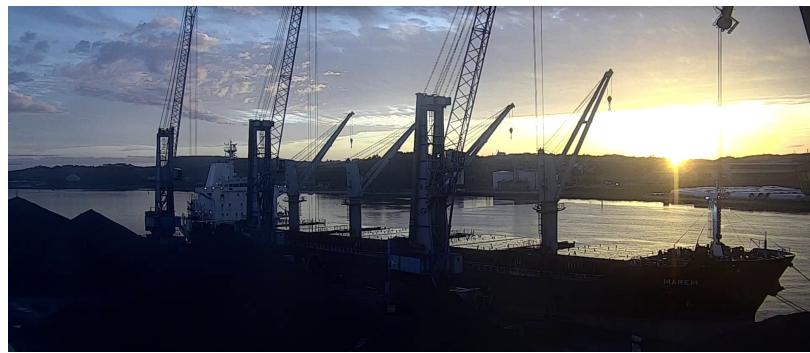


PRÁCTICA II

Pablo Chantada Saborido & José Romero Conde



Índice

1. Introducción	3
1.1. O problema	3
1.2. O noso <i>modus operandi</i>	3
2. Clasificación Ship/No-ship	3
2.1. Modelo base	3
2.2. Adestramento e evaluación	4
3. Clasificación Docked/Undocked	5
3.1. Resultados	6
4. Evaluación dos modelos	6
5. Conclusión	6
6. Apéndice	6

1. Introdución

1.1. O problema

Despois de ler e comprender o enunciado da práctica, e de examinar as imaxes unha por unha, chegamos ás seguintes observacións:

- Entre as imaxes de barcos existe unha gran variación de pose, escala, condicións de iluminación, proporción que ocupa na escena real (en metros) e proporción que ocupa na imaxen (píxeles). Ademáis os elementos en sí (barcos e peiraos) presentantan moita variación interclase (aínda que os barcos sexan todos industriais son distintos entre eles, e os peiraos poden estar baleiros ou cheos de montañas de area) e incluso unha maior similitude intraclase (sobretodo no caso barco-atracado contra barco-non-atracado).
- O alcance da práctica de esta asignatura era suficientemente limitado como para resolver o problema dun xeito satisfactorio. Creemos que poderíamos entrenar unha Rede de Neuronas Artificiais para resolver un subproblema ou un problema maior con fin de que a Rede principal o teña más doado para aprender. No obstante non é o que se pide entón aínda que preferiríamos resolver o problema dunha forma *mais científica* por cuestiós de tempo e alcance non o fixemos.
- En calqueira escenario (sexa unha práctica con fins docentes ou un traballo no mundo real) existe un problema co cómputo, no noso caso porque disponemos de poucos recursos para adestrar a Rede e no caso real porque o cálculo é posible que precise ser ou a tempo real ou nun sistema embebido, en calqueira caso recursos límitados.

Estas observacións dirixiron o noso *modus operandi*.

1.2. O noso *modus operandi*

Aínda que teñamos cursadas asignaturas de aprendizaxe non somos, en ningún caso, expertos no adestramento de Redes de Neuronas Artificiais. Por esta razón preferimos ser cautos e explorar maitas opcións, todas as que podamos, aínda que o noso coñecemento pode (e debería) guiar esta búsqueda, tomamos a decisión de non tomar poucas decisións ó principio, e que fora a evidencia a que avale as conclusiós. Por esto en vez de faceren unha Red estática, a fixemos altamente flexible. Referímonos a que fixemos un *script* que recibe como argumento unha gran cantidade de hiperparámetros, de xeito que sexa máis cómodo probar. Despois de algunas probas (e acompañadas de lecturas da literatura) chegamos ó noso *baseline*.

2. Clasificación Ship/No-ship

2.1. Modelo base

Describimos agora o noso modelo *baseline*, que valeunos para iterar y comparar resultados. O modelo componse dos seguintes elementos:

- **EfficientNet.** [4] Usámola como CNN (*Convolutional Neural Network*) de partida. Orixinalmente pensamos que tiña como requirimiento que as imaxes sexan de tamaño (244, 244, 3). E como as imaxes son (i) rectangulares (ii) de distinto tamaño entre si; tivemos que implementar un recortador automático de *o cadrado más grande* que chamamos desde os **DataLoader** de adestramento e proba. Máis tarde decatámosnos que a Rede era realmente capaz de procesar imaxes de calqueira tamaño porque ten ó final un **Global Pooling**. No obstante, como de

calqueira xeito os *mini-batches* tiñan que ser todos del mesmo tamaño (paralelepípedos) decidimos quedarnos coa idea de recortalas imaxes da forma mencionada.

