



viu

**Universidad
Internacional
de Valencia**

Aprendizaje por refuerzo aplicado a tareas de control

Titulación:
Máster de Inteligencia
Artificial

Curso académico
2021-2022

Alumno/a: Werner Seoane
Lucas Ezequiel
D.N.I: 39459365J

Director/a de TFM:
Gabriel Enrique Muñoz

Convocatoria:
Tercera

*The true sign of intelligence is not knowledge but
imagination.*

Albert Einstein

Agradecimientos

A mis padres y a mis hermanos. Sin ellos esto no sería posible.

A Agustina, por ser mi apoyo todos los días.

Índice general

Índice de figuras	III
Índice de tablas	V
Índice de algoritmos	VI
Resumen	1
1. Introducción	3
1.1. Acotación del problema	4
1.2. Dispositivo utilizado	5
1.3. Marco teórico	5
1.3.1. Aprendizaje por refuerzo	6
1.3.2. Algoritmo Policy Gradient	6
1.3.3. Algoritmo Actor-Critic	6
1.3.4. Transformers y Vision Transformers	6
2. Objetivos	9
3. Metodología	11
3.1. Obtención de los datos	11
3.2. Preprocesamiento de los datos	12
3.3. Análisis de la solución e implementación	14
3.3.1. Elección de los algoritmos utilizados por el agente	14
3.3.2. Definición del estado	15
3.3.3. Definición de la recompensa	16
3.3.4. Definición del entorno	18
3.3.5. Definición del agente	19
3.3.6. Entrenamiento	19
3.3.7. Evaluación	20
4. Resultados y Discusión	23

ÍNDICE GENERAL

4.1. Introducción	23
4.2. Policy Gradient	24
4.2.1. Experimento 1	24
4.2.2. Experimento 2	25
4.3. Actor-Critic	26
4.3.1. Experimento 1	27
4.3.2. Experimento 2	28
4.3.3. Experimento 3	29
4.4. Vision Transformers	30
5. Conclusiones	32
6. Limitaciones y Perspectivas de Futuro	33
A. Apéndize A	36
Bibliografía	38



Índice de figuras

1.1.	A la izquierda, el pipeline utilizado durante active tracking. A la derecha, el pipeline utilizado en passive tracking.	3
1.2.	DJI Tello. Dispositivo utilizado durante el proyecto	5
1.3.	Comparativa arquitectura (a) Transformer y (b) Vision Transformer	6
3.1.	Imágenes de ejemplo del conjunto final de datos	11
3.2.	Pipeline del procesamiento del conjunto de datos	12
3.3.	Algoritmo REINFORCE Policy Gradient	14
3.4.	Algoritmo <i>Actor-Critic</i>	15
3.5.	Arquitectura de red propuesta por Luo et al. (2019)	16
3.6.	Comparación de recompensa sobre la misma imagen dados dos puntos de vista distinto.	17
3.7.	Imagen ejecutada en dos experimentos distintos, a la misma distancia del punto objetivo, con diferente número de acciones.	21
4.1.	Experimento Policy Gradient 1 - Training Reward Mean	24
4.2.	Experimento Policy Gradient 1 - Testing reward mean	25
4.3.	Experimento Policy Gradient 1 - Episodes duration	25
4.4.	Experimento Policy Gradient 2 - Training Reward Mean	25
4.5.	Experimento Policy Gradient 2 - Testing reward mean	26
4.6.	Experimento Policy Gradient 2 - Episodes duration	26
4.7.	Experimento Actor Critic 1 - Training Reward Mean	27
4.8.	Experimento Actor Critic 1 - Testing reward mean	27
4.9.	Experimento Actor Critic 1 - Episodes duration	27
4.10.	Experimento Actor Critic 2 - Training Reward Mean	28
4.11.	Experimento Actor Critic 2 - Testing reward mean	28
4.12.	Experimento Actor Critic 2 - Episodes duration	28
4.13.	Experimento Actor Critic 3 - Training Reward Mean	29
4.14.	Experimento Actor Critic 3 - Testing reward mean	29
4.15.	Experimento Actor Critic 3 - Episodes duration	29
4.16.	Experimento <i>Vision Transformer</i> - Training Reward Mean	30

4.17. Experimento <i>Vision Transformer</i> - Training Loss	30
4.18. Experimento <i>Vision Transformer</i> - Testing reward mean	30
4.19. Experimento <i>Vision Transformer</i> - Episodes duration	31

Índice de tablas

Índice de algoritmos

1.	Algoritmo de recompensa inicial	17
2.	Ejecutar acción en el entorno	18

Resumen

El siguiente trabajo presenta una alternativa mediante el uso de técnicas de aprendizaje por refuerzo al algoritmo de seguimiento de una persona en tiempo real por parte de un dron. A diferencia de estudios ya realizados anteriormente y los cuales iremos detallando a lo largo del siguiente documento, no utilizaremos simuladores para la obtención de los datos, sino que utilizaremos la propia cámara del dispositivo para obtener estos, de manera que el entorno final sea lo más parecido al entorno de entrenamiento.

Construiremos en primer lugar una base teórica, en la que explicaremos brevemente los conceptos que involucra el siguiente trabajo y que da pie a entender las diferentes decisiones tomadas durante su ejecución.

Luego realizaremos un análisis de nuestros objetivos, tomaremos la definición de nuestro problema, y lo estructuraremos para adaptarlo al modelo de problema de aprendizaje por refuerzo. Además iremos comentando los problemas que cada uno de estos pasos conlleva y su evolución y distintas variantes a lo largo de los diferentes experimentos, incluyendo decisiones en el entrenamiento de nuestros agentes, así como la redefinición y adaptación mediante análisis de resultados, pero también mediante prueba y error de cada una de las piezas.

Finalmente haremos un análisis de los algoritmos utilizados y trataremos los diferentes problemas que estos fueron ocasionando durante el transcurso del entrenamiento.

Introducción

1

El objetivo es poder realizar el seguimiento a tiempo real de una persona a través de la cámara del dispositivo utilizando para ello visión por computador y técnicas de aprendizaje por refuerzo para controlar las acciones a tomar en cada momento. Lo que se conoce como *active tracking*.

Los actuales algoritmos se centran fundamentalmente en el reconocimiento del objeto y en la posición de éste con respecto a las coordenadas X e Y en relación al centro de la imagen a través de la cámara incorporada para guiar las acciones del dispositivo. Es lo que se conoce como *passive tracking*. Este tipo de algoritmos han ganado más atención debido a la simplicidad del problema y los avances en reconocimiento de objetos.

Otros intentos de realizar *object tracking* con aprendizaje supervisado se realizan utilizando modelos en simuladores tanto del dron como de las personas. Sin embargo, la idea del presente trabajo es la aplicación de diferentes técnicas de inteligencia artificial para lograr un agente que sea capaz de realizar el seguimiento utilizando simplemente imágenes que provengan del propio dispositivo en el cual se desplegará el agente.

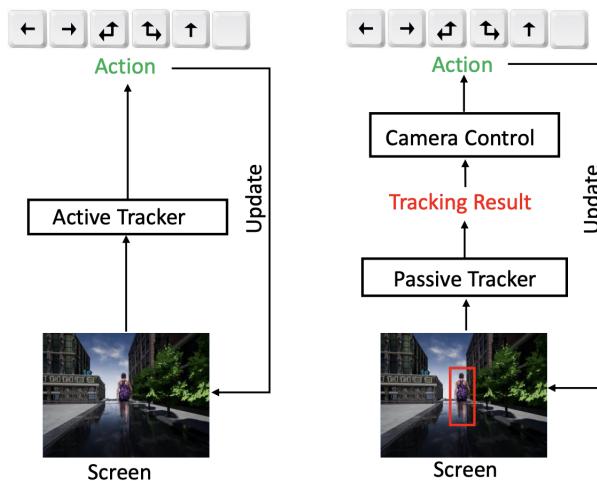


Figura 1.1: A la izquierda, el pipeline utilizado durante *active tracking*. A la derecha, el pipeline utilizado en *passive tracking*. Imagen tomada de [Luo et al. \(2019\)](#).

Para ello, no solo utilizaremos técnicas de aprendizaje por refuerzo, sino también técnicas de aprendizaje supervisado para la obtención de nuestro conjunto inicial.

El desarrollo del proyecto se llevará a cabo utilizando lenguaje Python y la librería [PyTorch](#) para el desarrollo de los modelos. El código completo estará disponible para su visualización en un repositorio de [GitHub](#), a excepción de los pesos de los diferentes agentes que se vayan guardando durante el entrenamiento debido al tamaño individual de cada uno de estos archivos.

1.1. Acotación del problema

Debido a que nos movemos en un entorno real, en el cual las posibilidades son muy amplias, no solo debido a la aleatoriedad del mismo sino también al número de acciones que podemos tomar, debemos definir ciertas restricciones.

En primer lugar, nos centraremos en obtener una solución que sea viable en entornos en los cuales las condiciones externas no sean un impedimento para el buen funcionamiento de nuestro algoritmo. Esto quiere decir que durante el desarrollo y testeo de los algoritmos obviamos factores tales como el viento, las condiciones de humedad o el grado de iluminación en un momento dado, que pudiesen afectar al rendimiento del dispositivo.

Por otro lado, debido a que el espacio de acciones disponibles es muy alto, en una primera fase dispondremos de tan solo 3:

- Girar a la derecha.
- Girar a la izquierda.
- Mantenerse en el mismo lugar.

