

Deep Learning Project - Image Classification

Universitat de Girona



U1934690 - Francesc Xavier Reverté Baró

Advanced Machine Learning

21 - 04 - 2023

Índex

Índex	2
Introducció	3
Plantejament del problema	4
VGG19	5
Fine tuning 1 {1.5 (lr=0.015) [0.874300]}	6
Transformacions en DataBlock i Dataloader	6
1.1 (FreezeE1) [0.835500]	6
1.3 (lr=0.005) [0.854500]	6
1.5 (lr=0.015) [0.874300]	7
Fine Tuning 2 {2.10 (lr=0.03) [0.878300]}	7
Transformacions en DataBlock i Dataloader	8
2.0 Suggested learning rate	8
2.2 (lr=0.0007) [0.790200]	8
2.7 (lr=0.015) [0.872500]	9
2.10 (lr=0.03) [0.878300]	9
Fine Tuning 3 {3.6 (lr=0.03) [0.869800]}	9
Transformacions en DataBlock i Dataloader	10
3.0 Suggested learning rate	10
3.4 (lr=0.015) [0.867500]	11
3.6 (lr=0.03) [0.869800]	11
Fine tuning 4 (progressive resizing) {[0.908600]}	12
Iteració 1 (lr=0.02, bs=128, size=128) [0.895800]	12
Iteració 2 (lr=0.015, bs=128, size=256) [0.906100]	13
Iteració 3 (lr=0.01, bs=64, size=512) [0.908600]	13
Param Tuning 2{weight_d=0.0, learning_r=0.02, dropout=0.0, batch_s=128}[0.9000]	14
Param T 6{weight_d=0.0, learn_r=0.04, freeze_e=2, epochs=7, dropout=0.0, batch_s=256}[0.9156]	15
TESTING VGG19 7 (progressive resizing) [0.914800]	15
ResNet34	17
Param Tuning 1 {weight_d=0.0, learning_r=0.02, freeze_e=1, dropout=0.0, batch_s=256}[0.9147]	18
Param Tuning 2 {weight_d=0.0, learning_r=0.01, freeze_e=1, dropout=0.001, batch_s=256}[0.914800]	18
Param Tuning 3 {weight_d=0.001, learning_r=0.02, freeze_e=3, dropout=0.001, batch_s=256}[0.9178]	19
Param T 4 {weight_d=0.001, learn_r=0.01, freeze_e=3, epochs=7, dropout=0.001, batch_s=256}[0.9278]	19
TESTING ResNet 5 (progressive resizing) [0.968800]	20
TESTING DenseNet 4 (progressive resizing) [0.972400]	21
TESTING EfficientNet 4 (progressive resizing) [0.995200]	23
Ensamble	24
Conclusions	25

Introducció

L'objectiu principal d'aquest projecte és el desenvolupament de tècniques de deep learning per a la classificació d'imatges. En concret, utilitzarem el conegut i internacional dataset "CIFAR-10", que consisteix en 60.000 imatges amb color, de mida 32x32, agrupades en 10 classes diferents ("airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship" i "truck").

S'ha dividit el conjunt de dades en 3 datasets, seguint la següent estructura:

- 40.000 imatges pel dataset d'entrenament, anomenat "train".
- 10.000 imatges pel dataset de validació "validation".
- 10.000 imatges pel dataset de testatge "test_unlabelled".

Aquests datasets prenen la forma de carpetes, dins les quals es troben el conjunt d'imatges agrupades en subcarpetes, anomenades segons la classe a la qual pertanyen (excepte pel cas de "test_unlabelled", que presenta totes les imatges bolcades a la carpeta, sense anomenar). Els 3 datasets es troben balancejats quant al nombre d'imatges per classe.

Per a la realització del projecte s'ha treballat amb el Google Collab framework, utilitzant la llibreria "fastai". S'ha triat un enfocament de "transfer learning", aprofitant models ja entrenats i ajustant-los al nostre cas d'estudi mitjançant "fine tuning".

Per analitzar i reportar els resultats dels nostres models usarem les mètriques obtingudes a partir de la contrastació de l'entrenament "train" amb la validació "validation". A les conclusions del treball, però, un cop seleccionat el millor model, proporcionarem els resultats en el dataset "test_unlabelled".

Cal mencionar que la intenció no és seleccionar només un model, sinó basar les prediccions en funció de diferents models, mitjançant un ensamble. Per tractar que les prediccions siguin el més encertades possibles, s'entrenaran els models escollits amb els dos conjunts d'imatges "train" i "validation" (seleccionant, com a nou conjunt de validació, una proporció aleatòria del 0.05 entre el nou conjunt d'entrenament).

Com s'ha comentat, ens centrarem només a fer "fine tuning"; altres apropaments, com la creació d'un model propi, s'han considerat, però s'han hagut de descartar a causa de les limitacions d'ús de la GPU del Colab, que en general ha anat entorpiat tot el projecte.

S'han utilitzat 4 models preentrenats diferents: VGG19, Resnet34, DneseNet121, i EfficientNet_b0. El primer ha estat el VGG19, de manera que és en el que més s'evidencia la falta de coneixements i pràctica pel que fa a la llibreria fast_ai i al cas d'estudi. Pel que fa a la resta de models, s'usa un apropament més òptim i estandarditzat. És per això que en l'informe s'explicarà amb molt detall el primer dels models, detallant l'evolució del plantejament. També s'explicarà amb cert detall DenseNet12, però, al compartir la mateixa estratègia que la resta de models, aquests altres no rebran massa importància.

Cal mencionar que, donada la gran quantitat de figures en l'informe, i al fet que totes elles són d'elaboració pròpia i es descriuen en el text, s'ha decidit no enumerar les figures ni col·locar cap peu de figura. Només aconseguiríem saturar l'informe d'informació redundant i avorrir tant a l'escriptor com al lector.

Plantejament del problema

Per poder fer una tasca complexa com la classificació d'imatges entre 10 possibles categories, és recomanable aprofitar models ja entrenats. El disseny d'un model des de zero, encara que l'arquitectura del model pugui ser més adequada pel cas d'estudi en concret, no compensa la falta de dades d'entrenament. La idea que les dades importen més que els algoritmes per problemes complexos va ser popularitzada per Peter Norvig, en l'article “The Unreasonable Effectiveness of Data”.

En el context de la llibreria “fastai”, s'utilitza el “DataBlock” per definir els passos per preparar les dades pel model. Et permet especificar els tipus d'entrada de dades i objectius de predicció, les transformacions que s'aplicaran a les dades per a la seva preparació i com dividir les dades entre el conjunt d'entrenament i validació.

El “DataLoader” s'usa per carregar les dades en batches i aplicar transformacions durant l'entrenament i validació del model. Rep el DataBlock com a input, defineix la mida del batch i aplica les transformacions a les dades indicades per “data augmentation”.

La idea general és trobar la combinació adequada de transformacions en el dataloader i el datablock, buscant un equilibri entre un accuracy prou elevat i una regularització de les dades per evitar overfitting mitjançant data augmentation.

Un cop trobat aquest equilibri, se selecciona un model ja entrenat per aplicar transfer learning i es proven un seguit de combinacions d'hiperparàmetres del model per trobar la més adequada. Els paràmetres que s'han tingut més present han estat learning rate, freeze epochs, wheight decay i *dropout (que forma part del learn, no del fine_tune) i *batch size (que forma part del DataLoader).

Cal comentar que per estriar el learning rate, en un principi ens hem ajudat de les recomanacions i gràfics extrets amb <<learn.lr_find()>> fins que ens vam adonar que el valor adequat era sempre notablement major (més d'1 ordre de magnitud). Pel que fa al batch size, en general, com més gran, més representatiu de la totalitat de les dades i, per tant, millor; però ha estat fortament delimitat per la memòria RAM disponible. Respecte al nombre d'epochs dels models d'entrenament, per motius de limitació temporal, en general, s'han establert en un valor de 3.

