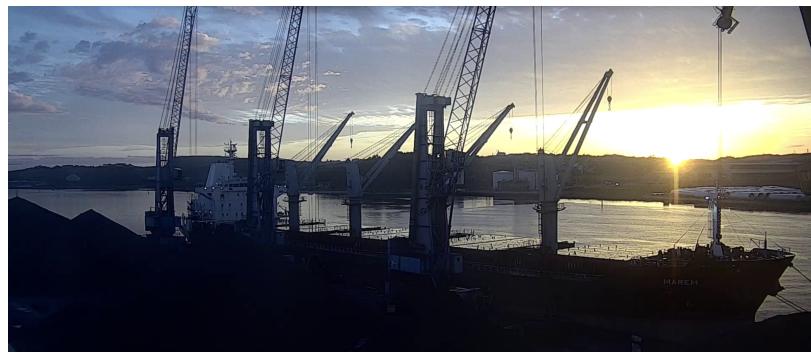


# PRÁCTICA II

Pablo Chantada Saborido & José Romero Conde

---



## Índice

<b>1. Introducción</b>	<b>3</b>
1.1. O problema . . . . .	3
1.2. O noso <i>modus operandi</i> . . . . .	3
<b>2. Clasificación Ship/No-ship</b>	<b>3</b>
2.1. Modelo base . . . . .	3
2.2. Adestramento e evaluación . . . . .	5
<b>3. Clasificación Docked/Undocked</b>	<b>7</b>
<b>4. Evaluación dos modelos</b>	<b>8</b>
<b>5. Conclusión</b>	<b>8</b>
<b>6. Apéndice A: figuras</b>	<b>9</b>
6.1. Primeiro exercicio . . . . .	9
6.1.1. Sen Aumento de datos e sen Preentrenado . . . . .	9
6.1.2. Sen Aumento de datos e con Preentrenado . . . . .	9
6.1.3. Con Aumento de datos e sen Preentrenado . . . . .	10
6.1.4. Con Aumento de datos e con Preentrenado . . . . .	10
6.2. Segundo exercicio . . . . .	11
6.2.1. Sen Aumento de datos e sen Preentrenado . . . . .	11
6.2.2. Sen Aumento de datos e con Preentrenado . . . . .	11
6.2.3. Con Aumento de datos e sen Preentrenado . . . . .	12
6.2.4. Con Aumento de datos e con Preentrenado . . . . .	12
<b>7. Apéndice B: como executar o noso código</b>	<b>13</b>

## 1. Introdución

### 1.1. O problema

Despois de analizar 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) presentan 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 desta asignatura era suficientemente limitado como para resolver o problema dun xeito satisfactorio. Creemos que poderíamos entrenar unha Rede de Neuronas Artificiais (RNA) para resolver un subproblema o un problema maior con fin de que a Rede principal o teña máis doado para aprender. No obstante non é o que se pide, aínda que preferiríamos resolver o problema dunha forma *máis científica* (en vez de simplemente darlle os datos á rede e que ela aprenda) por cuestións de tempo e alcance non foi posibel.
- En calqueira escenario (sexá 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*

Na hora da creación do *baseline* preferimos ser cautos e explorar moitas opcións. 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 algúns 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 evaluamos para iterar e comparar resultados. O modelo compónse dos seguintes elementos:

- **EfficientNet.** [5] 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*<sup>1</sup> que chamamos desde os **DataLoader** de adestramento e proba. Máis tarde decatámonos 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 do mesmo tamaño decidimos quedarnos ca

<sup>1</sup>Este operador recorta a imaxe rectangular nun cadrado tan alto como a imaxe e centrado aleatoriamente. De xeito que se se chama varias veces sobre a mesma imaxe, producirá distintos cadrados cada vez. Por tanto é unha regularización na que a nosa hipótesis é: para clasificar a imaxe rectangular, cun cadrado basta.

idea de recortalas imaxes da forma mencionada.

En iteracións posteriores, probamos con outros tamaños, coa esperanza de que imaxes rectangulares ofrezcan mellores resultados (en vez das 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. A continuación pódense ver as curvas de pérdida e precisión dos dous adestramentos. Apreciamos cas imaxes cadradas que o adestramento levou máis tempo e non converxe tan rápidamente, foi *máis difícil* pero a cambio xeneraliza moito mellor (a liña discontinua mostra a precisión en proba).<sup>2</sup>