Máis tarde, non obstante probamos con outros tamaños para o paralelepípedo, coa esperanza de que imaxes rectangulares ofrezcan mellores resultados (en vez de as cadradas) porque estas teñen máis información. Para a nosa sorpresa a evidencia demostrou moito mellor rendimento con imaxes cadradas. Hipotizamos que débese a que como da mesma imaxe rectangular, os cadrados resultantes do resultado aleatorio son realmente distintos entre sí, recortar rectángulos en cadrados pode supor un bo aumento de datos implícito.

En xeral a motivación para escoller esta rede e non outra débese á terceira observacións da sección 1.1 e á forma que tiveron os autores da Rede de atacar este problema directamente. Con respecto a que EfficientNet en contrexto usamos: despóis de probar ca EfficientNetB3 e observar malos resultados decidimos mudar á EfficientNetB4 e nos quedamos nesta e non subimos á EfficientNetBi $|i \in \{5, 6, 7\}$ precisamente por limitacións de cómputo.

- **Aumento de datos.** O aumento de datos nesta fase esencialmente consta de dúas partes, por un lado, as transformacións de `torchvision.transforms` e por outro imaxes recortadas a man. Con respecto ó primeiro, nosotros usamos: volteos horizontais, fluctuacións leves na cor, tamén leves transformacións afíns e conversión a branco e negro con baixa probabilidade. O segundo punto é que recortamos manualmente as imaxes de barcos de forma que queden centrados e sen fondo que estorbe, engadíronse estas imaxes ó conxunto de datos cando especificabase aumento de datos. Desta forma conseguimos dúas cousas distintas:

- Representar máis variacións que as presentes no conjunto de adestramento, ca esperanza de que se sí ocurriesen no conxunto de proba, estaríamos preparados. Por exemplo, como observamos imaxes de noite ou con chuvia encontramos razonable incluir a conversión a branco e negro.
- Representar con menos ruído adicional o que sí queremos aprender. Porque áinda que é beneficioso para a Rede someterse a moita variación, tamén dificulta o adestramento (é un problema máis difícil).

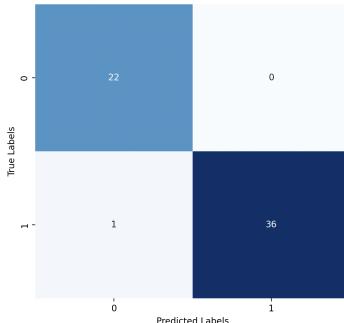
Sobre o aumento de datos comentar que a relación dos hiperparámetros que o definen e problemática para nos faceren probas: se cambiamos varios de unha vez non está necesariamente claro a cal dos cambiados débese a mellora ou empeora do rendimiento; se, en cambio, cambiamos só un de cada vez, a búsqueda no espacio de hiperparámetros do aumento de datos vólvese prohibitiva. E por isto que sabemos que a nosa configuración debe ser subóptima, pero non temos nada para evitalo.

- Un **MLP** (Perceptrón Multicapa ou *Multi-layer Perceptron* en inglés) de tres capas, sobre a saída da Rede. A saída da última capa `nn.Softmax()` ten 2 o 3 neuronas según se quería clasificar en {no barco, barco} o en {no barco, barco no amarrado, barco amarrado}. Tamén probamos con un perceptrón simple (unha capa) pero a evidencia demostrou que os resultados eran mellores con tres. Como o número de neuronas na primeira capa (a saída da CNN) era máis grande ca o número de exemplos, a rede podería ter aprendido unha solución trivial co conxunto de adestramento e non xeneralizar en absoluto. Por esto mesmo decantamos por usar a ben coñecida técnica de regularización *DropOut* [2].

2.2. Adestramento e evaluación

O adestramento foi difícil e ademáis somos inexpertos porén indentificamos algúns problemas para os que atopamos solución. Aquí preséntanse:

- **Datos non balanceados.** Observamos que a Rede, *hackeando* as nosas métricas decidía que todas as imaxes eran barcos. Esto débese ó desbalanceo dos datos (147 contra 88) dónde unha solución tan trivial e erronea como solo predecir unha clase non estaba tan penalizado con metricas como a precisión. Afrontamos isto alterando a probabilidade de ocorencia das clases de forma que se antes, ó escolleren unha imaxe de `DataLoader`, $p(X = \text{barco}) = \frac{147}{147+88}$ agora $p(X = \text{barco}) = 0,5$. Para ello usamos a utilidad `WeightedRandomSampler` de torch.
- **Similitud nos datos.** Non creemos conveniente que a Rede esté totalmente segura de que un barco non está (porque igual é pequeno ou está ocluído) cando ela dí que non hai barco na imaxe. Por isto, e para que xeneralice millor adoptamos a idea de [3] donde as etiquetas en vez de ser $[0, 1]$ para non-barco e $[1, 0]$ para barco, son $[\epsilon, 1 - \epsilon]$ e $[1 - \epsilon, \epsilon]$ respectivamente, onde fixamos $\epsilon = 0,1$.
- **Capacidade do modelo.** Orixinalmente usamos a EfficientNetB0, pero apreciando malos resultados decidimos mudar á EfficientNetB3 e logo á EfficientNetB4, que é un pouco máis grande pero precisamente como narran no artícuilo, é eficiente. Probamos con outras alternativas como as clásicas ResNet e VGGNet sen conseguiren melloras nos resultados que pagasen a pena a diferencia no consumo de recursos.
- **Mínimos locales.** Para aliviar as consecuencias da dificultade do adestramento implementamos un `learning_rate` que, con unha certa paciencia, mediaba a sua valor.¹
- **Sobreajuste y xeneralización.** Para cercionarnos de que a Rede aprendía correctamente, añadimos una penalización de la norma euclídea dos parámetros na función de perdida, es decir, regularización Tikhonov. Atopar valores axeitados para λ foi especialmente difícil.