Un cop trobada la millor combinació de paràmetres pel model de transfer learning seleccionat, s'ajunten totes les dades de “train” i “validation” en una sola carpeta per poder fer l'entrenament amb una major quantitat de dades i obtenir unes millors prediccions en “test_unlabelled”. Per poder fer això és necessari canviar el nom de les dades de validació (afegint un prefix “valid_” a cadascuna d'elles), si no hi hauria hagut conflictes per noms repetits amb les dades d'entrenament. Un cop agrupades, s'ha modificat el DataBlock per canviar el mètode de divisió (entrenament - validació) de “GrandparentSplitter” a “RandomSplitter”, permetent fer aquesta divisió entre els elements d'una mateixa carpeta. S'ha seleccionat una proporció de 0.05 per al conjunt de validació (utilitzant GrandparentSplitter tindria una proporció de 0.2).

Com a incís, els primers testatges en els quals agrupàvem les dades de “train” i “validation”, vam definir una random_seed, per tal que l'exemple fos reproduïble. Més tard ens vam

adonar que, de cara a construir l'ensamble, ens interessava que hi hagués aleatorietat en la divisió de dades, així que alguns exemples presenten una llavor definida i d'altres no. Pel que fa als testatges, s'han fet servir més de 3 epochs (més de 10 en algun cas).

Es repeteix aquest procediment tantes vegades com models preentrenats diferents vulguem usar. Per cada un dels testatges es descarregaran les prediccions en format csv. Finalment, s'acoblaran aquestes prediccions d'acord amb la mitjana ponderada en funció del rendiment del model (mesurat amb el seu accuracy).

Potser no es tracta d'una bona pràctica, però hem realitzat diversos testatges per cadascun dels models de transfer learning. Tenint en compte que no disposem de les etiquetes pel conjunt de "test_unlabelled", ens ha semblat que disposar de diferents testatges, amb petites variacions en els seus paràmetres, enriquiria el plantejament i nodriria a l'ensamble, traduint-se en prediccions més acurades. L'ensamble és una combinació de les prediccions fetes per 9 models: 1 de VGG19, 2 de ResNet34, 3 de DenseNet i 3 d'EfficientNet. El criteri ha estat estriar 1 per cadascun dels 4 diferents models preentrenats, i tots aquells l'accuracy dels quals superi el 0.95. Els testatges que podreu trobar en aquest informe, però, només són 1 per cadascun dels models de transfer learning mencionats.

VGG19

Tal com s'ha comentat, l'estrategia seguida per trobar una bona combinació de paràmetres i hiperparàmetres d'aquest primer model no ha estat massa ben dissenyada, posant de manifest la falta d'experiència i coneixements del cas d'estudi i la llibreria "fastai".

Aquest primer model és l'únic en el qual es van provar diferents combinacions de transformacions d'imatge i de batch, tant en el datablock com en el dataloader. Un cop seleccionada la combinació de transformacions, s'ha aplicat la mateixa pels diferents models de transfer learning. Entenem que l'indicat seria provar diferents combinacions per cadascun dels models preentrenats, però, com comentem, ens hem trobat molt condicionats per les limitacions d'ús de la GPU pública del Google Colab.

El punt de partida va ser intentar trobar una bona combinació de transformacions en el DataBlock i DataLoader. Cada una d'aquestes combinacions queden delimitades pel que anomenarem "apropaments", per exemple "Fine tune x", és a dir: dins el mateix apropiament, aquestes transformacions no varien. El cas ideal seria que per cada una d'aquestes combinacions de transformacions, es testegi el model assignant-li diferents combinacions de paràmetres pel model (el que anomenarem "aproximacions").

Cada apropiament, per exemple "Fine tuning 1 {1.5 (lr=0.015) [0.874300]}" indica entre els símbols " { } " l'aproximació que més accuracy presenta. Cada aproximació presenta una estructura com aquesta: "1.5 (lr=0.015) [0.874300]". El número inicial jerarquitza el nombre de l'aproximació; entre parèntesis trobem els paràmetres modificats i/o comentaris; i entre claudàtors el valor d'accuracy obtingut (generalment en l'epoch 3).

En general, podem veure com no s'ha dissenyat la millor de les arquitectures per trobar un model òptim, però, al llarg d'aquest mateix apartat "VGG10", veurem com s'anirà millorant. En els primers 3 apropiaments, per exemple, (Fine tuning 1, 2 i 3), només es proven diferents valors per a un únic paràmetre (learning rate) per les 3 diferents combinacions de

transformacions de les imatges i batch en el DataBlock i DataLoader. A partir de llavors, veurem com s'intenten diferents estratègies més sofisticades.

Fine tuning 1 {1.5 (lr=0.015) [0.874300]}

Aquí trobem el primer apropament “Fine tuning 1”, que ens indica també quina ha estat la millor de les seves aproximacions “[1.5 (lr=0.015) [0.874300]]”. Segons la combinació de transformacions definides en el DataBlock i DataLoader, s’ha trobat que el millor learning rate = 0.015.

Transformacions en DataBlock i Dataloader

En el DataBlock s’apliquen transformacions a les imatges per estandarditzar-les totes. Veiem com apliquem redundantment dues transformacions de mida. En el batch s’aplica una normalització en funció de “imagenet_stats”.

```
item_tfms=[RandomResizedCrop(224, min_scale=0.5), FlipItem(p=0.5), Resize(224)]
batch_tfms=[Normalize.from_stats(*imagenet_stats)]
```

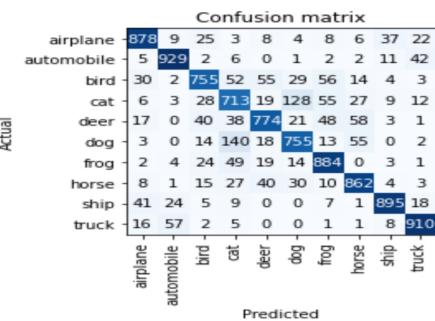
En el DataLoader no s’aplica cap transformació i se selecciona un batch size de 64 (cada batch tindrà 64 imatges).

```
dataloaders(path, bs=64, verbose=True)
```

1.1 (FreezeE1) [0.835500]

Així, aquesta és la primera aproximació del primer apropament, “modificant” el paràmetre “freeze_epochs=1” (*encara que realment es tracta del valor per defecte) i presenta un accuracy de 0.8355 en la tercera època. A no ser que hi hagi res a destacar, les diferents aproximacions només s’explicaran amb la taula de resultats de l’entrenament i la matriu de confusió, com els que veiem a continuació:

	epoch	train_loss	valid_loss	accuracy	error_rate	time
	0	1.173332	0.863180	0.689900	0.310100	04:42
	epoch	train_loss	valid_loss	accuracy	error_rate	time
	0	0.785329	0.602730	0.787200	0.212800	11:35
	1	0.656582	0.485815	0.829800	0.170200	11:35
	2	0.590823	0.461003	0.835500	0.164500	11:35



1.3 (lr=0.005) [0.854500]

Podem veure com saltem de l’1.1 al 1.3, aquí no mostrarem tots els apropaments, només aquells mínimament destacables. En aquest hem aconseguit una millora en la precisió i una reducció en l’error.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.228516	0.891664	0.678400	0.321600	04:44
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.782590	0.583277	0.795400	0.204600	11:38
1	0.612315	0.448171	0.847200	0.152800	11:37
2	0.491962	0.405499	0.854500	0.145500	11:37

Confusion matrix											
		Actual									
		airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane		872	13	25	5	7	4	9	7	38	20
automobile		3	941	1	3	0	2	2	1	6	41
bird		31	3	778	43	52	24	48	17	3	1
cat		8	2	32	720	25	121	58	19	6	9
deer		13	0	38	41	801	20	42	42	2	1
dog		1	0	20	126	19	789	8	35	0	2
frog		5	3	12	38	15	13	912	0	1	1
horse		4	1	16	18	36	37	4	880	2	2
ship		30	15	8	8	2	0	4	0	915	18
truck		12	35	1	4	0	0	2	2	7	937
Predicted											

1.5 (lr=0.015) [0.874300]

Aquesta es tracta de la millor aproximació per les transformacions fets en el Fine tuning 1. A partir d'ara (encara que no en tots els casos) disposem també d'una figura detallant les 9 imatges que més "loss", o error, han tingut (que més s'han allunyat en la predicción). Hi trobem la predicción, el valor real, loss i la probabilitat. *(només mostrarem 6 de les 9 imatges per no saturar l'informe).