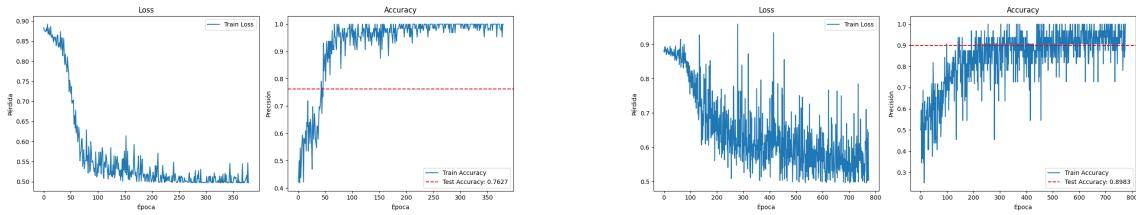


Figura 1: Comparativa de adestramento con imaxes rectangulares (esquerda) e cadradas (dereita)

En xeral a motivación para escoller esta rede e non outra débese á terceira observación 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, usamos: volteos horizontais, fluctuacións leves na cor, tamén leves transformacións afíns, conversión a branco e negro e emborronado Gausiano. 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 e clasificación en dúas clases (non consideramos adecuado incluir estas imaxes na clasificación de tres clases porque ahí é importante o contexto do barco). As transformacións pódense apreciar na figura 2. Desta forma conseguimos dúas cousas distintas:

- Representar máis variacións das presentes no conxunto 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

<sup>2</sup>As condicións do adestramento que se mostra foron: sen aumento de datos e sen preentrenado, pero probamos con todas as combinación e a conclusión e consistente.

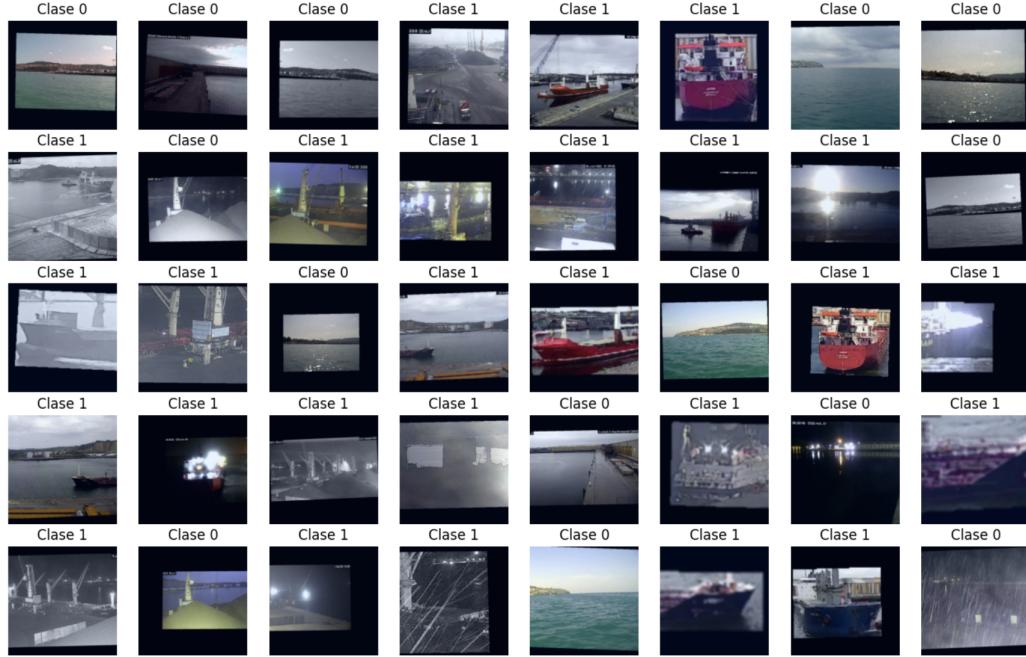


Figura 2: Imaxes cas transformacións do aumento de datos

vólvese prohibitiva<sup>3</sup>. E por isto que sabemos que a nosa configuración debe ser subóptima, pero non temos nada para evitalo.

- Un **MLP** (*Multi-layer Perceptron*) de tres capas, sobre a saída da Rede. A saída da última capa `nn.Softmax()` ten 2 ou 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<sup>4</sup>. 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 *DropOut* [3] en diversas zonas da Rede.

## 2.2. Adestramento e evaluación

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

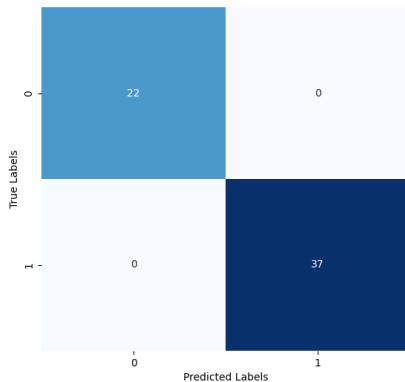
- **Datos non balanceados.** Observamos que nunha primeira instancia a Rede marcaba todas as imaxes coma barcos. Esto débese a que non estaba tan penalizado na función de pérdida debido ó desbalanceo dos datos (147 contra 88) dónde unha solución tan trivial e erronea como solo predecir unha clase non estaba tan penalizado con metricas como a precisión. Afrontamos isto alterando a probabilidade de ocurrencia 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 elo usamos a utilidad `WeightedRandomSampler` de torch.