Tanto el giro a la derecha como el giro a la izquierda se realizará en el dispositivo de manera controlada, esto quiere decir que nos moveremos siempre utilizando el mismo ángulo de giro. Si quisieramos añadir además un ángulo de giro variable junto con la acción a tomar podríamos hacerlo como parte de la salida de nuestro algoritmo, aunque involucraría un entrenamiento más prolongado en el tiempo y se escaparía a las restricciones de tiempo de este proyecto.

De tal forma que nuestra solución será valorada positiva o negativamente con respecto al eje X exclusivamente. Esto quiere decir que consideraremos que el algoritmo funciona de forma correcta si es capaz de moverse de tal manera que la persona se encuentre siempre centrada en el eje horizontal y no en el eje vertical.

En cuanto a la detección de la persona, tenemos en cuenta que nuestro dispositivo solo realizará el seguimiento de una sola persona, debido a la complejidad que esto supondría en cuanto al funcionamiento de nuestro algoritmo. Obviamos un escenario real y plausible que es el de encontrarnos en una misma imagen con múltiples personas, en cuyo caso deberíamos también desarrollar un mecanismo de selección de la persona a la cual quisiéramos seguir.

Por lo tanto, quedándonos con una parte simplificada del problema, podemos centrarnos en conseguir un algoritmo que pueda considerarse como el mínimo viable para conseguir nuestro objetivo.

1.2. Dispositivo utilizado

Para el desarrollo del proyecto se utilizará el dispositivo DJI Tello, ya que nos proporciona una interfaz de programación compatible con nuestras necesidades: control del dispositivo y transmisión de datos a través de una API en lenguaje Python.



Figura 1.2: Imagen del dispositivo utilizado durante el proyecto.

El dispositivo cuenta con una cámara integrada con una resolución máxima de 1280x720 píxeles, la cual creemos que es suficiente para el desarrollo del trabajo.

Un punto negativo del uso de drones como dispositivo de captura es el poco rendimiento de las baterías durante el vuelo. Concretamente, el dron utilizado permanece en vuelo unos 13 minutos como máximo.

Además, durante el desarrollo inicial del proyecto se pudo observar una latencia reseñable al intentar comunicar el dispositivo con el ordenador a la hora de transmitir las imágenes y de poder enviar señales de control debido a la débil conexión WiFi entre ambos puntos de comunicación. El intentar solucionar este problema no solo consumió tiempo de ejecución del proyecto sino que también nos impide poder llevar a cabo pruebas de control más realistas y nos acotará el margen de ejecución de nuestras pruebas finales.

1.3. Marco teórico

En esta sección haremos una breve introducción a los diferentes componentes teóricos que nos iremos encontrando a lo largo del trabajo. Empezaremos repasando los principales conceptos relacionados con el aprendizaje por refuerzo y en qué se diferencia de otros tipos de entrenamiento.

Después haremos una introducción a cada uno de los algoritmos que vamos a utilizar y comentaremos brevemente las diferencias y el por qué los elegimos. Por último comentaremos los *Transformers*([Vaswani et al., 2017](#)) y en especial los *Vision Transformers*([Dosovitskiy et al., 2020](#)), ya que serán utilizados durante uno de los experimentos que explicaremos en la sección 4 .

1.3.1. Aprendizaje por refuerzo

1.3.2. Algoritmo Policy Gradient

1.3.3. Algoritmo Actor-Critic

1.3.4. Transformers y Vision Transformers

Sin entrar en muchos detalles en cuanto a cómo funcionan los *Transformers* y los *Vision Transformers*, haremos un breve repaso de las características únicas de estas redes, ya que uno de los experimentos nombrados en la sección 4.4 será el de un agente con una red de tipo *Vision Transformer* ([Dosovitskiy et al., 2020](#)), que se encargará de realizar la transformación de la imagen de entrada.

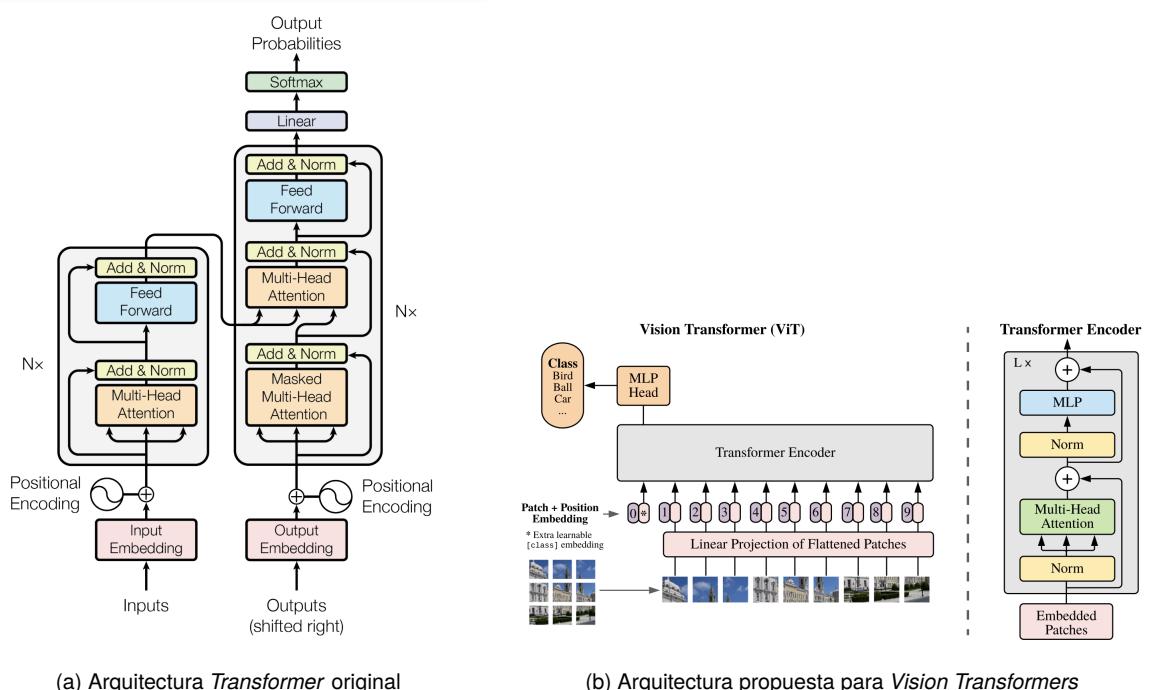
(a) Arquitectura *Transformer* original(b) Arquitectura propuesta para *Vision Transformers*

Figura 1.3: Comparativa arquitectura (a) Transformer y (b) Vision Transformer

Las redes basadas en *Attention*, en particular las redes *Transformers*, desarrolladas inicialmente por Google, se han convertido en la elección por defecto en problemas de lenguaje natural o NLP, ya que gracias a su eficiencia computacional y su escalabilidad, es posible entrenar modelos de más de 100 billones de parámetros. La idea detrás de estas redes es solucionar los problemas que presentan las redes neuronales recurrentes o RNN, tales como las redes LSTM o GRU, las cuales

no son capaces de captar dependencias a largo plazo, tal y como podemos encontrarnos en un texto de una novela.

Aunque los *Transformers* son ampliamente utilizados en procesamiento de lenguaje natural, las arquitecturas de redes convolucionales o seguían siendo el tipo de red dominante en el campo de la visión por computador. Es por ello que, inspirados en la arquitectura original de Transformer y en el éxito que estas contaban (y aún cuentan), se buscó aplicar los mismos principios en la visión por computador. Esto es lo que dio lugar a los *Vision Transformer*, cuya idea principal es la de partir una imagen en segmentos o *patches* y ser capaces de aplicar capas de Attention a cada uno de ellos.

A día de hoy nos encontramos muchas versiones disponibles y preentrenadas. Durante la ejecución de nuestro experimento utilizaremos un modelo disponible en la web [HuggingFace](#). En concreto utilizamos un modelo entrenado por Google, cuya entrada es una imagen de 224x224 píxeles, que será luego dividida en 16 parches de 14x14 píxeles y que fue entrenada en el conjunto de datos *ImageNet21k* ([Ridnik et al., 2021](#)). La salida de este modelo, es decir, la entrada a nuestro agente será un vector de tamaño 197x512, donde la primera dimensión representa el tamaño de la secuencia en la cual nuestra imagen ha sido partida tras el paso por el *Vision Transformer*, y cada elemento de esa secuencia consta de un total de 512 características.

Objetivos

2

Describe aquí el objetivo general de tu Trabajo Fin de Máster y, a continuación, define los objetivos parciales:

- 1. Objetivo parcial 1.**
- 2. Objetivo parcial 2.**
- 3. Objetivo parcial 3.**

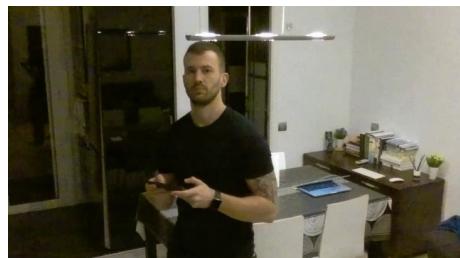
Metodología

3

3.1. Obtención de los datos

La toma de datos se hace mediante el análisis de archivos de vídeo tomados desde el propio dispositivo en un entorno controlado. Es importante destacar que la calidad de las imágenes no es alta, sobre todo en entornos en los cuales la iluminación no es favorable, por ejemplo en entornos con iluminación artificial.

El dispositivo nos proporciona una aplicación móvil para controlar la grabación y el almacenamiento de los videos que luego serán volcados en el ordenador para su posterior procesamiento, del cual hablaremos posteriormente.