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.213991	0.943371	0.661700	0.338300	04:50
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.813642	0.624871	0.780500	0.219500	11:21
1	0.592981	0.459417	0.833900	0.166100	11:19
2	0.445998	0.358876	0.874300	0.125700	11:19

Confusion matrix											
		Actual									
		airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	897	7	21	6	3	3	5	9	32	17	
automobile	4	938	2	3	0	2	1	1	7	42	
bird	25	2	821	37	33	28	33	14	4	3	
cat	6	4	27	782	17	108	31	13	6	6	
deer	8	1	39	38	826	18	33	35	2	0	
dog	2	1	13	131	17	798	6	30	0	2	
frog	3	2	15	45	9	18	906	1	0	1	
horse	7	1	14	21	30	26	2	895	1	3	
ship	25	8	8	2	0	0	3	3	941	10	
truck	12	38	1	6	0	1	1	0	2	939	
Predicted											

Prediction/Actual/Loss/Probability



Fine Tuning 2 {2.10 (lr=0.03) [0.878300]}

Aquí introduïm una nova eina per ajudar-nos a seleccionar, ràpida i òptimament, un valor adequat per la learning rate. Veurem com no és tan útil com ens imaginàvem. Trobem un petit increment en la precisió respecte a l'anterior apropament. En general, les aproximacions que coincideixen amb el mateix learning rate per l'anterior apropament, quan

aquests són baixos, tendeixen a oferir uns valors de precisió menors, però per learning rate alts, l'accuracy millora.

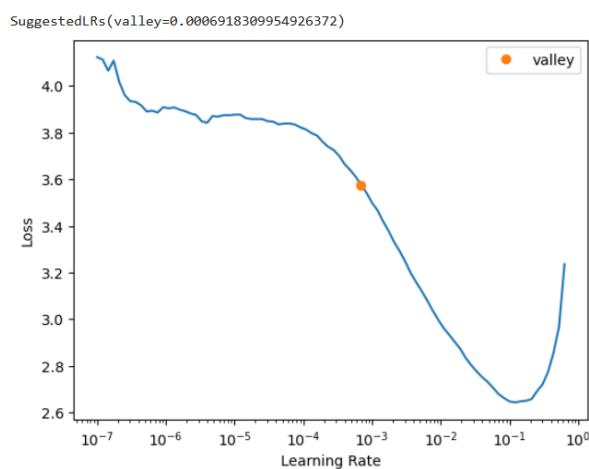
Transformacions en DataBlock i Dataloader

En el DataBlock presenta les mateixes transformacions que en l'anterior apropiament. Pel que fa al DataLoader, definim un batch size de 128 i aquest cop sí que presenta transformacions. Com ja hem comentat, les transformacions fetes al DataLoader són per tractar d'aconseguir data augmentation i reduir l'overfitting.

```
dataloaders(path, bs=128, item_tfms=Resize(460), batch_tfms=aug_transforms(size=224),  
verbose=True)
```

2.0 Suggested learning rate

Aquí s'introduceix l'eina esmentada que, en teoria, ens hauria de permetre trobar una bona estimació de l'òptim valor per la learning rate: `<<lr_find()>>`. Dissenyem un gràfic que evalua la loss o error del model en funció de diferents learning rates, i ens suggereix el seu valor (0.0007).



2.2 ($|r|=0.0007$) [0.790200]

Veiem com el learning rate proposat deixa bastant a desitjar si el comparem amb l'accuracy obtingut pel primer apropament. Però potser és degut a les transformacions fets en el DataBlock i DataLoader, així que continuem fent aproximacions rondant aquest líndar, anant pujant de mica en mica el valor.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.383742	0.892943	0.685500	0.314500	04:24
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.048084	0.697415	0.753600	0.246400	10:47
1	0.851517	0.607755	0.781300	0.218700	10:47
2	0.784380	0.589167	0.790200	0.209800	10:47

		Confusion matrix											
Actual	Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck		
		848	24	23	5	12	5	12	7	44	20	20	
airplane	848	24	23	5	12	5	12	7	44	20	20	20	
automobile	16	901	4	4	1	0	6	3	12	53	53	53	
bird	45	5	668	59	75	37	84	20	6	1	1	1	
cat	7	2	41	632	34	139	93	28	13	11	11	11	
deer	20	1	46	40	705	25	80	78	3	2	2	2	
dog	5	1	28	144	28	717	24	53	0	0	0	0	
frog	2	2	36	48	26	17	866	2	5	2	2	2	
horse	11	1	19	27	63	41	17	813	3	5	5	5	
ship	61	24	5	6	4	0	5	2	870	23	23	23	
truck	20	71	1	6	1	0	4	1	14	882	882	882	
		airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck		

2.7 (lr=0.015) [0.872500]

A mesura que anem creixent el valor de learning rate, l'accuracy augmenta, ens estranya que <<lr_find()>> ens hagi suggerit un valor tan baix.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.259900	0.904091	0.671300	0.328700	05:24
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.750827	0.589325	0.798000	0.202000	13:26
1	0.575047	0.443768	0.843900	0.156100	13:23
2	0.435690	0.372847	0.872500	0.127500	13:24

Confusion matrix										
airplane	6	25	4	3	2	5	8	28	18	
actual	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	901	6	25	4	3	2	5	8	28	18
automobile	4	943	2	3	0	0	2	2	14	30
bird	33	0	817	41	35	19	35	14	5	1
cat	7	2	31	778	22	93	37	17	8	5
deer	11	0	32	30	839	16	31	37	3	1
dog	2	0	20	139	20	779	12	27	0	1
frog	5	1	16	38	10	11	914	2	3	0
horse	9	1	14	25	30	25	4	889	1	2
ship	30	10	4	8	1	0	3	1	934	9
truck	12	40	1	5	2	0	0	4	5	931

2.10 (lr=0.03) [0.878300]

Aquesta és la millor aproximació, amb el learning rate el doble de gran que a l'anterior apropament.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.182958	0.904181	0.670100	0.329900	04:40
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.805295	0.651154	0.769800	0.230200	12:10
1	0.597062	0.473922	0.833400	0.166600	12:09
2	0.434432	0.356805	0.878300	0.121700	12:10

Confusion matrix										
airplane	6	14	6	2	3	6	9	32	15	
actual	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	907	6	14	6	2	3	6	9	32	15
automobile	3	950	2	5	0	1	2	1	7	29
bird	34	1	837	36	27	16	31	15	3	0
cat	10	2	34	776	16	103	31	16	6	6
deer	10	0	36	29	849	24	24	24	4	0
dog	4	0	19	116	20	801	10	30	0	0
frog	3	4	16	38	11	10	918	0	0	0
horse	9	0	19	22	36	24	3	884	0	3
ship	28	12	4	4	1	0	1	2	943	5
truck	11	55	1	4	0	0	1	3	7	918

Prediction/Actual/Loss/Probability



Fine Tuning 3 {3.6 (lr=0.03) [0.869800]}

Igual que en Fine Tuning 2, amb la comparativa de les aproximacions amb Fine Tuning 1 pels mateixos learning rates, quan aquest és petit, trobem que el primer apropament és més encertat, contràriament a quant presenta un valor més elevat.

En general, en termes de precisió, aquest apropiament es comporta pitjor que l'anterior. Però, ja que la diferència és mínima, i que aquest presenta millors pràctiques pel que fa a data augmentation per assegurar un menor overfitting a les dades d'entrenament, hem decidit seleccionar-lo com a millor combinació de transformacions en el DataLoader i DataBlock.