<sup>3</sup>Se un pásase cas fluctuacións de cor, non conseguirá imaxes útiles nin converxirá a Rede, se un quedase corto será como non ter feito nada. As modificacións que atopamos axeitadas foron guiadas pola nosa (non entrenada) intuición e con probas.

<sup>4</sup>Estos resultados poden reproducirse executando o `main.py` con e sin `--mlp_head` (se non se especifica non se engaden capas).

- **Similitud nos datos.** Non creemos conveniente que a Rede esté totalmente segura de que un barco non está (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 [4] onde 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 artículo [5], é eficiente. Probamos con outras alternativas como ResNet e VGGNet sen conseguiren melloras nos resultados que pagasen a pena a diferencia no consumo de recursos.
- **Mínimos locais.** Para aliviar as consecuencias da dificultade do adestramento implementamos un `learning_rate` que, con unha certa paciencia, mediaba a sua valor.<sup>5</sup>
- **Sobreajuste e xeneralización.** Para cercionarnos de que a Rede aprendía correctamente, engadimos unha penalización da norma euclídea dos parámetros na función de perdida, é dicir, regularización Tikhonov [1]. Atopar valores axeitados para  $\lambda$  foi especialmente difícil.

Outras consideracións sobre o adestramento que consideramos menos profundas e más ordinarias son: como optimizador usamos AdamW [2] con un `learning_rate` de  $10^{-3}$  ou  $10^{-4}$ . Usamos un `batch_size` de 32.



Con respecto ós resultados obtidos nesta primeira iteración, poden observarse a grandes rasgos na matriz de confusión da esquerda. A precisión no conxunto de datos de proba é perfecta. Creemos, non obstante que non é esta a única métrica que un pode evaluar. Figuras das curvas de pérdida e precisión en aprendizaxe ou das etiquetas predichas pola Rede para algunhas imágenes en concreto<sup>6</sup> poden atoparse para este exercicio e o seguinte no Apéndice ó final del documento.

<sup>5</sup>Esta paciencia mencionada e a do *earlystopping* poden ambas ser axustadas como *flags* no noso *script*.

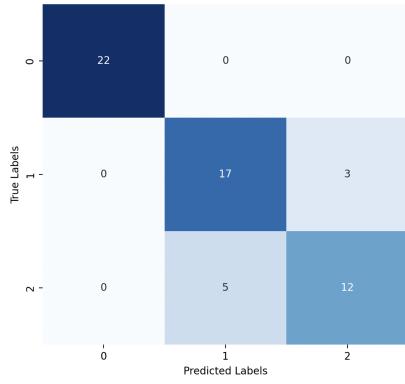
<sup>6</sup>Atopamos especialmente útiles estas figuras xa que podíamos ver que imaxes levaban a Rede ó fracaso durante as iteracións. Deste xeito, podíamos *debuggear* un problema de visión e non simplemente un adestramento de Redes Neuronais calquera.

### 3. Clasificación Docked/Undocked

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

- Cando especificase `pretrained=True`, a nosa Rede non toma os parámetros de Imagenet (como tiñamos feito ate agora, xa que é o defecto ó descargar parámetros para a Rede), senón do exercicio anterior. Deste xeito xa ten moito aprendido e o proceso de optimización será más doado.
- Como agora a cabeza da Rede ten 3 neuronas pero antes tiña 2, en vez de inicializar aleatoriamente as tres neuronas, dous copianse e só unha inicializase aleatoriamente. De novo, deste xeito ten menos traballo por diante. De calqueira xeito esperaríamos que a rede mude moito os seus parámetros durante o adestramento e esta inicialización soamente é unha inicialización afortunada, pero por suposto é preciso adestrar a Rede (con moitas épocas).

Ainda que a priori semella que son peores resultados que no exercicio 1(agora non conseguimos a perfectitude), queremos sinalar as seguintes observacións:



- Agora o problema orixinal (barco contra no barco) está totalmente resolto <sup>7</sup>. 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 (ver Figura 3) e a un humano 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, e os errores concentránse onde intuitivamente pensaríamos, e dicir, entre as clases *barco atracado* e *barco non atracado*.



Figura 3: Comparativa dunha imaxe do barco amarrado (esquerda) e no mar (dereita)

<sup>7</sup>Nas figuras do anexo pódese ver como, se úsasemos o segundo exercicio para resolver o primeiro (xuntar as últimas duas columnas e filas) tendríamos mellores resultados que se intentásemos resolver o primeiro problema directamente.