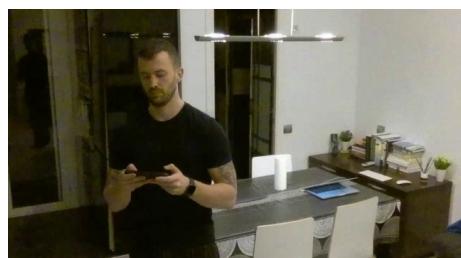
(a)



(b)



(c)



(d)

Figura 3.1: Imágenes de ejemplo del conjunto final de datos.

Se ha decidido también utilizar sólo imágenes procedentes del dispositivo y no incluir imágenes de terceras fuentes que podrían haber ayudado a mejorar el tamaño final de nuestro conjunto de datos ya que la idea es acercarse lo máximo posible a un entorno final real.

Debido a las leyes vigentes sobre vuelo de drones, los escenarios en los cuales fueron grabados los videos se redujeron drásticamente a un entorno cerrado, lo cual nos lleva a pensar que más adelante podríamos tener un problema de overfitting en nuestro modelo.

En un escenario ideal contaría con diferentes entornos que nos permitiesen descartar la posibilidad de que el agente aprenda que la posición de determinados objetos es condicionante para tomar una decisión y que se centrara solo en las persona de la imagen.

3.2. Preprocesamiento de los datos

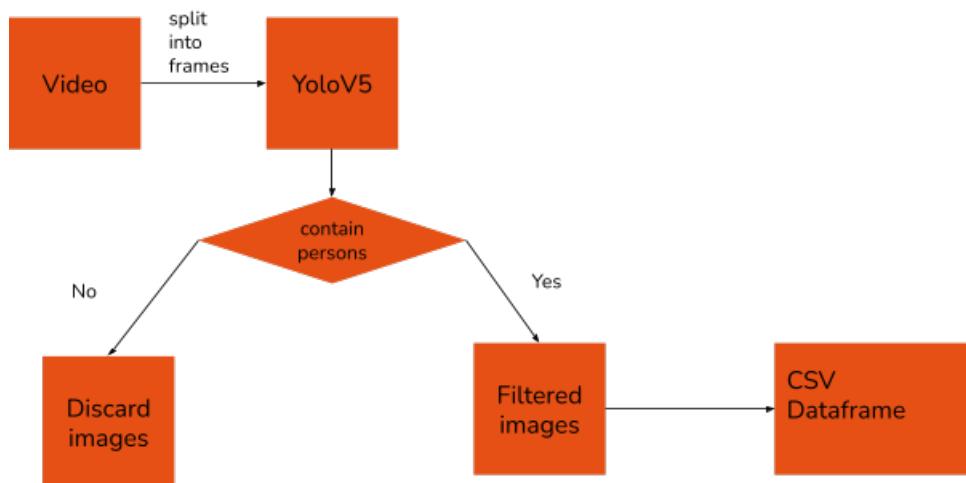


Figura 3.2: Pipeline del procesamiento del conjunto de datos.

Tras la recolección de videos y fotos desde el dispositivo, se realiza un primer proceso de filtrado. Este primer paso consiste en realizar un split de los archivos de video en frames individuales y realizar una operación de reducción del tamaño sobre los frames originales, pasando de 1280 píxeles de ancho y 720 píxeles de alto, a 640 píxeles de ancho y 360 píxeles de alto. Más adelante se tendrá que realizar una segunda reducción de las imágenes para adaptarlas al tamaño de entrada de la red convolucional previa al agente.

Este primer paso de filtrado es posible gracias a los algoritmos de detección de objetos, mediante el uso de redes convolucionales neuronales, que nos proporcionan las coordenadas de las bounding box de aquellas clases de objetos que queremos encontrar en nuestro conjunto de datos.

Durante el transcurso de nuestro proyecto utilizaremos *YOLOv5*, cuya implementación se encuentra disponible de forma gratuita como librería open source. Dicha implementación nos proporciona además múltiples configuraciones de la red. En nuestro caso, esto no fue muy relevante dado que solo lo utilizaríamos para una primera fase de preprocesamiento de las imágenes.

Debido a que la red *YOLOv5* fue entrenada sobre el conjunto de datos *ImageNet* (Deng et al., 2009) para detectar 1000 clases de objetos diferentes, debemos modificarla para que sea capaz de

detectar tan solo una persona en las imágenes que le pasamos, si es que la hay. La documentación de la librería nos ayuda a conseguir esto, de manera que tras una serie de pruebas tanto de diferentes arquitecturas de *YOLO* como de configuración de los parámetros, en concreto: el intervalo de confianza y el IoU threshold, la red nos devuelve la ubicación de la bounding box en donde se encuentra la persona si es que la hubiese.

Sin embargo, para poder adaptar la salida de la red a nuestras necesidades, lo que hicimos fue calcular el punto central de la bounding box en coordenadas X e Y, para que nuestro agente intente acercarse a ese punto durante el entrenamiento en el menor número de pasos posible. El resultado fue por un lado, un dataframe de Pandas que contenía la ruta de la imagen que *YOLO* había filtrado y las coordenadas X e Y del punto central de la bounding box.

Un punto a destacar es que las imágenes que no contenían personas fueron descartadas del conjunto de datos. Esta fue una decisión que se valoró al inicio con el fin de conseguir un agente y una definición más sencilla del problema. De no haber sido así, tendríamos que haber tenido en cuenta el hecho de que la imagen no presenta un punto objetivo y por lo tanto el dron debería permanecer quieto o girar hasta encontrar una persona.

El número total de imágenes de nuestro conjunto de datos inicial tras este primer paso se reduce a tan solo 1690 imágenes. Sin embargo, y debido a que nuestro objetivo es modelar los movimientos de giro del dron tenemos que ser conscientes de cuántas imágenes contamos para que el dron nos detecte a la derecha, a la izquierda o en el centro de la imagen.

En un primer análisis, nuestro conjunto de datos se encontraba completamente desbalanceado. Lo que hicimos para comprobar esto fue tomar las coordenadas centrales de la bounding box sobre el eje X que calculamos con *YOLO* y comprobar en qué parte de la imagen se encontraba. Si esta coordenada se encontraba entre los píxeles 0 y 280, entonces el dron tendría que girar a la izquierda, si se encontraba entre el 280 y el 360, entonces podríamos decir que el dron se mantendría prácticamente quieto, y si la coordenada se encontraba por encima de 360, el dron tendría que girar hacia la derecha.

Los resultados de este análisis fueron los siguientes:

- 340 imágenes se encontraban con la coordenada X en el lado izquierdo.
- 870 imágenes se encontraban con la coordenada X en el lado derecho.
- 480 imágenes se encontraban con la coordenada X en el centro de la imagen.

Debido a este desbalance, se tomó la decisión de igualar la cantidad de imágenes en cada lado, dejando un total de 340 imágenes por cada categoría, lo que hizo un total de 1020 imágenes finales para el entrenamiento y la validación de nuestro agente, lo que en contrapartida podría incrementar aún más el overfitting, pero nos aseguramos de que las decisiones que toma nuestro agente no se verán condicionadas por el número de acciones totales que debería tomar sobre un lado u otro.

Sin embargo, antes de tomar la decisión de descartar las imágenes se estudió la posibilidad también de realizar técnicas de data augmentation tal y como se recomienda en la literatura, utilizando técnicas de crop y volteo horizontal y vertical en cada una de las imágenes, pero la complejidad añadida de tener que recalcular el nuevo punto central hizo que descartemos esa posibilidad.

Debido a que nuestro conjunto de datos no es muy grande, decidimos que el conjunto de entrenamiento sea el 90 %, del cual el 80 % se usará para la fase de entrenamiento de nuestro modelo y el otro 20 % se usará para la fase de test. El 10 % restante de nuestro conjunto, aproximadamente 100 imágenes, lo reservamos para la etapa de validación, de manera que probaremos los resultados de nuestro agente sobre un conjunto de imágenes que no haya visto previamente.

3.3. Análisis de la solución e implementación

En la siguiente sección hablaremos de la implementación de cada uno de los elementos que componen nuestro problema, así como de las decisiones iniciales tomadas.

3.3.1. Elección de los algoritmos utilizados por el agente

Para entrenar nuestro agente utilizaremos métodos que optimicen la *policy* del agente, es decir, que optimicen la toma decisiones para maximizar la *reward* del entorno.

Comenzaremos utilizando el algoritmo *REINFORCE Policy Gradient*, el cual se considera de los más sencillos dentro de la familia de *Policy Gradient* y nos servirá como punto de partida para analizar los posibles problemas y limitaciones de nuestro entorno y nuestras definiciones. Con este algoritmo, el agente colecciona ejemplos del episodio utilizando para ello la *policy* actual. La figura 3.3 muestra dicho algoritmo:

$$Q^{\pi_\theta}(s_t, a_t) = v_t$$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

```
function REINFORCE
    Initialise  $\theta$  arbitrarily
    for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  do
        for  $t = 1$  to  $T - 1$  do
             $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$ 
        end for
    end for
    return  $\theta$ 
end function
```

Figura 3.3: Algoritmo REINFORCE Policy Gradient. Obtenida de la [UToronto](#)

Como siguiente paso, escogeremos nuevamente un algoritmo de la familia de *policy gradient* para el entrenamiento del agente. En este caso, el algoritmo *Actor-Critic* ([Haarnoja et al., 2018](#)). La

mayor diferencia entre este algoritmo y el anterior, es que en este caso contamos con 2 componentes separados: el *Actor*, que será el responsable de devolver la distribución de probabilidades de cada una de las acciones del agente y el *Critic*, que será el encargado de estimar el valor del estado en el que se encuentra dicho agente. El algoritmo utilizado puede verse en la figura 3.4.