A partir d'ara, si bé pot ser que les transformacions variïn una mica entre els diferents apropiaments i models de transfer learning, en general mantindran aquesta estructura. Així que, rere aquest apropiament, no tornarem a parlar de les transformacions aplicades en les imatges i batch dels DataBlocks i DataLoaders.

Transformacions en DataBlock i Dataloader

Trobem les mateixes transformacions en el DataBlock excepte pel fet que no transformem la imatge redundantment dues vegades.

```
item_tfms=[RandomResizedCrop(224, min_scale=0.5), FlipItem(p=0.5)]
batch_tfms=[*aug_transforms(), Normalize.from_stats(*imagenet_stats)]
```

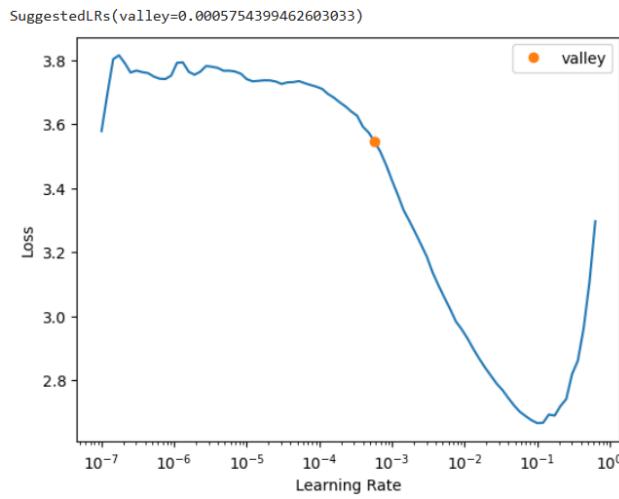
Respecte al DataLoader, ara hem aplicat un gran conjunt de transformacions. Les explicarem amb més detall perquè aquestes seran les que es faran servir en la resta d'apropaments:

```
dls3 = data.dataloaders(path,bs=128,verbose=True,
                         item_tfms=[Resize(224, method='squish'),
                                     FlipItem(p=0.5),
                                     Normalize.from_stats(*imagenet_stats)],
                         batch_tfms=[*aug_transforms(size=224, max_rotate=30.0, min_scale=0.75),
                                     Normalize.from_stats(*imagenet_stats)])
```

- Resize per redimensionar les imatges a mida 224x224; amb mètode 'squish' per aixafar o estirar la imatge segons sigui necessari (recordem que la mida original és de 32x32, així que totes s'estiraran).
- Flip item per bolcar horitzontalment les imatges, amb una probabilitat del 50%,
- Normalitzem per estandarditzar les imatges segons el conjunt de dades d'Imagenet Stats, utilitzant la mitjana i desviació estàndard dels valors de píxels d'Imagenet.
- *aug_transforms per aplicar un conjunt d'augmentació de dades aleatòries a les imatges en batch. Delimitem alguns dels seus paràmetres.
- Per últim renormalitzem de nou per estandarditzar les imatges en batch (creiem que realment no és necessari).

3.0 Suggested learning rate

<<lr_find()>> ens recomana un valor de 0.0006 per la learning rate. Aquest cop no hi confiem gaire i tractem directament amb valors elevats.



3.4 (lr=0.015) [0.867500]

Presenta un accuracy inferior als altres apropaments amb el mateix valor de learning rate.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.349346	0.967819	0.656200	0.343800	04:57
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.840018	0.594187	0.786900	0.213100	12:39
1	0.642810	0.492112	0.827700	0.172300	12:35
2	0.513113	0.386049	0.867500	0.132500	12:32

Confusion matrix										
airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	
Actual										Predicted
airplane	900	11	13	2	8	2	5	6	32	21
automobile	5	948	1	1	0	1	2	2	10	30
bird	36	6	812	25	37	21	43	13	4	3
cat	12	6	37	701	33	108	57	23	10	13
deer	12	0	33	22	851	7	36	37	1	1
dog	3	1	23	117	18	765	16	52	2	3
frog	4	6	18	26	16	8	916	1	3	2
horse	8	2	13	18	32	14	6	902	1	4
ship	28	8	6	0	1	1	3	0	942	11
truck	12	38	1	3	0	0	0	1	7	938

3.6 (lr=0.03) [0.869800]

Aquesta és la millor aproximació. Presenta el mateix learning rate que la millor de l'anterior apropament.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.275452	0.943229	0.659700	0.340300	04:59
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.865219	0.769233	0.723100	0.276900	12:04
1	0.667476	0.526192	0.818200	0.181800	12:02
2	0.489983	0.367354	0.869800	0.130200	12:03

Confusion matrix										
airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	
Actual										Predicted
airplane	877	14	28	9	5	3	7	8	36	13
automobile	3	963	0	1	1	1	1	2	7	21
bird	26	5	823	24	34	24	46	13	2	3
cat	10	4	31	724	21	114	57	22	7	10
deer	7	1	37	29	843	11	28	39	5	0
dog	4	0	21	106	26	793	12	34	0	4
frog	5	2	17	30	11	10	919	2	3	1
horse	9	1	18	17	30	24	6	887	2	6
ship	29	9	5	3	1	0	2	0	937	14
truck	13	40	2	3	0	1	1	2	6	932



Fine tuning 4 (progressive resizing) {[0.908600]}

Aquí introduïm 2 grans canvis que es traduiran en un gran avenç en el rendiment del modelatge i, alhora, ens imaginem, en una millora en l'encert o, com a mínim, en les bones pràctiques d'aquest cas d'estudi.

La primera es tracta de definir un major nombre de treballadors “num_workers=4” en el DataLoader. Això augmentarà molt el paral·lelisme a l'hora de l'entrenament del model, reduint molt notablement els temps d'execució.

La segona es tracta d'establir una estratègia de redimensionament progressiu de les imatges en batch. Aquesta tècnica pot millorar el rendiment del model. Es basa, a grans trets, en fer un entrenament amb les imatges en mida petita; llavors, aprofitar aquest mateix model per reentrenar-lo augmentant la mida de les imatges (sent un procés iteratiu).

Mitjançant l'ús d'aquest redimensionament progressiu, es pot aconseguir que el model aprengui “features” de baix nivell en les imatges petites (com les voreres i la textura), i augmentar el nivell dels features que aprèn (com formes geomètriques) a mesura que s'augmenta la mida de les imatges. Aquesta tècnica, a més de reduir l'overfitting, pot ser indicada per models de transfer learning, aprenent els features de baix nivell en funció de les dades del model aprofitat (no les nostres), creant una bona base sobre la qual sustentar-se el nostre model, i després aprenent els features de més alt nivell en funció de les nostres dades.

Destaquem que aquest apropament actua com a aproximament. No es proven diferents combinacions de paràmetres, sinó que totes les iteracions s'han d'interpretar com un sol model al qual se l'i van modificant una mica certs paràmetres.

Iteració 1 (lr=0.02, bs=128, size=128) [0.895800]

Presenta un learning rate de 0.02, un batch size de 128, i la mida de les imatges en batch és la que hem predefinit per defecte en el dataloader. En aquest cas mostrem per sobre el codi per indicar com hem fet el progressive resizing:

```
learn4 = cnn_learner(dls4, vgg19, metrics=[accuracy, error_rate], pretrained=True)
learn4.fine_tune(epochs=3, base_lr=0.02)
```

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	1.113528	0.883850	0.691900	0.308100	01:42
epoch train_loss valid_loss accuracy error_rate time					
0	0.647079	0.513700	0.823100	0.176900	03:45
1	0.471131	0.384548	0.868900	0.131100	03:46
2	0.334178	0.308315	0.895800	0.104200	03:47

Iteració 2 (lr=0.015, bs=128, size=256) [0.906100]

Aprofitem l'entrenament anterior i el reentrenem reduint una mica el learning rate i augmentant la mida de les imatges. Per això cal redefinir el DataLoader i reintegrar-lo dins el model anterior:

```
dls4 = data_block.dataloaders(path, bs=128, num_workers=4, verbose=True,
    batch_tfms=[*aug_transforms(size=256, max_rotate=30.0, min_scale=0.75, flip_vert=True),
        Normalize.from_stats(*imagenet_stats)],
    item_tfms=[FlipItem(p=0.5),Normalize.from_stats(*imagenet_stats)])
learn4.dls = dls4
learn4.fine_tune(epochs=3, base_lr=0.015)
```