## 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	Non aumento de datos	Aumento de datos
2 clases	94 %	84 %	100 %	98 %
3 clases	76 %	67 %	86 %	83 %

Sobre estos resultados podense facer as seguintes reflexións:

- Que o modelo esté preentrenado ou non supón unha gran diferencia, sobretodo no caso de aumento de datos. Esto é loxico e esperable. Que ocurra máis claramente no caso de aumento de datos xustifícase con que: ó seren un problema máis difícil, vai ser maior a axuda que supón ter os parámetros preentrenados.
- O aumento de datos sempre ofrece peores resultados que non facelo. Esto pode parecer alarmante, e indicador de que fixemos un mal aumento de datos. Pero hai varias cousas que matizar. En primeiro lugar, como en calqueira caso as imaxes rectangulares recortanse a cadrados, sempre hai algún tipo de aumento de datos, entón na táboa realmente estase a mostrar a comparativa de aumento de datos ou moito aumento de datos. Ademais é posible que con máis épocas a Rede seguisse adestrándose ata o punto de que con aumento de datos exista unha mellora. Ó final o obxectivo do aumento de datos e impoñer restriccións sobre o que debe aprender ca esperanza de que xeneralice mellor; por tanto, ó enfrentar á Rede a un problema maior, que lle custe máis é esperable. En calqueira caso vemos que no problema de clasificación en dúas clases non fai falta aumento de datos.
- O problema de 3 clases ofrece un rendemento sempre peor, o cal é esperable por dous motivos: (i) son máis clases (ii) hai unha gran similitude entre as clases *barco atracado* e *barco non atracado*.
- Para a combinación 3 clases, non preentrenado, aumento de datos, por ser 3 clases e ter un 67 % de precisión en test podría parecer que a Rede chegou a unha solución trivial (asignar a todas as imaxes de barcos a mesma clase e de ahí a métrica) pero observando as figuras do Anexo pódese ver que non, que a Rede esta cando menos encamiñada (a matriz de confusión segue sendo diagonal malia moito error nas duas clases con barco).
- Para a combinación 2 clases, preentrenado, non aumento de datos, podría parecer que a Rede sobreentrena, e de ahí a perfectitude, más temos que recordar que as métricas son no conxunto de datos de proba.

## 5. Conclusión

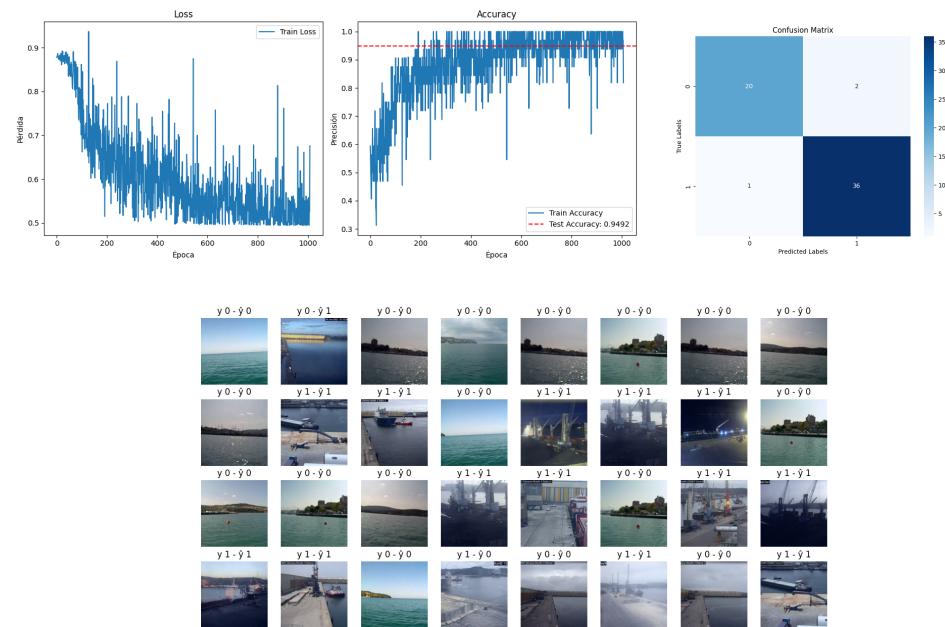
Agradecemos que o conxunto de datos desta práctica fora realista porque intentando obter mellores resultados tivemos que probar e aprender moitas cousas. Creemos, non obstante, que se tivesemos que resolver este problema nun contexto real, debería intentarse algunha aproximación más centrada en visión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

Esperamos que esta práctica sexa tan amena de corregir como foi de facer.