Algorithm 1 Monte Carlo on policy actor-critic.

Require: Initialize policy π with parameters θ_π and value critic v_π with parameters θ_v

- 1: **for** each episode **do**
- 2: Get initial state s
- 3: Initialize storage buffer S, A, R, S'
- 4: **for** $i = 1, 2, 3 \dots N$ steps **do**
- 5: Sample action with policy: $a \sim \pi_\theta(s)$
- 6: Run action through environment, obtain reward and post state: $r, s' \leftarrow ENV(s, a)$
- 7: Collect and store: $S, A, R, S' \leftarrow s, a, r, s'$
- 8: $s \leftarrow s'$
- 9: **end for**
- 10: Compute discount returns: $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
- 11: Update θ_v to minimize $\sum_{n=1}^N \|v_\pi(s_n) - \hat{V}_n\|^2$
- 12: With learning rate α , update policy: $\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_\theta \log \pi(A|S)v_\pi(S)$
- 13: **end for**

Figura 3.4: Algoritmo *Actor-Critic*. Obtenida de [Reddit](#)

3.3.2. Definición del estado

El estado es la representación del problema que el agente debe resolver. En este caso, el estado será una imagen de la cámara del dron, a la que se le aplicó previamente un procesamiento, tal y como comentamos en la sección 3.2, junto con el punto en el que se encuentra el agente, expresado en un número entre 0 y 1, representando el 0 estar en el borde izquierdo de la imagen y 1 estar en el borde derecho de esta.

Es decir, el agente recibirá un vector como entrada, de tamaño 4097. Los primeros 4096 elementos representan la imagen, la cual ha sido procesada por una CNN preentrenada, que en nuestro caso ha sido VGG16 ([Simonyan y Zisserman, 2014](#)), la cual recibe como entrada un vector de tamaño 224x224x3, es decir, una imagen en formato RGB de 224x224 píxeles. Si bien la elección de nuestra CNN no fue basándose en ningún criterio en particular, la idea era disminuir el tamaño de la entrada de nuestro agente.

También se tuvieron en cuenta diferentes soluciones para la definición del estado. Entre ellas, la posibilidad de que el estado se formase de varias imágenes al mismo tiempo, debido el componente temporal con el que cuenta nuestro problema. Este estado condicionaría también la arquitectura de la red de nuestro agente, dado que en este caso necesitaríamos tratar con redes recurrentes o en su defecto con redes de tipo Transformers ([Vaswani et al., 2017](#)), tal y como se explica en el trabajo realizado por [Luo et al. \(2019\)](#) y podemos ver en la figura 3.5, donde podemos ver que después del *encoder*, tal y como se cita en el trabajo, se utiliza una red LSTM.

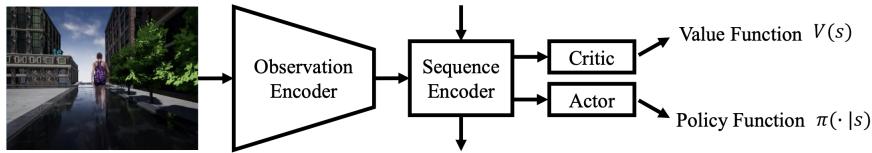


Figura 3.5: Arquitectura de red propuesta por [Luo et al. \(2019\)](#).

Para aumentar la velocidad de nuestro entrenamiento, la imagen será preprocesada por la red convolucional al inicio de cada episodio y se mantendrá constante, dado que lo que iremos cambiando con cada acción durante el entrenamiento es el punto en el que se encuentra el agente.

Por último, el estado contiene en su último elemento, el punto en el que se encuentra el agente en un momento dado, de manera que este debería ser capaz de tomar decisiones basándose no solo en la representación de la imagen sino también del valor que este valor tenga en ese instante.

3.3.3. Definición de la recompensa

El desarrollo de la recompensa fue uno de los puntos críticos en cuanto a la implementación y la definición de nuestro problema. Por un lado, tenemos que intentar que nuestro agente sea capaz de ir aprendiendo los pasos intermedios que lleven a la obtención de la recompensa óptima, en el menor número de pasos posibles y por otro, definir cuánta recompensa en términos absolutos el agente recibirá cuando acaba correctamente el episodio, en contraposición a cuando llega al número máximo de intentos sin haber obtenido la recompensa máxima.

La idea inicial fue crear una zona de recompensa proporcional a la distancia del punto en el que se encontraba el agente y el punto obtenido durante el preprocesamiento de los datos. Esto lo hicimos dividiendo la imagen en secciones longitudinales de tamaño fijo. Dado que nuestra imagen era de 640 píxeles de ancho, lo que hicimos fue dividirla en 32 secciones de 20 píxeles cada una. La recompensa final obtenida sería inversamente proporcional a la distancia, medida en número de secciones, en que el agente se encontraba con respecto al punto objetivo. De manera que si el agente se encontraba a 5 secciones de nuestro punto final, obtendría menos recompensa que si se encontrara a 2 secciones, tal y como se puede observar en el algoritmo 1:

Algoritmo 1: Algoritmo de recompensa inicial

1. Dividimos la imagen en secciones longitudinales de tamaño fijo $Secciones_{total}$.
 2. Establecemos una recompensa máxima del entorno $Recompensa_{max}$.
 3. Calculamos la sección en la que se encuentra nuestro agente:
 $Seccion_{agente} = round(Pagente_{pixels}/Ancho_{imagen})$.
 4. Calculamos la sección en la que se encuentra nuestro punto final:
 $Seccion_{objetivo} = round(Pobjetivo_{pixels}/Ancho_{imagen})$.
 5. Si $Seccion_{agente} = Seccion_{objetivo}$ devolvemos $Recompensa_{max}$.
Sino devolvemos $1 - (Seccion_{agente} - Seccion_{objetivo})/Secciones_{total}$
-

Lo que conseguimos con esta recompensa fue obtener un *heatmap* alrededor del punto objetivo, de manera que el agente podría interpretar si se estaba acercando o alejando.

La recompensa máxima fue establecida a 1, de manera que los saltos entre las recompensas parciales y la recompensa final fuese siempre proporcional a la distancia. Más adelante también comprobaremos cómo este valor puede afectar al desarrollo de una solución por parte de nuestro agente.



Figura 3.6: La misma imagen, con 2 puntos definidos por el agente a diferente distancia del punto objetivo. La imagen superior obtendría la puntuación máxima mientras que la imagen inferior solo obtendría una recompensa relativa a la distancia del punto objetivo.

En la figura 3.6 podemos observar cómo funciona nuestro sistema de recompensa inicial. En

la imagen situada en la parte superior, el punto amarillo, que representa el punto en el que se encuentra el agente, está dentro de la misma sección que el punto objetivo, y por lo tanto la recompensa será máxima. Mientras que en el caso de la imagen inferior, el punto donde se encuentra el agente está a una distancia mayor a una sección (definida por defecto en 20 píxeles), y por lo tanto la recompensa será proporcional al número de secciones que se encuentra hasta llegar al punto objetivo.

3.3.4. Definición del entorno

Para desarrollar el entorno, lo que se hizo fue implementarlo siguiendo como ejemplo los entornos propuestos por OpenAI Gym ([Brockman et al., 2016](#)). Esto nos facilitó tener una primera estructura y una primera definición de los métodos que deberíamos implementar.

Al inicio de cada episodio, nuestro entorno se encargará de devolver una imagen del conjunto de datos (entrenamiento, test o validación), así como las coordenadas del punto en el que se encuentra la persona, expresados en píxeles, que es lo que compone nuestro estado tal y como describimos en la sección [3.3.2](#). Estos valores se guardan durante toda la ejecución del episodio y se renuevan una vez que se empieza uno nuevo.

La ejecución de cada acción sobre el entorno nos devolverá un nuevo estado, la recompensa asociada con ese estado y un valor que nos indicará si el estado es final o no, es decir, si el episodio finaliza en ese estado.

El algoritmo [2](#) detalla el pseudocódigo utilizado tras recibir una acción por parte del agente. Podemos observar que este algoritmo mueve el punto del agente basándose en la acción que este toma, siempre en una cantidad fija de píxeles, que corresponde con el tamaño de cada una de las secciones en las que fue dividida la imagen tal y como se explica en el apartado [3.3.3](#). Esto se hizo de tal forma que el agente se moviese siempre a una sección diferente y por lo tanto la recompensa obtenida también sea diferente.

Algoritmo 2: Ejecutar acción en el entorno

```

width ← 640
distance ← 20
if accion == 'LEFT' then
| point ← point - distance
end
if action == 'RIGHT' then
| point ← point + distance
end
point ← torch.clamp(point, 1, width - 1)
reward ← calculateReward(point)
done = EnvironmentMaxReward == reward
return reward, point, done
  
```

Podemos observar también que la acción de permanecer quieto no tiene ninguna consecuencia en el estado del agente y la utilizaremos como posible acción para finalizar el episodio por parte del agente.

Durante las fases de entrenamiento, testeo y validación de nuestro agente, cada una de las acciones sobre el entorno involucra el cálculo de la nueva posición del punto en el que se encuentra el agente. Una vez el agente fuese desplegado para controlar el dron, cada ejecución de la acción involucraría la ejecución de esa acción en el dron, ya sea girar a la derecha, a la izquierda, o permanecer quieto y obtener una nueva imagen. En este caso, el punto de vista del dron siempre sería el centro de la imagen.