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.359053	0.339493	0.885600	0.114400	01:49
epoch train_loss valid_loss accuracy error_rate time					
0	0.405704	0.523330	0.831500	0.168500	03:48
1	0.323816	0.319958	0.890400	0.109600	03:50
2	0.231966	0.281553	0.906100	0.093900	03:48

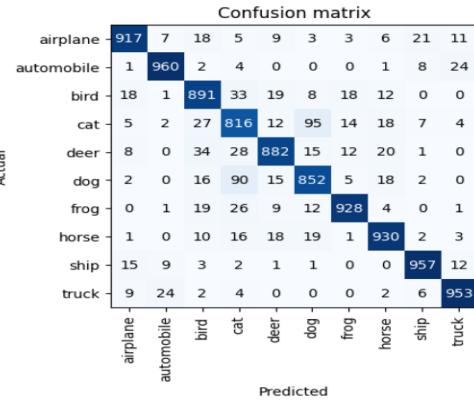
Iteració 3 (lr=0.01, bs=64, size=512) [0.908600]

De la mateixa manera, aprofitem l'entrenament anterior i reentrenem reduint el learning rate, el batch size i augmentant la mida de les imatges:

```
dls4 = data_block.dataloaders(path, bs=64, num_workers=4, verbose=True,
    batch_tfms=[*aug_transforms(size=512, max_rotate=30.0, min_scale=0.75, flip_vert=True),
        Normalize.from_stats(*imagenet_stats)],
    item_tfms=[FlipItem(p=0.5),Normalize.from_stats(*imagenet_stats)])
learn4.dls = dls4
learn4.fine_tune(epochs=3, base_lr=0.01)
```

Reduïm la mida del batch perquè poden augmentar els requisits de memòria RAM a mesura que s'augmenta la mida de les imatges.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.290025	0.337257	0.895800	0.104200	01:57
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.385342	0.420756	0.862000	0.138000	03:52
1	0.320087	0.336791	0.885700	0.114300	03:52
2	0.205468	0.278632	0.908600	0.091400	03:51



Prediction/Actual/Loss/Probability



Param Tuning 2{weight_d=0.0, learning_r=0.02, dropout=0.0, batch_s=128}[0.9000]

A partir d'aquí l'arquitectura dels apropiaments canvia, així com la seva sintaxi. Això és pel fet que finalment s'ha trobat una manera de fer param tuning de manera més automatitzada, entrenant el model d'una manera iterativa per fer un procés similar al que es faria usant param tuning amb "GridSearch". Els paràmetres testejats han estat:

```
'batch_size': [128],
'learning_rate': [0.02, 0.03],
'weight_decay': [0.0, 0.001],
'dropout': [0.0, 0.1]
```

A més de destacar quina és la iteració amb millor resultat, per cadascuna de les 8 iteracions hem compilat el valor mitjà d'accuracy pels diferents paràmetres testejats:

```
when 'learning_rate'=0.02 --> mean(acc)=0.8974; {{{when 'learning_rate'=0.03 --> mean(acc)=0.897925}}}
{{{when 'weight_decay'=0 --> mean(acc)=0.898575}}}; when 'weight_decay'=0 --> mean(acc)=0.89675
{{{when 'dropout'=0 --> mean(acc)=0.89865}}}; when 'dropout'=0.1 --> mean(acc)=0.896675
```

I, com podem sospitar mirant el títol de l'apropament, la millor iteració ha estat {'weight_decay': 0.0, 'learning_rate': 0.02, 'dropout': 0.0, 'batch_size': 128} [0.9000].

Aleshores el que s'ha anat fent és provar diferents combinacions de paràmetres. Tan bon punt ens hem assegurat que hem trobat el valor més òptim per algun d'aquests, el fixem perquè hi hagi menys iteracions i continuem fent joc amb la resta de paràmetres.

Param T 6{weight_d=0.0, learn_r=0.04, freeze_e=2, epochs=7, dropout=0.0, batch_s=256}[0.9156]

Després d'un seguit de param tunings s'ha aconseguit fixar la majoria dels paràmetres al que creiem que és el seu valor òptim. A continuació el set de paràmetres testejats per l'últim param tuning de VGG19, i la mitjana de les precisions obtingudes per cada paràmetre:

```
'batch_size': [256],  
'learning_rate': [0.02, 0.04],  
'weight_decay': [0.0],  
'dropout': [0.0],  
'freeze_epochs': [2],  
'epochs': [3, 5, 7]
```

A continuació la mitjana de les precisions obtingudes per cada paràmetre:

```
learning rate=0.02-->[0.901967] {{learning rate=0.04-->[0.9071]}}  
epochs=3-->[0.8936] epochs=5-->[0.9066] {{epochs=7-->[0.9134]}}
```

TESTING VGG19 7 (progressive resizing) [0.914800]

Hem parlat d'apropaments i aproximacions, combinacions de paràmetres i hiperparàmetres, en definitiva, estratègies per optimitzar el model. Ara és l'hora de posar en pràctica aquests coneixements adquirits sobre com són les nostres dades i l'arquitectura del model que millor les pot esprémer, testejant-lo amb les dades de “test_unlabelled”.

A l'hora de fer el testatge, s'han ajuntat totes les dades d'entrenament i validació, per disposar d'un major volum de dades i que les prediccions siguin tan encertades com sigui possible. Com ja hem comentat, per aconseguir-ho cal canviar el nom de les dades de validació i modificar el DataBlock per canviar el mètode de divisió (entrenament - validació) de “GrandparentSplitter” a “RandomSplitter”, seleccionant una proporció de 0.05 per al conjunt de validació. Per desconeixement de la metodologia necessària a l'inici, les primeres entrades a l'arxiu .ipynb no presenten aquesta bona pràctica.

Es seleccionarà el testeig que que presenti una major taxa d'encerts segons la validació (recordem que els encerts en les prediccions els desconeixem, ja que les imatges no estan etiquetades). Es tracta del TESTING VGG19 7 (progressive resizing) [0.914800].

Utilitzem la tècnica de redimensionament progressiu de les imatges. En la primera iteració s'ha delimitat una mida del batch de 256; una mida d'imatge de 128; 7 èpoques; una ràtio d'aprenentatge de 0.04 i 2 congelaments d'èpoques:

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.868660	0.887070	0.698000	0.302000	01:51
1	0.780898	0.759468	0.731200	0.268800	01:51
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.445494	0.527597	0.829200	0.170800	04:33
1	0.439332	0.554810	0.804400	0.195600	04:33
2	0.360707	0.433123	0.862800	0.137200	04:32
3	0.275173	0.438620	0.862000	0.138000	04:31
4	0.190412	0.344122	0.884800	0.115200	04:30
5	0.105808	0.279105	0.910000	0.090000	04:30
6	0.063870	0.273066	0.911200	0.088800	04:30

Per a la següent iteració introduirem un nou paràmetre, que s'usarà en la majoria dels testatges. Es tracta de “cbs” el qual, gràcies a la llibreria “fastai.callback”, podrem especificar que aturi l’entrenament si la precisió de les èpoques deixa d’augmentar un cert valor. També permet reduir progressivament el valor del learning rate, també segons el mateix argument:

```
from fastai.callback.all import *
dls = dblock.dataloaders(path, bs=128, num_workers=8, verbose=True,
    batch_tfms=[*aug_transforms(size=256, max_rotate=30.0, min_scale=0.75, flip_vert=True),
        Normalize.from_stats(*imagenet_stats)],
    item_tfms=[FlipItem(p=0.5),Normalize.from_stats(*imagenet_stats)])
callbacks = [EarlyStoppingCallback(monitor='accuracy', min_delta=0.01, patience=1),
    ReduceLROnPlateau(monitor='accuracy', factor=0.2, min_delta=1, patience=1)]
learn.dls = dls
learn.fine_tune(epochs=20, base_lr=0.01, wd=0.0, freeze_epochs=1, cbs=callbacks)
```

De la manera que s’ha codificat, l’entrenament s’aturarà si la precisió deixa d’augmentar en un 0.01, i el learning rate es reduirà en un factor de 0.2 si l’accuracy deixa d’augmentar en 1. És evident que pel que respecta al callback de la learning rate, està mal construït, però no rep importància pel fet que l’entrenament s’atura a la segona època. Dins del callback, el paràmetre pactience, delimita el nombre d’èpoques que han de passar abans d’actuar en conseqüència (delimitant patience=3, donaria 3 èpoques de marge perquè el model millors la precisió per sobre del paràmetre establert, abans d’actuar). *Nosaltres no tenim massa paciència*.