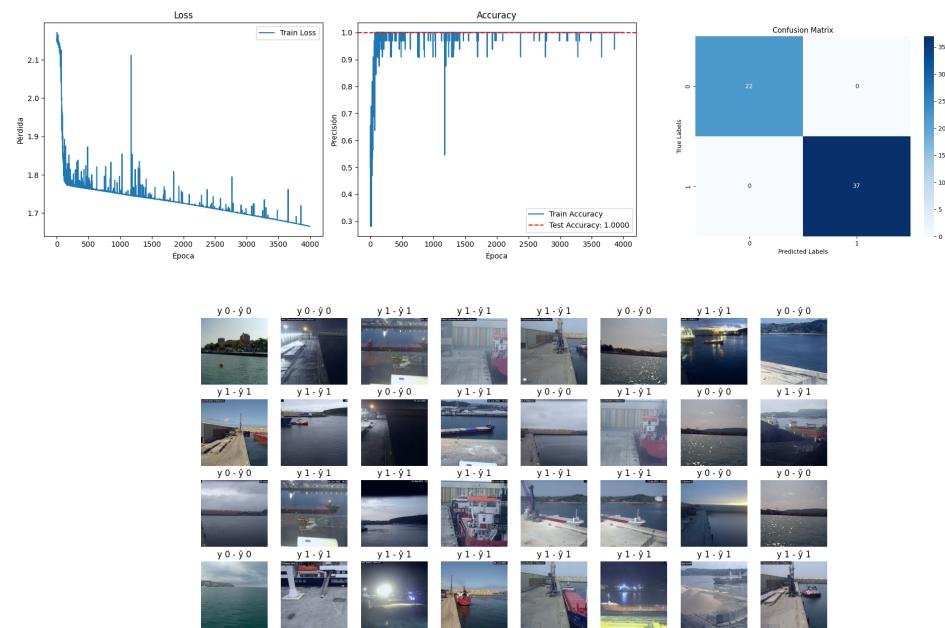
## 6. Apéndice A: figuras

### 6.1. Primeiro exercicio

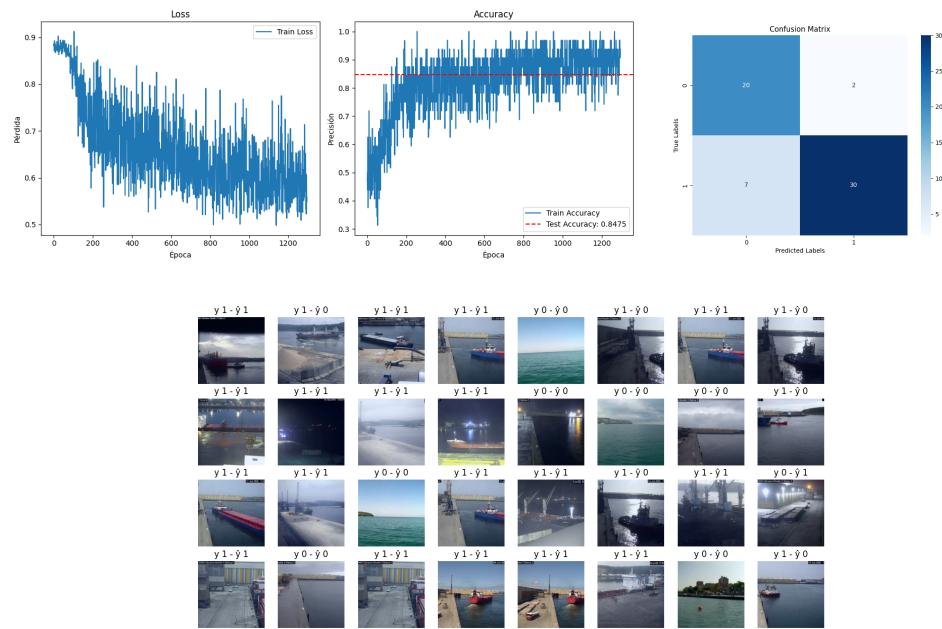
#### 6.1.1. Sen Aumento de datos e sen Preentrenado