3.3.5. Definición del agente

El agente es el componente que se encarga de tomar decisiones. En este caso, el agente será el encargado de tomar la decisión de girar la cámara del dron a la izquierda, a la derecha o de permanecer quieto, dado un estado del entorno, el cual está definido como se detalla en la sección [3.3.2](#).

El objetivo de nuestro agente será encontrar aquellas acciones que maximicen las recompensas obtenidas. Este objetivo será alcanzado dependiendo del algoritmo que utilicemos para ello, pero independientemente de esto, las recompensas obtenidas y el conjunto de acciones será el mismo.

En cuanto a la implementación, el agente será una red neuronal cuya arquitectura iremos variando según los diferentes experimentos, que extenderá de la clase nn.Module de PyTorch y a la cual le añadiremos métodos propios para facilitar la lectura del código.

3.3.6. Entrenamiento

En cuanto al proceso de entrenamiento, destacamos el hecho de que cada episodio está compuesto por una sola imagen. Es decir, cuando el agente llegue a la recompensa máxima establecida en cada imagen o cuando se llegue al número máximo de acciones que definimos durante el bucle de ejecución, el episodio acabaría.

La inclusión de un número máximo de pasos por imagen fue una decisión tomada para que nuestro agente no entrase en un bucle infinito si decidiese por ejemplo, girar siempre hacia la derecha, llegando al extremo de la imagen donde no se encuentra la persona. Esta decisión también nos lleva a experimentar con un parámetro extra en nuestro entrenamiento, ya que un número muy elevado de acciones puede llevarnos a que el agente no aprenda correctamente y por lo tanto la exploración sobre el entorno no tenga ningún efecto, y por otro lado, un número muy pequeño podría llevarnos a que el agente no tuviese el tiempo suficiente para aprender la policy adecuada dado el estado. Por lo general, este valor oscilaba entre 50 y 100 acciones por imagen, aunque también se realizaron pruebas con valores menores y mayores a estos.

Debido al problema de la falta de imágenes en diferentes entornos, tal y como comentamos en la sección [3.2](#). Se tomó la decisión de que el punto inicial se inicialice de manera aleatoria,

entre un valor de 0 y 1, que luego será multiplicado por el ancho de la imagen para darnos las coordenadas reales y calcular la recompensa en ese punto. La idea detrás de esta decisión es dotar al entrenamiento de un componente aleatorio y por lo tanto evitar que nuestro agente pueda realizar overfitting sobre el conjunto de datos, dado que para una misma imagen , durante el entrenamiento, el punto de partida sea diferente.

Durante el proceso de entrenamiento se realizaron multitud de pruebas, no solo en lo que respecta a los parámetros de nuestro modelo, sino también a los criterios de parada de cada episodio, el número máximo de acciones a tomar o la recompensa obtenida al seleccionar la acción de permanecer quieto. De estas diferentes decisiones hablaremos en cada uno de los experimentos.

3.3.7. Evaluación

Para evaluar la calidad de los resultados del agente, comprobaremos sobre cada imagen del conjunto de validación, cuándo el agente decide quedarse quieto y la recompensa que obtiene al hacerlo. Esto se asemeja al comportamiento real que tendría al ser desplegado en el dron.

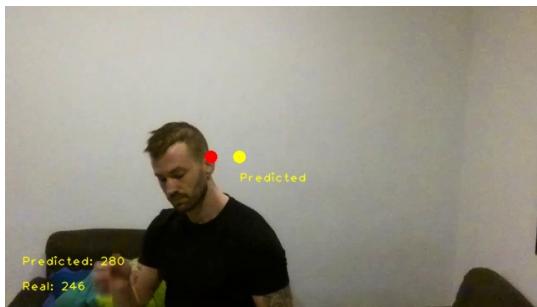
La evaluación comienza con las coordenadas del punto del agente en las coordenadas centrales de la imagen, de manera que simula el punto central de la cámara del dron. Lo que se hace a continuación es ejecutar el agente sobre ese estado y aplicar las acciones de este hasta encontrarnos con la acción de permanecer quieto, lo cual sería el indicador de que el agente se encuentra en la posición en la que se encuentra la persona. En ese momento, dado que contamos con las coordenadas reales obtenidas durante el procesamiento, calculamos la recompensa.

Esta operación se realiza tanto para el conjunto de test durante el entrenamiento, como para el conjunto de validación después del entrenamiento. En ambos casos, nuestro modelo funciona en modo evaluación y por lo tanto no se propagan cambios a los pesos de este.

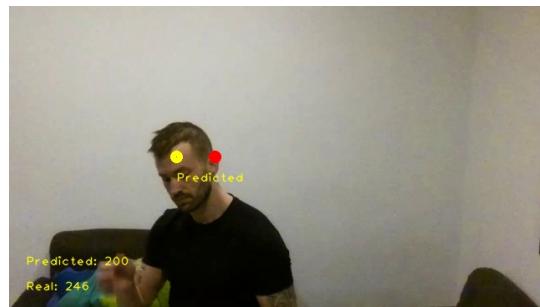
Además de la recompensa total, nos interesa en este caso obtener un agente que decida moverse de manera uniforme hacia una dirección en el menor número de acciones posible. Pensemos que cada acción que tome el agente será una acción que el dispositivo tendrá que realizar y por lo tanto es tiempo de ejecución que se pierde. Es decir, un agente que consigue la recompensa máxima en 5 acciones será más valioso que uno que la consiga en 10.

Aunque el hecho de conseguir la recompensa máxima del entorno en nuestro caso no es indicativo de la calidad de nuestro agente, por ejemplo podemos pensar que nuestro agente se acerca al punto en el que se encuentra la persona, sin llegar a estar exactamente en donde debería, pero sin embargo lo hace de manera consistente, en un número de acciones reducido por frame, lo que al final nos permite ser más rápidos en la ejecución final y por lo tanto, a efectos prácticos, nos puede incluso llegar a ser más valioso.

Un ejemplo de este comportamiento lo vemos en la figura 3.7, en el que se puede ver que sobre la misma imagen, ejecutada en 2 experimentos diferentes, tenemos 2 puntos de vista por parte del agente a prácticamente la misma distancia. La imagen de la izquierda requirió de solamente 2



(a) Ubicación del agente tras 2 acciones



(b) Ubicación del agente tras 6 acciones

Figura 3.7: Imagen ejecutada en dos experimentos distintos, cuyo punto predicho se encuentra a la misma distancia del punto objetivo, con diferente número de acciones.

acciones, mientras que la imagen de la derecha de 6 acciones. Por lo tanto, para esta imagen nos interesaría el agente que produjo la imagen de la izquierda.

Debido a la alta latencia que se producía al conectar el dron con nuestro ordenador, la ejecución en el dispositivo real no fue posible. Lo que se hizo en su lugar fue analizar frame por frame una serie de videos grabados desde el dispositivo, ejecutando una acción en cada uno de ellos para que simulara la acción a tiempo real que tomaría el agente. Todos estos factores están sujetos a una evaluación subjetiva que iremos comentando más adelante en los diferentes experimentos que realicemos.

Resultados y Discusión

4

4.1. Introducción

A continuación haremos un repaso de los resultados obtenidos en el subconjunto de experimentos más representativo, realizados con los diferentes algoritmos propuestos.

Detallaremos brevemente también las conclusiones parciales que obtenemos en cada experimento, así como posibles soluciones a los problemas que se nos plantean en cada uno.

Cabe mencionar que debido al coste computacional de ejecutar cada experimento sobre el conjunto de datos total, cada experimento se compone de subexperimentos previos, en los cuales intentábamos probar nuestras soluciones con un conjunto de datos menor al original, alrededor de unas 20 imágenes.

La idea detrás de esta subexperimentación era conseguir un modelo de agente que sea capaz de realizar *overfitting* sobre el conjunto de datos de manera relativamente rápida y que obtuviese una convergencia tanto en la recompensa obtenida tanto en las imágenes del conjunto de entrenamiento como en el conjunto de test como en el número de pasos que realizaba sobre cada una de las imágenes. Esto nos permitió también acortar el tiempo entre las diferentes pruebas, ya que el conseguir este overfitting nos aseguraba que el agente seguiría en un principio las ideas que queríamos implementar y además provocó que pudiesemos experimentar con diferentes definiciones tanto de nuestra recompensa como de nuestro entorno, así como hacer *fine tuning* de nuestros hiperparámetros.

Durante los experimentos podremos comprobar que en alguno de ellos el número de métricas que mostraremos es más alto que en otros, esto no es solo debido a que queramos destacar una cualidad específica del experimento, sino que también formó parte de la experimentación y el descubrimiento de algunos de los problemas que nos fuimos encontrando, que nos llevó a tener que realizar el seguimiento de algunos aspectos del entrenamiento que no estaban contemplados en experimentos anteriores. Estos problemas estuvieron relacionados principalmente con la convergencia en la toma de decisiones del agente tal y como se discutirá más adelante.

Por último también mostraremos algunos resultados de las imágenes que fuimos obteniendo en cada caso. Cabe recordar que la ejecución del agente durante el test se detiene cuando decide ejecutar la acción de permanecer quieto, y que tanto esta recompensa como el número de acciones

hasta llegar a ella se tienen en cuenta como medidas subjetivas de la eficacia del agente, tal y como comentamos en la sección 3.3.7.

El total de los experimentos está disponible en el repositorio de [GitHub](#) para un análisis más extenso y se adjunta el título de cada uno de ellos en el Apéndice A.

