a entrenar el model, hem decidit aplicar-hi 30 epochs, al ser un model més gran i amb més característiques a aprendre. La millor accuracy que hem obtingut ha estat del 33.2%.

Podem observar que aquest model no ha aconseguit un resultat tan bo com la resta, ja que hi ha moltes més possibilitats i menys mostres per classe. A més, podem apreciar com pareix que ambdues línies de pèrdua no baixen tant ràpidament com la resta, sino que pareixen estabilitzar-se en el darrer tram 25-30.

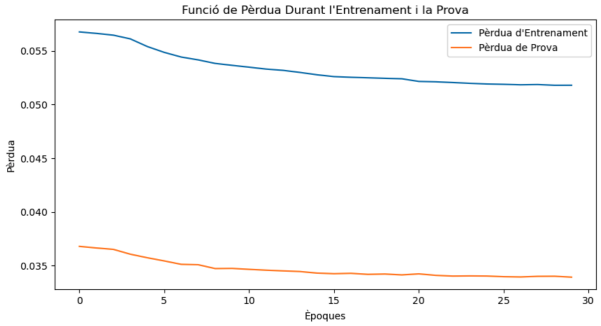

Figura 8: Funció de pèrdua per a la MultiAlexNet

6 Conclusions

Aquest apartat es veurà dividit en 3 tipus de conclusions: comparar els models, unes reflexions més genèriques i possibles canvis a futurs projectes similars.

6.1 Comparacions

Anem a veure la comparació entre els diferents models que hem creat:

	Epochs	**Average Loss**	**Accuracy(%)**	**Training Time**
AlexNet	10	0.0037	94.97%	14m 37s
AdroBalaNet	10	0.0051	75.92%	29m 35s
MultiAlexNet	30	0.0339	33.2%	28m 34s

Els dos primers, que fan referència a problemes de classificació binaris, mostren un resultat molt interessant. Podem apreciar que obtenim una millora de pràcticament un 20% amb un temps d'entrenament que és la mitat. Per si no era suficient amb la millora d'*accuracy*, el temps d'entrenament és molt menor. És a dir, que AlexNet, el model on hem aplicat *transfer learning*, ha demostrat ser el rei i la millor solució de la classificacií binària.

Per l'altra banda, si observam la MultiAlexNet, podem veure que amb el triple nombre d'epochs, no arribam a una precisió tan bona com podria ser un dels models de classificació anteriors. Això ens demostra que aquests models necessiten molt més entrenament per a poder assolir les característiques de cada raça i una precisió més elevada. Tmabé podem apreciar que, al ser un model carregat, ha tardat gairebé el mateix que el nostre propi binari però fent 3x més epochs.

6.2 Genèriques

Personalment, hem extret conclusions molt interessants de cara a la realització d'aquesta pràctica. Anem a veurer-les:

- **1. Construïr models**: cal destacar la increïble diferència entre els models ja importats o els creats per nosaltres. Tot i utilitzar estructures similars, podem arribar a tenir una diferència d'*accuracy* $>= 20\%$. Considerant això, veim que la construcció de models des de 0 és molt més costosa computacionalment i duu pitjors resultats. Hem de tenir en compte que els models ja creats provenen d'organitzacions o grups amb molta més capacitat que nosaltres i han estat entrenats amb molta més *data* que 10k fotos de cans i moixos.
- **2. Reutilitzar arquitectures**: com a conseqüència del punt anterior, el que més ens convé es fer *transfer learning*, ja que és la tècnica que ens fa evitar gran part de l'entrenament i adequar model x per a que sigui més específic.

- **3. Escassa potència computacional**: finalment, ens hem adonat que la nostra potència computacional(no tenim GPU) no és elevada per a agilitzar els processos d'entrenament, de manera que ens hem vist un poc limitats segons amb quines coses.

6.3 Possibles Millores

En aquest mini apartat, farem una llista de possibles millores en cas d'assolir futurs projectes similars a. Si aconseguim més capacitat computacional, milloraríem això:

- **1. Major nombre d'epochs**: ens centraríem en fer més combinacions amb més o menys epochs, per a veure com avança fins a la convergència o detectar possible overfitting.
- **2. Data Augmentation**: hem de tenir en compte que el nostre dataset no abarca molts dels possibles casos reals de imatges, per exemple, no tenim casi fotos obscures, pocs angles de rotació, etc. Hauríem de modificar i intentar obtenir més imatges, ja sigui aplicant tècniques com canviar colors, angles de rotacions, mosaics, etc. D'aquesta manera aconseguiríem models més pròxim al sistema real.
- **3. Càrrega de models més grans**: ens agradaria experimentar amb models més grans, de manera que els reentranaríem sencers. Així veuríem el costós que és realment aconseguir un model de certa precisió

7 Bibliografia

- ChatGPT
- Tree Diagram Generator
- Draw.IO