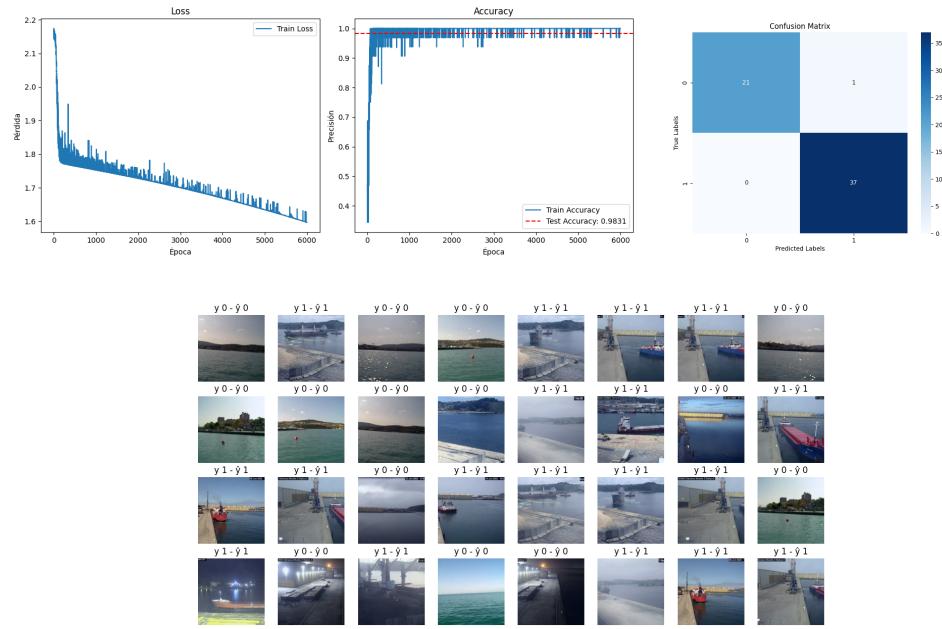
#### 6.1.2. Sen Aumento de datos e con Preentrenado