La herramienta utilizada para monitorear el progreso de las diferentes métricas durante los diferentes entrenamientos es [Tensorboard](#), ya que además de su fácil integración con [PyTorch](#) nos permitía comparar dichas métricas entre los diferentes experimentos.

4.2. Policy Gradient

El algoritmo *Policy Gradient* fue nuestra primera opción a la hora de implementar nuestro agente. Se buscaba obtener una primera aproximación a nuestro problema y buscábamos sobre todo tener un modelo que sirviese como *baseline*, así como definir la estructura general de nuestro programa.

4.2.1. Experimento 1

En este primer experimento utilizamos como criterio de finalización del episodio que el agente realizase un máximo de 50 acciones o que obtuviese la máxima recompensa del entorno, en este caso se mantuvo la recompensa proporcional con la distancia al punto objetivo y no se hizo ninguna recompensa extra por llegar al final del episodio correctamente.

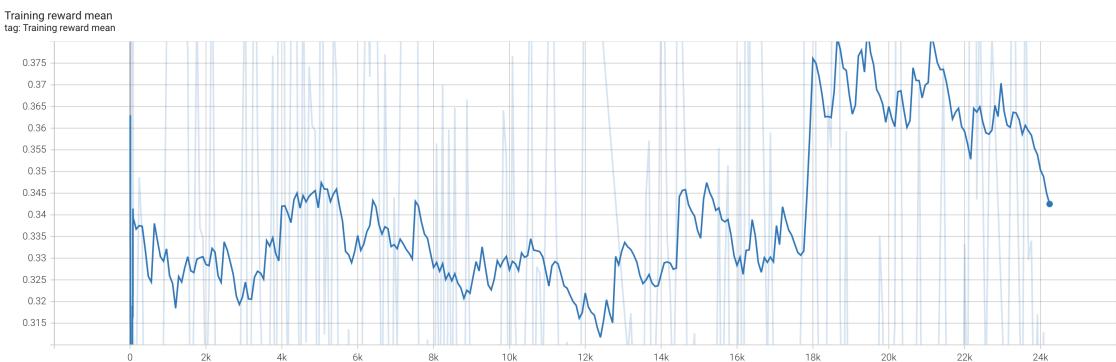


Figura 4.1: Experimento Policy Gradient 1 - Training Reward Mean

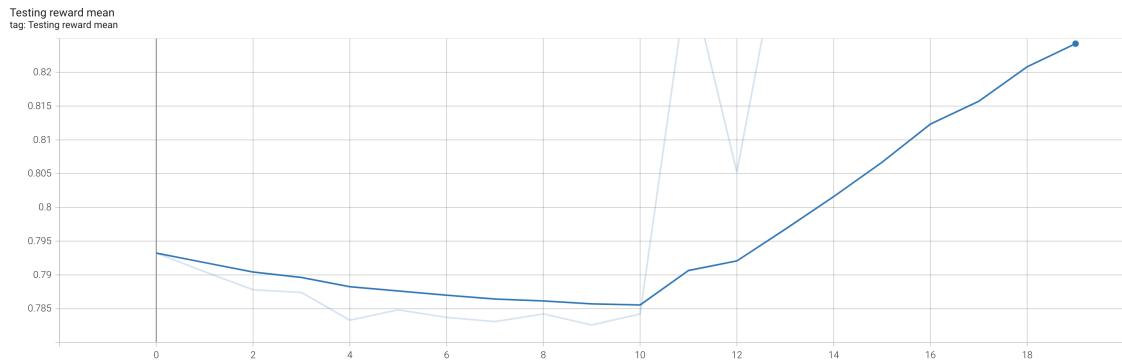


Figura 4.2: Experimento Policy Gradient 1 - Testing reward mean



Figura 4.3: Experimento Policy Gradient 1 - Episodes duration

Lo que se puede apreciar es que existe una tendencia a la baja de nuestro algoritmo en cuanto al número de acciones en el conjunto de entrenamiento. Por otra parte, se aprecia que la recompensa en el conjunto de test comienza a subir a partir de la época 10, lo cual marca un punto importante de inflexión en esta.

4.2.2. Experimento 2

Los parámetros utilizados en este experimento son similares al anterior, pero en este caso el entrenamiento se ejecuta durante unas 50 épocas, unas 30 más que el experimento 4.2.1.

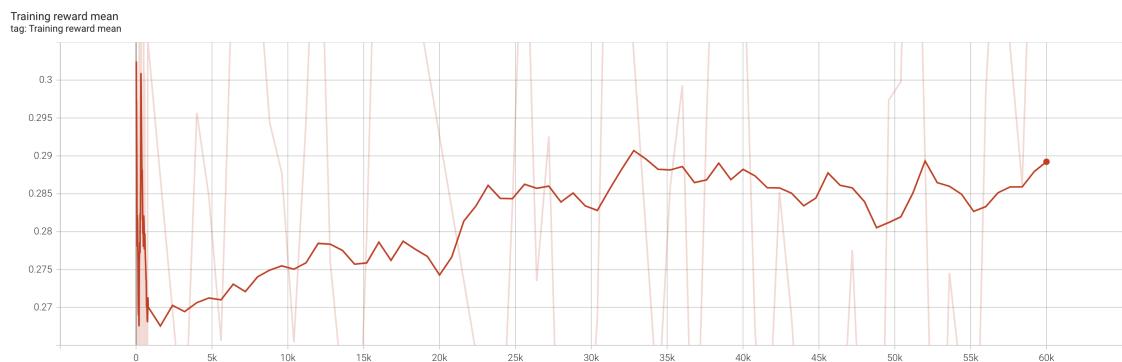


Figura 4.4: Experimento Policy Gradient 2 - Training Reward Mean

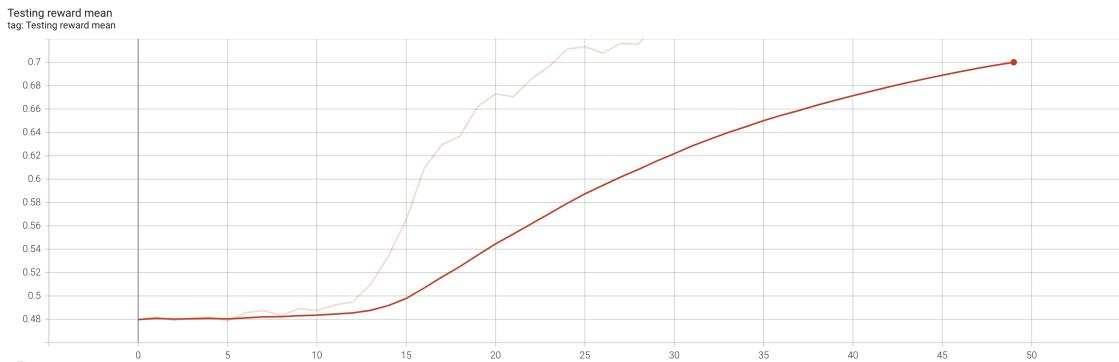


Figura 4.5: Experimento Policy Gradient 2 - Testing reward mean

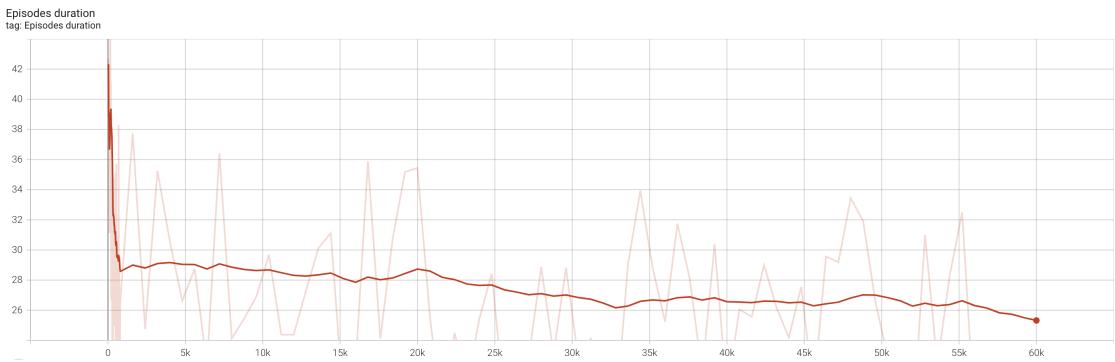


Figura 4.6: Experimento Policy Gradient 2 - Episodes duration

Vemos que en este experimento la tendencia que vimos en el experimento anterior se confirma: el número de acciones por imagen en el conjunto de entrenamiento parece seguir una tendencia a la baja y tanto la recompensa en el conjunto de test como en el entrenamiento aumentan según van avanzando las épocas.

4.3. Actor-Critic

Veremos ahora el comportamiento de algunos de los experimentos llevados a cabo con el algoritmo Actor Critic.