Aleshores, com veiem en l’extracte de codi, en la segona i última iteració, augmentem la mida de les imatges a 256 i reduïm la mida del batch a 128. Especifiquem 20 epochs, un learning rate de 0.01 i freeze epochs=1.

```

epoch train_loss valid_loss accuracy error_rate time
0 0.082380 0.352151 0.904800 0.095200 01:49
[0/20 00:00<?] 0.00% [0/20 00:00<?]

epoch train_loss valid_loss accuracy error_rate time
[314/371 03:41<] 84.64% [314/371 03:41<]

epoch train_loss valid_loss accuracy error_rate time
0 0.072682 0.307160 0.913600 0.086400 04:28
1 0.065978 0.323046 0.914800 0.085200 04:28
No improvement since epoch 0: early stopping

```

Confusion matrix										
Actual	airplane	0	1	2	1	0	1	0	5	4
	automobile	252	0	0	0	0	2	0	0	8
	bird	6	1	201	7	8	2	8	1	3
	cat	1	0	4	222	3	28	9	3	0
	deer	1	0	0	7	202	5	4	3	1
	dog	0	0	1	15	1	217	1	2	0
	frog	1	1	4	9	3	6	212	0	0
	horse	2	0	4	4	7	5	0	223	1
	ship	2	0	1	0	1	1	0	0	265
	truck	1	4	0	0	0	0	0	2	237

Prediction/Actual/Loss/Probability



El testing presenta una precisió bastant elevada (0.9148, la millor dels realitzats en VGG19), per la qual cosa esperem que faci prediccions força acurades. Segons la matriu de confusió, on comet més errors és en les següents dues situacions: etiquetar una imatge com a gos, quan és un gat, i a la inversa. Curiosament, tot i que aquestes siguin les situacions en la que més equivocacions ha fet, en la figura dels màxims errors (que mostra els casos en què el model s'ha desviat més en la predicció), no es reflecteix cap d'aquestes dues situacions. Allà on es desvia més en les seves prediccions (en qualitat, no en quantitat, en termes de loss), és quan la imatge és un ocell.

ResNet34

Ja hem explicat l'enfocament anterior, detallant l'estrategia seguida i com s'han plantejat els diferents apropaments i aproximacions. Com s'ha comentat, a partir d'aquest enfocament no s'explicarà amb tant de detall els plantejaments seguits, ens limitarem a mostrar els resultats de les diferents aproximacions i s'intervindrà només quan sigui necessari. El que sí que cal comentar és que, en l'anterior enfocament, s'han comès molts errors i imprecisions en el disseny de l'arquitectura del model i l'estrategia a seguir, males pràctiques que s'ha procurat que no es perpetuin en els següents enfocaments.

El testing del VGG19 presenta una precisió força alta, i hem identificat les seves màximes carències (falla al confondre els gossos amb gats i els gats amb gossos). Seria ideal trobar un model amb una major precisió i que supleixi aquestes carències.

A continuació veurem en seguit totes les aproximacions fetes per trobar la millor combinació de paràmetres, així com l'accuracy trobat en la millor de les iteracions per cada Param Tuning, en el mateix títol, i la mitjana de les precisions pels diferents paràmetres. Veiem, doncs, com passem d'una accuracy de 0.9147 en la millor de les iteracions del primer ParamTuning, a una de 0.9278 en la final.

Param Tuning 1 {weight_d=0.0, learning_r=0.02, freeze_e=1, dropout=0.0, batch_s=256}[0.9147]

Pot ser, el plantejament ideal seria provar múltiples valors per cadascun dels paràmetres, però cada valor afegit en la combinació fa créixer molt el nombre d'iteracions resultant. És per això que hem optat per un apropament més conservador, fent joc primer amb diferents possibilitats pels quals creiem que són els paràmetres més transcendentals: la mida del batch i el learning rate.

```
'batch_size': [128,256],  
'learning_rate': [0.1,0.02,0.03],  
'weight_decay': [0.0],  
'dropout': [0.0],  
'freeze_epochs': [1]
```

A més de destacar en el títol quina és la iteració amb millor resultat, per cadascuna de les 6 iteracions hem compilat el valor mitjà d'accuracy pels diferents paràmetres testejats:

```
learning_rate=0.01-->[0.8743]; {{learning_rate=0.02-->[0.91175]}}; learning_rate=0.03-->[0.90435]  
batch_size=128-->[0.8923]; {{batch_size=256-->[0.9014]}}
```

Param Tuning 2 {weight_d=0.0, learning_r=0.01, freeze_e=1, dropout=0.001, batch_s=256}[0.914800]

La idea (així com estan enfocats la resta de models preentrenats (DenseNet i EfficientNet)) seria que, amb el primer apropament, haguéssim decidit els millors valors per la mida del batch i el learning rate, però en aquest cas no vam testejar el valor batch_size = 512, així que ens cal enfocar l'apropament tenint en compte aquests paràmetres de nou. De totes maneres, com que s'han pogut descartar alguns valors, reduint així el nombre d'iteracions, hem tingut en compte també altres paràmetres com weight decay i dropout.

```
'batch_size': [256,512],  
'learning_rate': [0.01,0.02],  
'weight_decay': [0.0, 0.001],  
'dropout': [0.0,0.001],  
'freeze_epochs': [1]
```

A més de destacar quina és la iteració amb millor resultat, per cadascuna de les 16 iteracions hem compilat el valor mitjà d'accuracy pels diferents paràmetres testejats:

```

learning_rate=0.01-->[0.911675]; {{learning_rate=0.02-->[0.9119625]}};
weight_decay=0.0-->[0.9112125]; {{weight_decay=0.001-->[0.912425]}}
dropout=0.0-->[0.91155]; {{dropout=0.001-->[0.911975]}}
{{batch_size=256-->[0.9133875]}}, batch_size=512-->[0.9101375]

```

Param Tuning 3 {weight_d=0.001, learning_r=0.02, freeze_e=3, dropout=0.001, batch_s=256}[0.9178]

Ara si, ja tenim fixats els valors dels paràmetres de learning rate i batch size; a més tenim coneixement que és millor no definir els valors per defecte de weight decay i dropout (0), així que encaminem l'apropament cap a definir els valors d'aquests últims dos paràmetres i el de freeze epochs.

```

'batch_size': [256],
'learning_rate': [0.02],
'weight_decay': [0.001, 0.01],
'dropout': [0.001, 0.01],
'freeze_epochs': [1, 2, 3]

```

A més de destacar quina és la iteració amb millor resultat, per cadascuna de les 12 iteracions hem compilat el valor mitjà d'accuracy pels diferents paràmetres testejats:

```

{{weight_decay=0.001-->[0.91523]}}, weight_decay=0.01-->[0.91275]
freeze_epochs=1-->[0.911075], freeze_epochs=2-->[0.91425], {{freeze_epochs=3-->[0.91665]}}
{{dropout=0.001-->[0.914083]}}, dropout=0.01-->[0.913883]

```

Param T 4 {weight_d=0.001, learn_r=0.01, freeze_e=3, epochs=7, dropout=0.001, batch_s=256}[0.9278]