### 6.1.3. Con Aumento de datos e sen Preentrenado

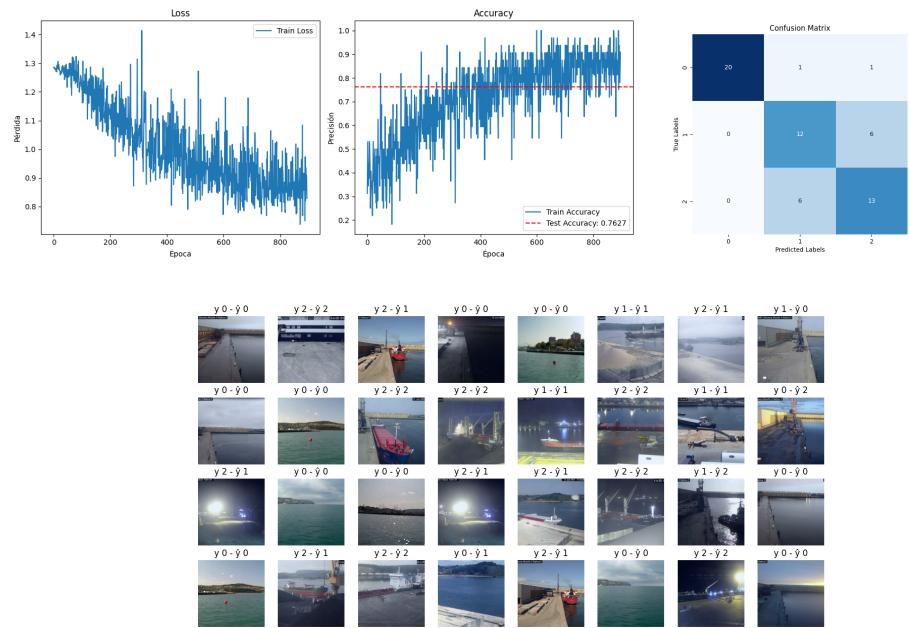


### 6.1.4. Con Aumento de datos e con Preentrenado

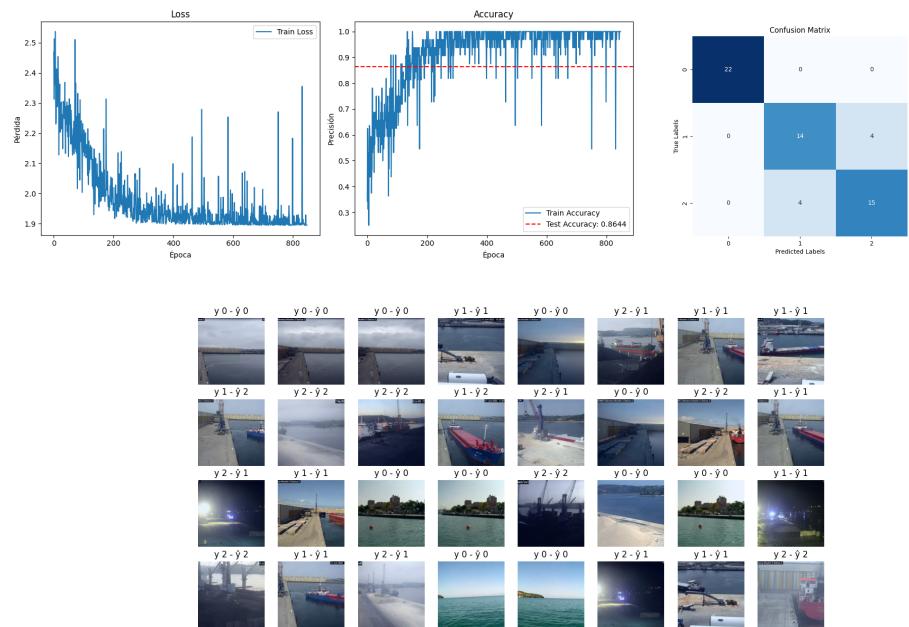


## 6.2. Segundo ejercicio

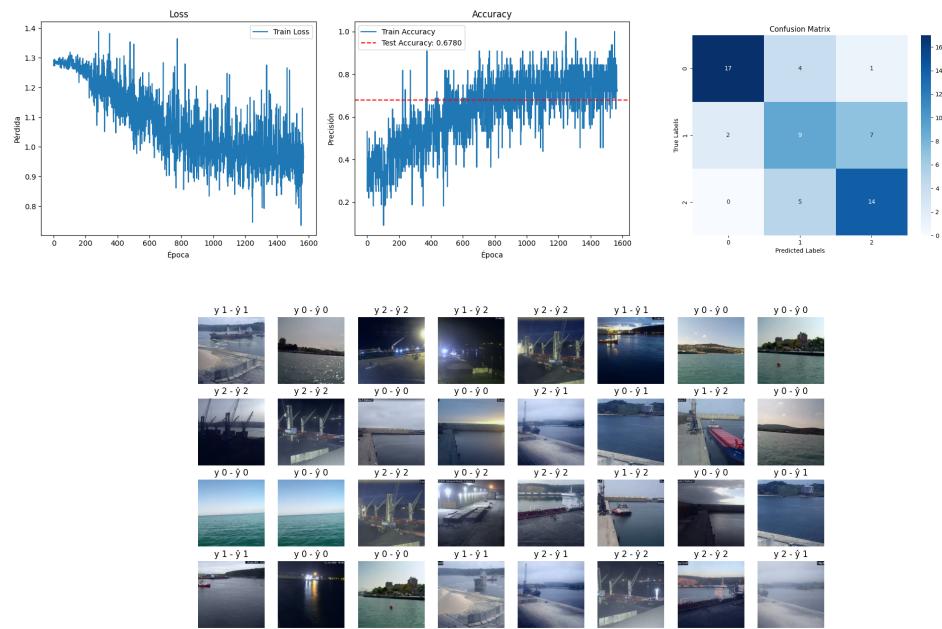
### 6.2.1. Sen Aumento de datos e sen Preentrenado



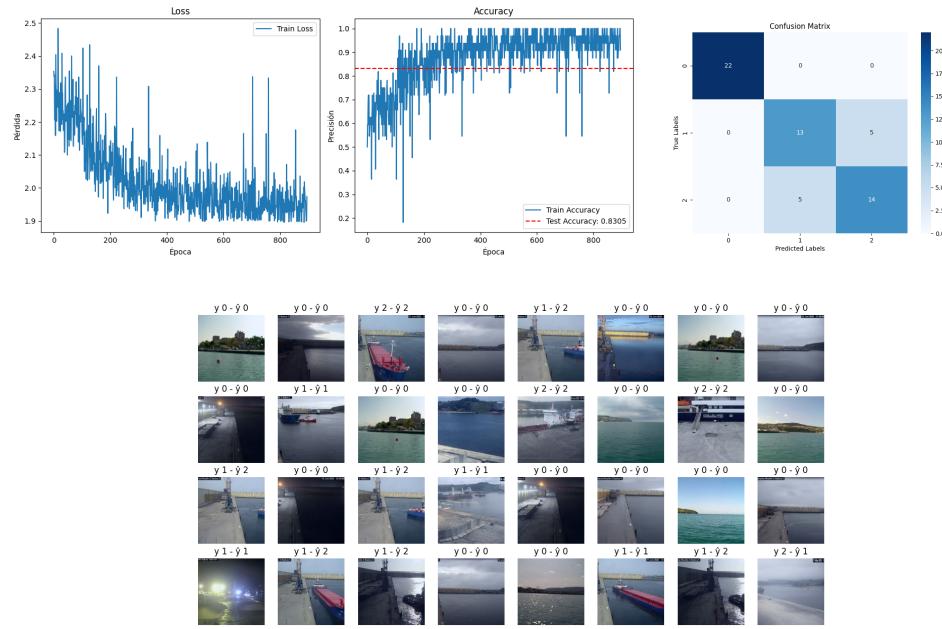
### 6.2.2. Sen Aumento de datos e con Preentrenado