4.3.1. Experimento 1

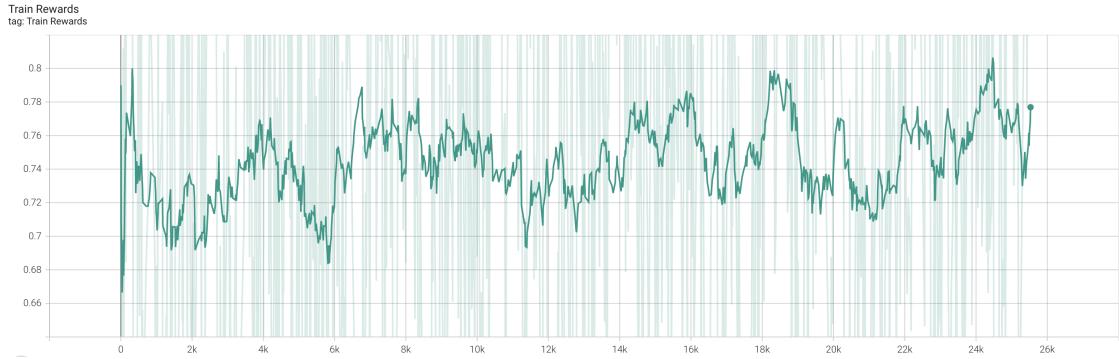


Figura 4.7: Experimento Actor Critic 1 - Training Reward Mean

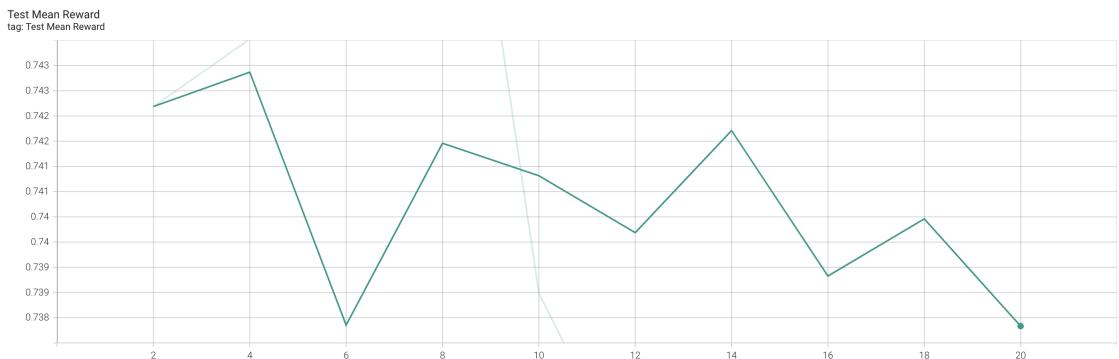


Figura 4.8: Experimento Actor Critic 1 - Testing reward mean

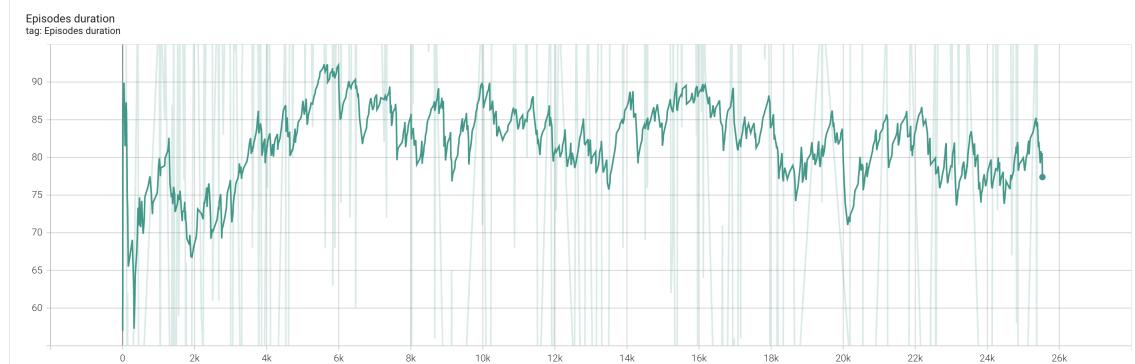


Figura 4.9: Experimento Actor Critic 1 - Episodes duration

4.3.2. Experimento 2

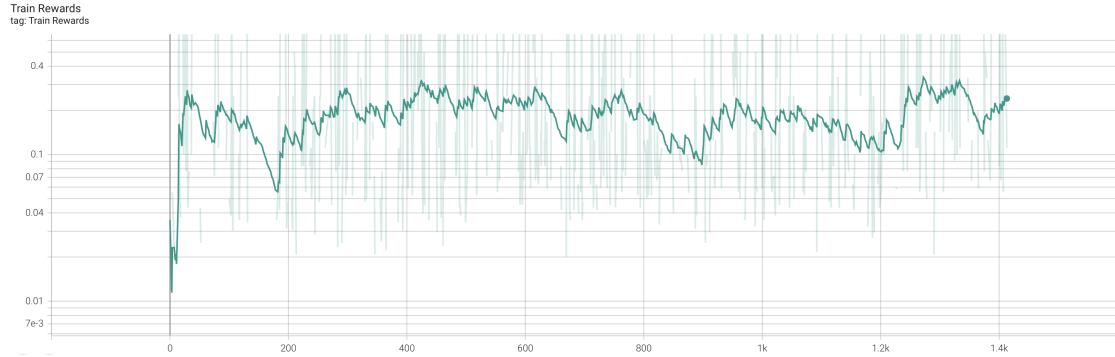


Figura 4.10: Experimento Actor Critic 2 - Training Reward Mean

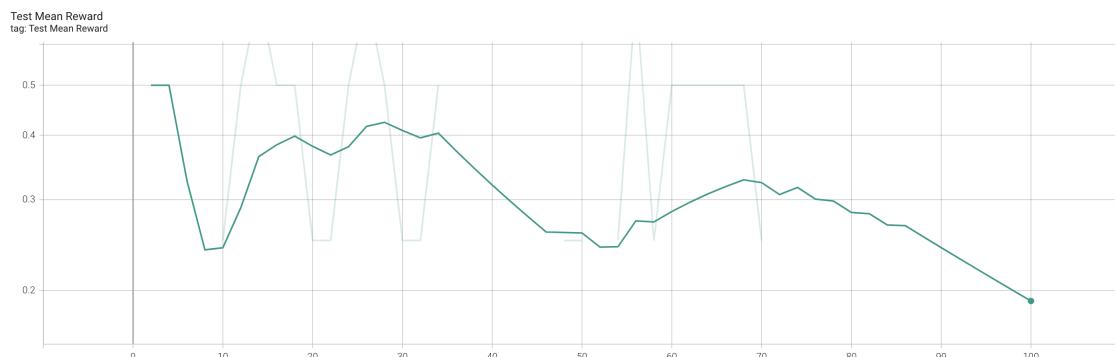


Figura 4.11: Experimento Actor Critic 2 - Testing reward mean

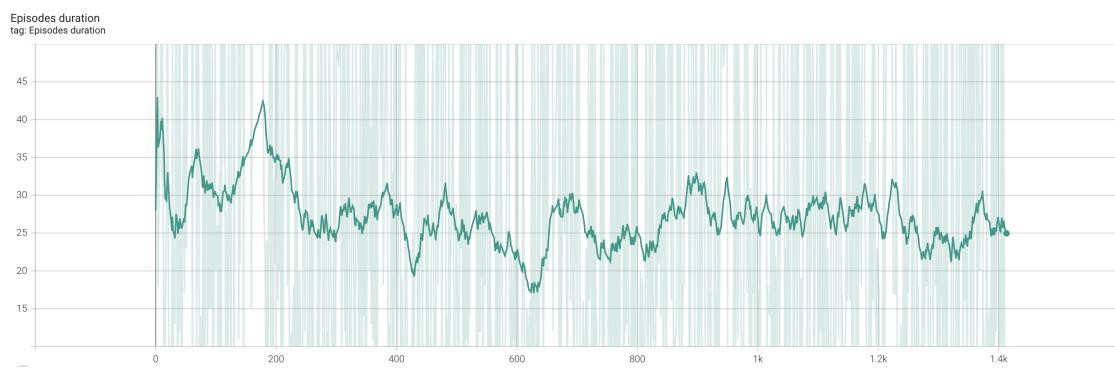


Figura 4.12: Experimento Actor Critic 2 - Episodes duration

4.3.3. Experimento 3

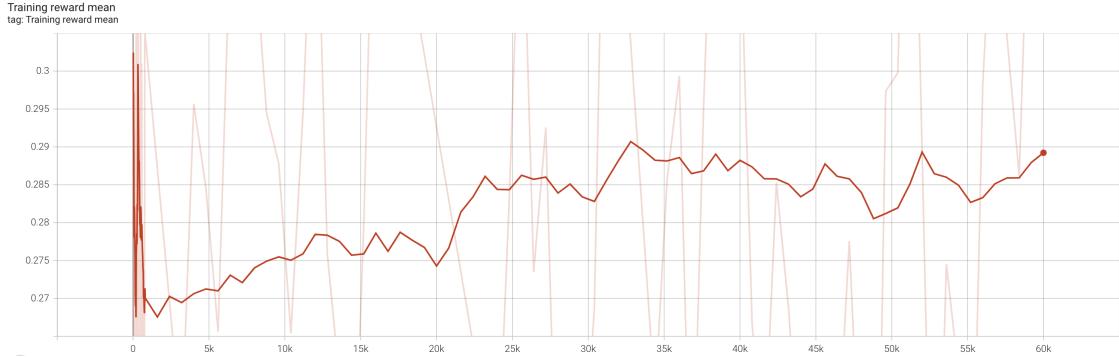


Figura 4.13: Experimento Actor Critic 3 - Training Reward Mean

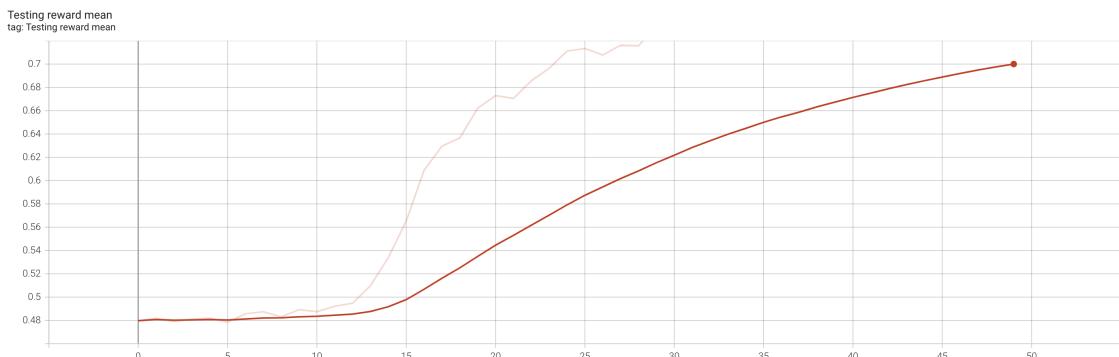


Figura 4.14: Experimento Actor Critic 3 - Testing reward mean

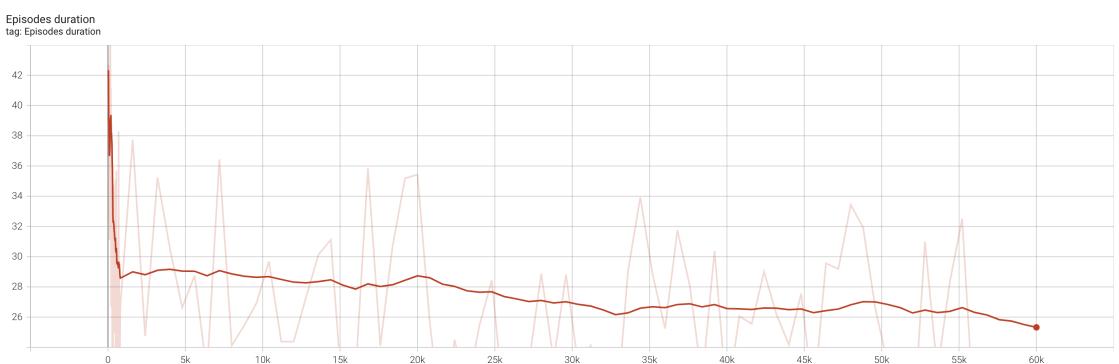


Figura 4.15: Experimento Actor Critic 3 - Episodes duration

4.4. Vision Transformers

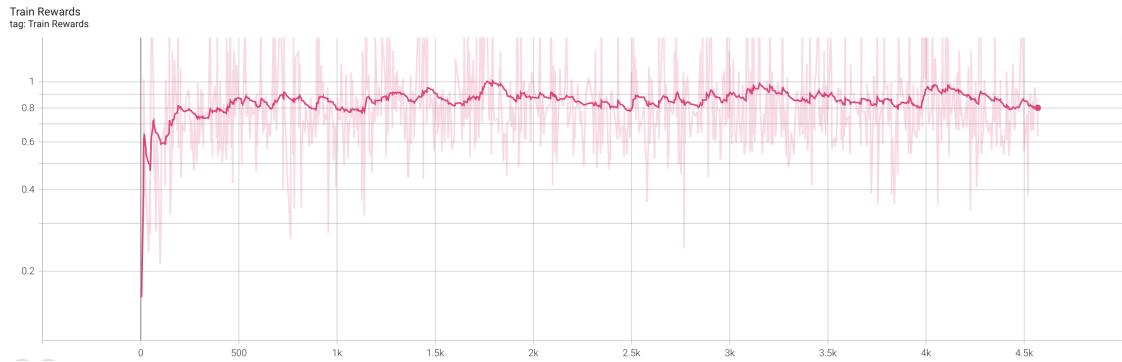


Figura 4.16: Experimento *Vision Transformer* - Training Reward Mean

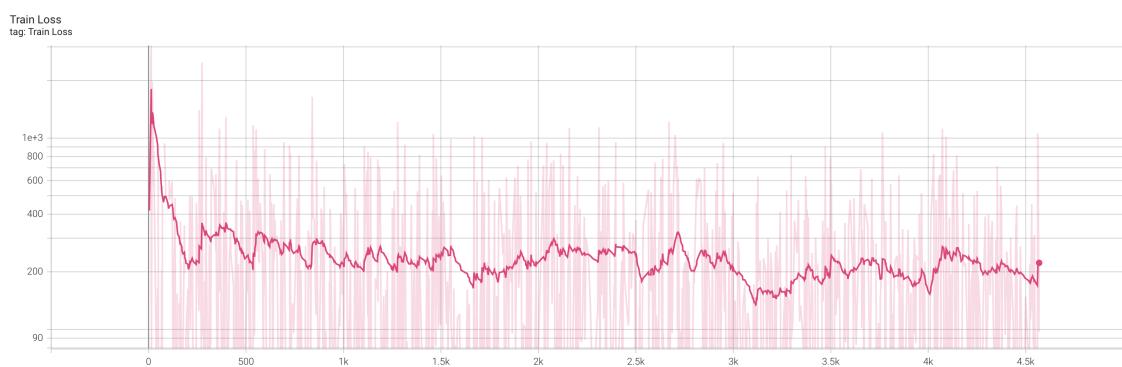


Figura 4.17: Experimento *Vision Transformer* - Training Loss

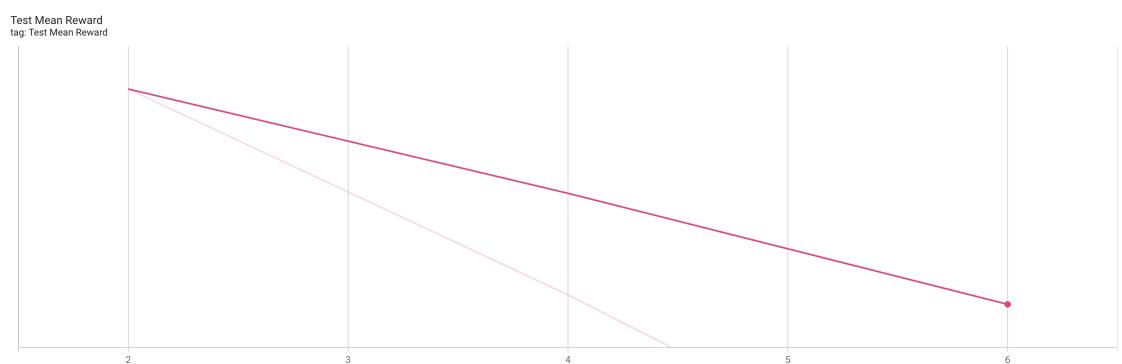


Figura 4.18: Experimento *Vision Transformer* - Testing reward mean



Figura 4.19: Experimento *Vision Transformer* - Episodes duration

Conclusiones

5

1. Conclusión 1.

2. Conclusión 2.

3. Conclusión 3.

Limitaciones y Perspectivas de Futuro

6

CAPÍTULO 6. LIMITACIONES Y
PERSPECTIVAS DE FUTURO

Apéndize A

A

El siguiente apéndize muestra el conjunto de experimentos totales realizados durante la ejecución del proyecto. Cada uno de los experimentos se puede encontrar dentro de la carpeta runs/ del código del proyecto en el repositorio de [GitHub](#).

1. Actor-Critic-ac-continuous-action
2. Actor-Critic-ac-discourage
3. Actor-Critic-ac-discourage-reward-1