Outras consideracións sobre o adestramento que consideramos menos profundas y más ordinarias son: como optimizador hemos usado AdamW[1] con un `learning_rate` de 10^{-3} ou 10^{-4} . Usamos un `batch_size` de 16, que aunque es relativamente pequeño, no mucho más pueden hacer nuestros portátiles. Por fortuna un de nos disponía a tiempo parcial de unha NVIDIA 4070 cedida pola Universidade e nela podíamos usar ata 64 imaxes á vez. Aínda así o tempo máquina reconocemos foi un cuello de botella.

Con respecto ós resultados obtidos nesta primeira iteración, poden observarse a grandes rasgos na matriz de confusión da esquerda, a cal nosotros prudentemente podemos dicir que estamos orgullosos porque as predicciones da Rede concentransen nas etiquetas verdaderas, errando soamente [DEPENDE DE LA IMAGEN USADA].

Figuras de las curvas de pérdida y precisión en aprendizaje o de las etiquetas predichas por la red para algunas imágenes en concreto pueden encontrarse para este ejercicio y el siguiente en el Apéndice al final del documento.

3. Clasificación Docked/Undocked

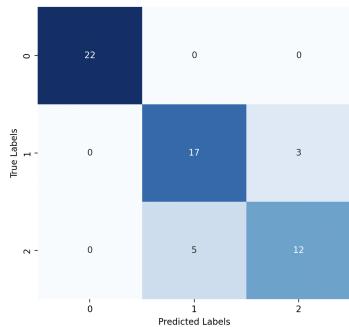
Nós concebimos este exercicio como unha extensión do anterior, deste xeito pensamos en reutilizar todas as representacións aprendidas.

- Cuando especificase `pretrained=True`, a nosa Rede non toma os parámetros de Imagenet, senón do exercicio anterior. Deste xeito xa ten moito aprendido e o proceso de optimización será más doado.

¹Esta paciencia mencionada e a do `earlystopping` poden ambas ser axustadas como `flags` no noso `script`.

- Como agora a cabeza da Rede teu 3 neuronas pero antes tiña 2, en vez de inicializar aleatoriamente las tres neuronas, dous copianse e só unha inicializase aleatoriamente. De novo, deste xeito ten menos traballo por diante.

3.1. Resultados



Ainda que a priori semella que son peores resultados que no exercicio 1, queremos señalar las siguientes observaciones:

- Agora o problema orixinal (barco contra no barco) está totalmente resolto (no barco tiene la etiqueta de 0). A isto encontramoslle a seguinte intuición: que a Rede se esforce nun problema máis complexo fai que sexa máis capaz de resolver o problema sinxelo.
- As diferenzas entre as clases atracado contra no atracado son realmente sutiles [FOTO] e a un humano (nós probamos con amigos) que non foi *adestrado* cas etiquetas tamén pode resultarlle complexo resolver.
- Non obstante en xeneral poderíamos dicir que malia non seren a clasificación perfecta, polo menos é bastante diagonal.

4. Evaluación dos modelos

Na tabla a continuación mostránse os resultados da precisión das prediccións da Rede cos datos de proba, a separación de datos adestramento e proba foi dun 80 %

Non preentrenado		Preentrenado	
		Non aumento de datos	Aumento de datos
2 clases			
3 clases			

5. Conclusión

Agradecemos que o conxunto de datos desta práctica fora realista porque intentando obter mellores resultados tivemos que probar e aprender moitas cousas. Creemos, non obstante, que se tivesemos que resolver este problema nun contexto real, debería intentarse algunha aproximación más centrada en visión e non tan *darlle os datos á rede e que ela aprenda*. Ou incluso usar arquitecturas máis sofisticadas que intenten resolver un problema máis complexo como segmentación coa esperanza de que o rendimento na tarefa máis sinxela de clasificación véxase mellorado.