### 6.2.3. Con Aumento de datos e sen Preentrenado



#### 6.2.4. Con Aumento de datos e con Preentrenado



## 7. Apéndice B: como executar o noso código

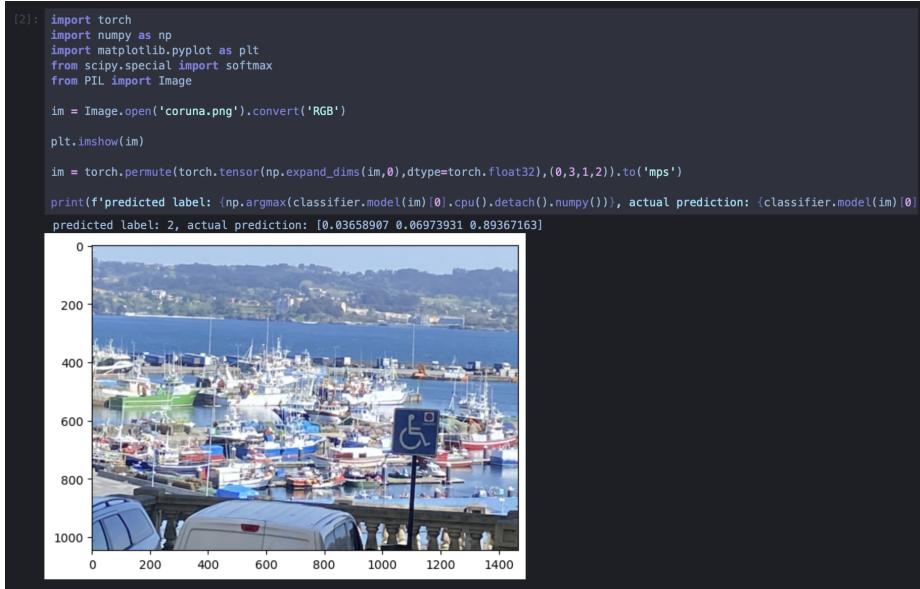
Para probar o primeiro e segundo exercicio con novas imaxes pode facerse algo así:

```
$ python main.py --load_model --model_path ex1params --test_images imagen.jpg
imagen2.jpg imagen3.jpg --not_train --mlp_head
$ python main.py --load_model --docked --model_path ex2params --test_images
imagen.jpg imagen2.jpg imagen3.jpg --not_train --mlp_head
```

Donde o programa respondería con <sup>8</sup>:

```
Model loaded from ex2params
Testing individual images:
Image: imagen.jpg
Prediction: Ship (Docked)
Confidence: tensor([0.2157, 0.2130, 0.5713], device='mps:0')
Image: imagen2.jpg
Prediction: Ship (Docked)
Confidence: tensor([0.2120, 0.2120, 0.5761], device='mps:0')
Image: imagen3.jpg
Prediction: No Ship
Confidence: tensor([0.5737, 0.2129, 0.2134], device='mps:0')
```

Ou adicionalmente, no pruebas.ipynb:



```
[2]: import torch
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import softmax
from PIL import Image

im = Image.open('coruna.png').convert('RGB')

plt.imshow(im)

im = torch.permute(torch.tensor(np.expand_dims(im,0),dtype=torch.float32),(0,3,1,2)).to('mps')

print(f'predicted label: {np.argmax(classifier.model(im)[0].cpu().detach().numpy())}, actual prediction: {classifier.model(im)[0]}')
predicted label: 2, actual prediction: [0.03658907 0.06973931 0.89367163]
```

The screenshot shows a Jupyter Notebook cell with the following code. It imports necessary libraries (torch, numpy, matplotlib, scipy, PIL), opens an image file 'coruna.png', converts it to RGB, and displays it using plt.imshow. Then, it converts the image to a torch tensor, permutes its dimensions, and moves it to the 'mps' device. Finally, it prints the predicted label (2) and the actual prediction tensor.

Como nota final podemos decir que a foto superior sacouse no porto de Os Castros en A Coruña e o modelo predice que é unha foto con *barco atracado*, e faino ainda que tanto os barcos como o peirao sexan distintos do que estivo acostumado a ver no adestramento.

<sup>8</sup>As imaxes **imagen.jpg**, **imagen2.jpg** e **imagen3.jpg** poden atoparse na entrega e se examínanse, pode verse como efectivamente, a primeira é unha imaxen de barco atracado, a segunda e a mesma imaxen pero recortada (sen ruído a confianza da rede aumenta levemente) e a terceira é tamén un recorte da primeira pero esta vez do chan (consideramos oportuno que a Rede clasifique este tipo de imaxes como clase 0, é dicir, sabe que non hai un barco).

## Referencias

- [1] Aitor Lewkowycz and Guy Gur-Ari. On the training dynamics of deep networks with  $l_2$  regularization, 2021.
- [2] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.
- [3] 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.
- [4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
- [5] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.