4. Actor-Critic-ac-discourage-reward-2
5. Actor-Critic-ac-discourage-stop-action
6. Actor-Critic-ac-no-rewards-till-complete
7. Actor-Critic-ac-no-rewards-till-complete-divided
8. Actor-Critic-ac-no-rewards-till-complete-full-training
9. Actor-Critic-ac-reduce-reward-by-four
10. Actor-Critic-ac-reward-2-divide-rewards-lr-e6
11. Actor-Critic-ac-reward-2-min-3-disc-none-only-1000-steps
12. Actor-Critic-ac-reward-2-min-actions
13. Actor-Critic-ac-reward-2-min-actions-3-discount-none-only
14. Actor-Critic-ac-reward-2-normalized-dataframe
15. Actor-Critic-ac-reward-2-normalized-dataframe-lr-e7
16. Actor-Critic-ac-reward-2-rew-by-two-stop-with-none
17. Actor-Critic-ac-reward-2-rew-by-two-stop-with-none-only
18. Actor-Critic-ac-vit-encoder
19. Actor-Critic-no-sigmoid-state-value-adam-l2-reg-30steps-per-image-no-limit

APÉNDICE A. APÉNDIZE A

20. Actor-Critic-no-sigmoid-state-value-adam-l2-reg-30steps-per-image-stop-action
21. Actor-Critic-softmax-state-value-adam-l2-reg-30steps-per-image-stop-action-is
22. Actor-Critic-softmax-state-value-rmsprop-30steps-per-image-stop-action-is-non
23. Actor-Critic-v1
24. Actor-Critic-v2
25. Policy gradient normalized image rewards
26. Policy gradient normalized image rewards v2
27. Policy gradient recompensa 10
28. softmax-state-value

Bibliografía

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., y Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Haarnoja, T., Zhou, A., Abbeel, P., y Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290.

Luo, W., Sun, P., Zhong, F., Liu, W., Zhang, T., y Wang, Y. (2019). End-to-end active object tracking and its real-world deployment via reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 42(6):1317–1332.

Ridnik, T., Ben-Baruch, E., Noy, A., y Zelnik-Manor, L. (2021). Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*.

Simonyan, K. y Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., y Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.