Aquest es tracta de l'últim apropament. Veiem com la precisió dels models s'ha anat millorant a mesura que hem anat restringint els paràmetres als diferents valors òptims. Ara és l'hora de trobar un valor ideal pel nombre d'èpoques. Som coneixedors que potser el millor plantejament seria definir el nombre d'èpoques dels primers paràmetres donat que, si més no, és igual o més transcendental que la resta; però hem decidit limitar a 3 les èpoques de quasi tots els entrenaments per evitar temps d'execució massa llargs. Així doncs, al interessar-nos per aquest paràmetre ara, cal recercar alhora, com a mínim, els valors de learning rate de nou.

```

'batch_size': [256],
'learning_rate': [0.01, 0.02],
'weight_decay': [0.001],
'dropout': [0.001],
'freeze_epochs': [3],
'epochs': [3, 5, 7]

```

A més de destacar quina és la iteració amb millor resultat, per cadascuna de les 6 iteracions hem compilat el valor mitjà d'accuracy pels diferents paràmetres testejats:

```

{{learning_rate=0.01-->[0.92453]}}, learning_rate=0.02-->[0.9214]
epochs=3-->[0.9176], epochs=5-->[0.92425], {{epochs=7-->[0.92705]}}

```

TESTING ResNet 5 (progressive resizing) [0.968800]

Se seguirà el mateix procediment seguit en el testing de VGG19. En aquest cas, s'han trobat 2 testatges amb una predicción molt acurada (segons el que creiem mesurant l'accuracy amb el conjunt de validació, representant un 0.05 de la proporció del conglomerat de dades d'entrenament ("train" i "validation")). De manera que introduirem les dues en l'acoblat de models per donar més pes a les prediccions fetes per Resnet34 (encara que ja es construeix l'acoblat donant més pes a les prediccions amb major precisió). Aquí, però, només explicarem el següent: TESTING ResNet 5 (progressive resizing) [0.968800].

Com en tots els testatges, utilitzem la tècnica de redimensionament progressiu, així que aquesta serà l'última vegada que es mencioni. En la primera iteració, els paràmetres usats són: batch_size=256; dropout=0.001; epochs=7; learning_rate=0.03; weight_decay= 0.001; freeze_epochs=3.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.823143	0.669271	0.768400	0.231600	01:48
1	0.484895	0.467176	0.850400	0.149600	01:38
2	0.413432	0.393492	0.856800	0.143200	01:36

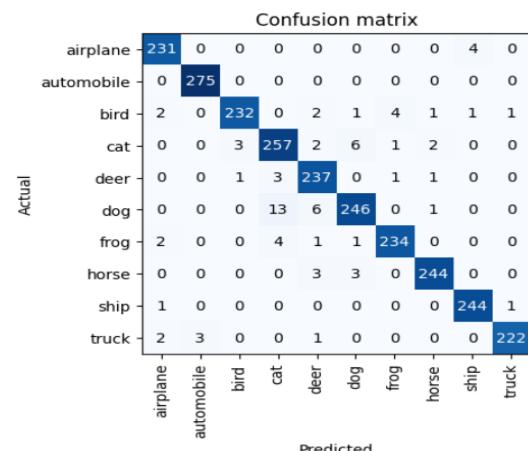
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.430529	0.529658	0.821200	0.178800	01:36
1	0.439346	0.498802	0.828800	0.171200	01:34
2	0.378013	0.452930	0.842400	0.157600	01:34
3	0.275070	0.394217	0.865200	0.134800	01:33
4	0.187151	0.301956	0.895600	0.104400	01:33
5	0.106850	0.249238	0.914400	0.085600	01:33
6	0.070288	0.248632	0.919200	0.080800	01:34

A la segona iteració: batch_size=128; image_size=512; dropout=0.001; epochs=20; learning_rate=0.01; weight_decay= 0.001; freeze_epochs=2; cbs=callbacks.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.081342	0.062436	0.980400	0.019600	01:32
1	0.106437	0.081033	0.974400	0.025600	01:36

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.085689	0.066642	0.976800	0.023200	01:38
1	0.082903	0.086720	0.966400	0.033600	01:39
2	0.085967	0.103386	0.968800	0.031200	01:39

No improvement since epoch 0: early stopping



Prediction/Actual/Loss/Probability



Amb una precisió realment bona, estimem que aquest model haurà fet molt bones prediccions. A partir de la matriu de confusió veiem que hem patit relativament una de les dues situacions problemàtiques de les prediccions fetes amb VGG19: quan la imatge és un gat, ja no el confon tantes vegades amb un gos. Però, encara continua equivocant-se bastant a la inversa. En la figura dels màxims errors, veiem que les equivocacions més greus, qualitativament parlant, s'acostumen a donar quan la imatge és un cérvol o un cavall. La figura que més error té (una granota categoritzada com a un avió, amb una loss de 9.33), presenta menys error que la sisena en el model VGG19 (10.63),

TESTING DenseNet 4 (progressive resizing) [0.972400]

Donat que el plantejament en l'entrenament de la resta de models ha estat exactament el mateix que el fet en l'anterior (ResNet34), i que l'extensió del treball ja comença a ser massa gran, pels propers models (DenseNet i EfficientNet) només detallarem en l'informe 1 dels testatges. Pels curiosos, es podrà veure la totalitat de l'extensió del treball en l'arxiu .ipynb o en el següent enllaç al colab:

"<https://colab.research.google.com/drive/1yvDPvxgvLs1trN0RWmPRMBb2uAIJDIn?usp=sharing>".

En la primera iteració, els paràmetres usats són: `batch_size=256; dropout=0.001; epochs=3; learning_rate=0.03; weight_decay= 0.015; freeze_epochs=3`.

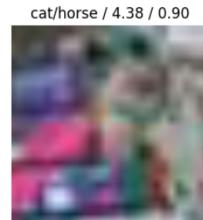
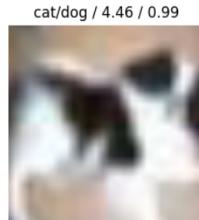
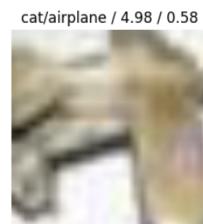
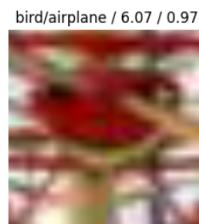
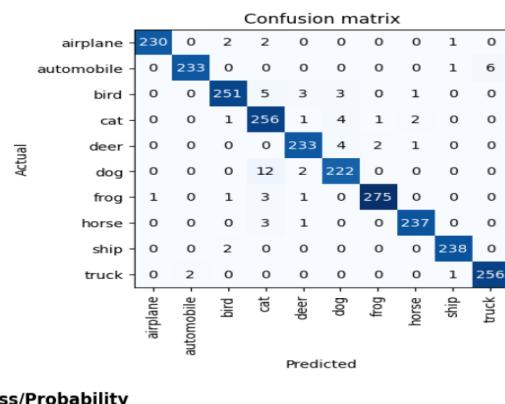
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.648398	0.624292	0.781600	0.218400	02:23
1	0.415518	0.444888	0.850400	0.149600	02:37
2	0.362690	0.405405	0.858000	0.142000	02:30
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.410636	0.645725	0.789200	0.210800	03:01
1	0.288764	0.330387	0.891600	0.108400	02:59
2	0.144195	0.257928	0.908400	0.091600	02:59

A la segona iteració: batch_size=256; image_size=256; dropout=0.001; epochs=20; learning_rate=0.005; weight_decay= 0.015; freeze_epochs=1; cbs=callbacks.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.113630	0.103516	0.965200	0.034800	02:39
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.094192	0.088341	0.969200	0.030800	03:03

epoch	train_loss	valid_loss	accuracy	error_rate	time
1	0.083804	0.083207	0.972400	0.027600	03:01

No improvement since epoch 0: early stopping



Tenim poc a comentar d'aquest model, veient la matriu de confusió podem arribar a la conclusió que és molt similar al de ResNet, sols que una mica millor. Sembla que concentra els seus errors de predicción: apareixen 6 errors en categoritzar un camió quan es tracta d'un cotxe i 5 en un gat quan es tracta d'un ocell; la resta d'errors solen ser molt baixos (en ResNet els errors eren més dispersos). La màxima problemàtica torna a ser la de categoritzar un gat quan realment es tracta d'un gos.