6. Apéndice

Options

- h, --help** Show this help message and exit
- root_dir ROOT_DIR** Root directory containing the image folders

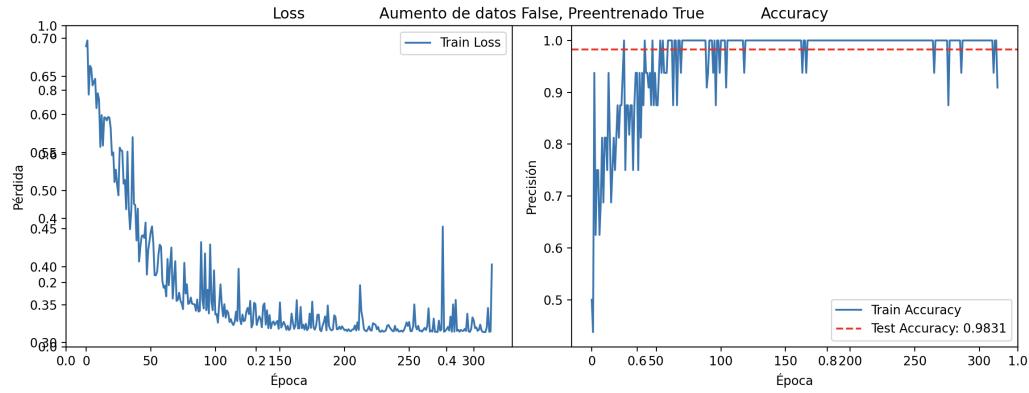


Figura 1: Loss y Accuracy para el primer ejercicio



Figura 2: Imágenes con su etiqueta correcta y el vector SoftMax predicha por la red para el primer ejercicio

- `--data_augmentation` Apply data augmentation and include cropped ship images
- `--docked` Include docked status in labels
- `--not_train` Not train the model, used in conjunction with `--test_images`
- `--train_ratio TRAIN_RATIO` Ratio of training data to total data
- `--unbalanced` Unless specified, class balancing will be applied
- `--pretrained` Use pretrained weights for the model
- `--mlp_head` Use a 2-layer MLP in the head of the model
- `--model_path MODEL_PATH` Path to save or load the model
- `--load_model` Load a pretrained model instead of training
- `--arquitecture ARQUITECTURE` Model architecture, default is `efficientnet_b4`.

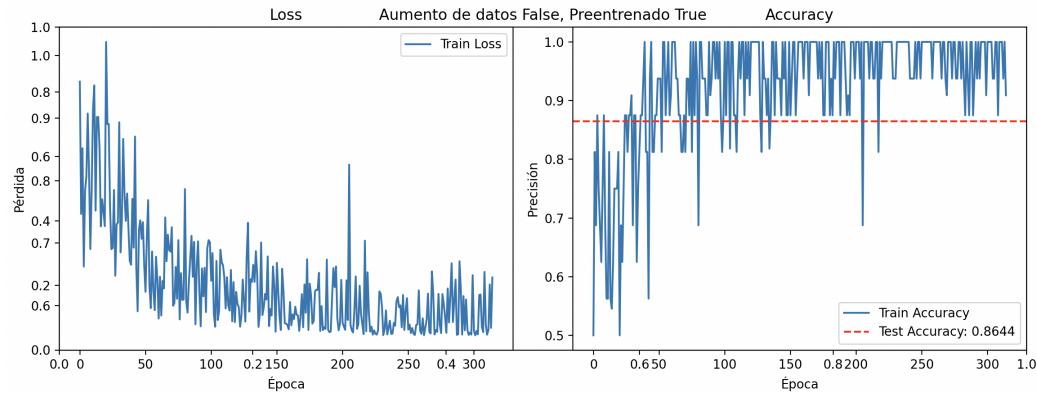


Figura 3: Loss y Accuracy para el segundo ejercicio



Figura 4: Imágenes con su etiqueta correcta y el vector SoftMax predicha por la red para el segundo ejercicio

■ --batch_size BATCH_SIZE	Batch size for training and testing
■ --num_workers NUM_WORKERS	Number of workers for data loading
■ --num_epochs NUM_EPOCHS	Number of epochs for training
■ --patience PATIENCE	Patience for early stopping
■ --lr_patience LR_PATIENCE	Patience for reducing learning rate
■ --learning_rate LEARNING_RATE	Learning rate for optimizer
■ --l2_lambda L2_LAMBDA	Lambda for L2 weight decay regularization
■ --show	Show figures instead of saving them
■ --device DEVICE	Device, default is mps
■ --test_images TEST_IMAGES [TEST_IMAGES ...]	List of image paths to test individually

Referencias

- [1] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.
- [2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
- [4] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.