TESTING EfficientNet 4 (progressive resizing) [0.995200]

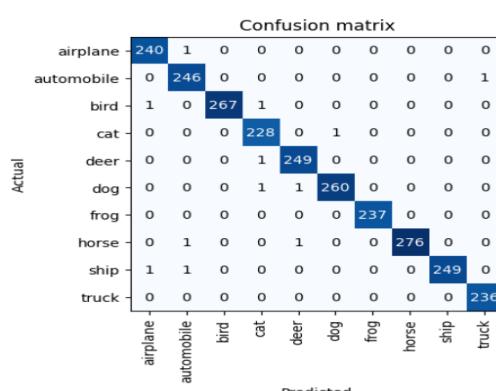
Pel que fa a aquest model, és curiós que en el seu entrenament hem arribat a seleccionar un batch size de fins a 512, però a l'hora del testatge, per algun motiu, donava problemes de requisits de memòria RAM si se seleccionava un batch size major a 64 en la primera iteració. El que és encara més curiós, és que no va haver-hi problemes en redimensionar el batch size a 256 en la segona iteració. Com que es tracta, de llarg, el model amb més precisió de tots, no ens queixarem.

En la primera iteració, els paràmetres usats són: batch_size=64; dropout=0.005; epochs=10; learning_rate=0.005; weight_decay= 0.0; freeze_epochs=3.

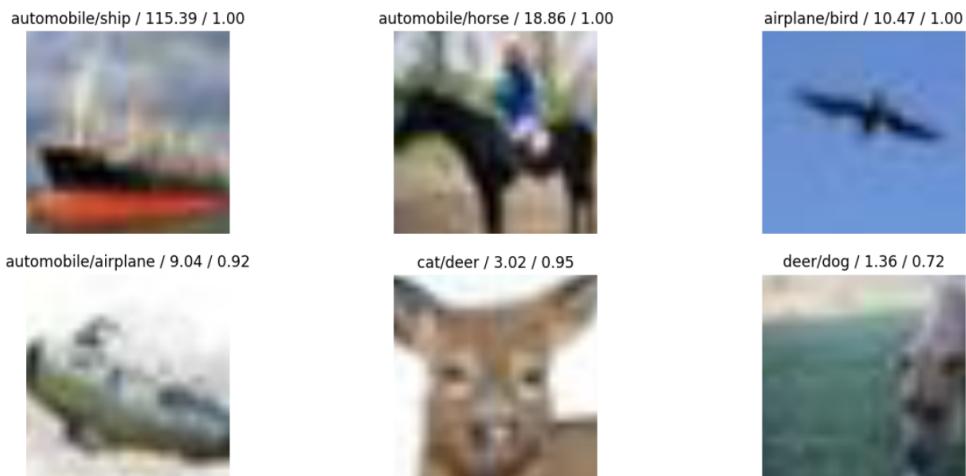
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.915433	0.801592	0.726000	0.274000	02:06
1	0.699633	0.565365	0.820400	0.179600	01:56
2	0.568758	1.474147	0.854800	0.145200	01:52
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.358472	1.619563	0.879200	0.120800	02:09
1	0.354703	0.749890	0.890000	0.110000	02:10
2	0.328110	1.121996	0.894800	0.105200	02:09
3	0.266561	1.374368	0.904000	0.096000	02:07
4	0.212142	3.713680	0.906000	0.094000	02:09
5	0.140982	9.973156	0.894400	0.105600	02:09
6	0.117452	10.485888	0.914800	0.085200	02:07
7	0.062076	7.028413	0.927600	0.072400	02:07
8	0.047958	6.551011	0.930000	0.070000	02:08
9	0.048076	1.017339	0.932400	0.067600	02:08

A la segona iteració: batch_size=256; image_size=256; dropout=0.005; epochs=20; learning_rate=0.01; weight_decay= 0.0; freeze_epochs=1; cbs=callbacks.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.038642	0.101390	0.993600	0.006400	01:32
<hr/>					
epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.037192	0.111092	0.995600	0.004400	01:42
1	0.035223	0.071492	0.995200	0.004800	01:42



Prediction/Actual/Loss/Probability



Serem sincers, no sabem per què aquest model ha estat tan encertat, no hem fet res especial pel que fa a param tunning, res diferent del que hem fet en la resta de models. Deu ser que el conjunt d'imatges amb què s'ha entrenat EfficientNet ha resultat ser molt similar al nostre conjunt d'imatges. Sigui com sigui, tenim un model amb un percentatge de precisió del 99.52%!

Tenint un accuracy tan elevat, no és estranyar que la matriu de confusió no ens serveixi de gaire. Seria una altra cosa si els pocs errors que prengués estiguessin concentrats, però aquests no podrien estar més dispersos.

Com a curiositat, les 4 primeres imatges del gràfic de màxim error, s'allunyen molt més en la seva predicción que pel que fa a la resta de models; fins i tot apareix una imatge amb un error de 115!

Ensamble

Com s'ha comentat al llarg del treball, s'han realitzat diversos testatges per cadascun dels models preentrenats. No entrarem en debat de si es tracta d'una bona pràctica o no; posat que es desconeixen les etiquetes del conjunt “test_unlabelled”, no ens ha semblat que estiguéssim fent “trampeig”.

També s'ha comentat en el “plantejament del problema” que l'ensamble és una combinació de les prediccions fetes per 9 models: 1 de VGG19, 2 de ResNet34, 3 de DenseNet i 3 d'EfficientNet.

S'han seleccionat tots aquells que presenten una precisió major al 95% i, si es donés el cas que cap dels models preentrenats presenta “submodels” amb aquest llindar o major (com VGG19), se seleccionarà 1 per disposar de major variabilitat.

S'ha construït l'acoblat de models de manera que, a l'hora de seleccionar la predicción, reparteixi els pesos ponderats en funció de la precisió del model.

Conclusions

Es fa evident la falta de visió en plantejar l'estratègia a seguir en encarar la ruta per trobar el model amb la millor combinació de paràmetres en VGG19. Igual d'evident que la millora en aquest sentit respecte a la resta de models preentrenats.

Igualment, s'han comès molts errors i males pràctiques. Alguns per desconeixement i d'altres a consciència, com ja s'ha comentat, principalment degut a problemes derivats del fort condicionament de les limitacions d'ús de la GPU del Google Colab.

El model VGG19 tot i això és un bon punt de partida amb un 91,48% de precisió. En destaquem dues problemàtiques: etiquetar una imatge com a gos, quan és un gat, i a la inversa. Problemàtiques que són solucionades pels altres models: ResNet soluciona la primera, i el totpoderós EficientNet la darrera.

Es podria mirar de construir un acoblament de models encara més perfeccionat, no només tenint en compte el pes dels models en funció de la seva precisió, sinó en funció de la matriu de confusió també. Donant així, per exemple, a aquell model que no s'equivoqui en categoritzar un gat, més pes quan aparegui un gat en les seves instàncies.

Disculpeu si l'extensió del treball és massa gran, hem procurat incorporar a l'informe només allò que hem considerat necessari. Per exemple, potser hauria estat bé no fer tan extensa la part de VGG19, ja que ha estat la pitjor encarada i amb pitjor resultat. Però precisament perquè no ha estat ben plantejada, trobem molt interessant documentar els diferents i nombrosos canvis en l'enfocament.

A part de lliurar aquest arxiu en format pdf, el document “.ipynb” on es troba tot l'aspecte pràctic del projecte i l'arxiu “.csv” amb les prediccions finals; es lliurarà també el document “.csv” amb les prediccions dels 9 models seleccionats per poder fer córrer l'ensamble.

Estem desitjant saber quina haurà estat la taxa d'encerts real en el conjunt “test_unlabelled”.