Universidad de Valladolid

Escuela de Ingeniería Informática

Identificación morfológica de galaxias en el espacio profundo mediante Deep Learning

Grado en Ingeniería Informática
Mención en Computación

Autor:
Enrique Carpintero Gutierrez

Tutores:
Benjamin Sahelices Fernandez

Resumen

El trabajo expuesto en este documento es un proyecto de investigación en el que se pretende realizar un estudio de la tecnología asociada al Deep Learning y su posterior aplicación en el campo de la Astro-informática automatizando la clasificacion morfológica e identificación del contorno de galaxias de espacio profundo.

Debido a las necesidades particulares de este campo se ha profundizado en el estudio de las redes neuronales convolucionales y como framework de desarrollo de los experimentos se ha utilizado Fastai. Los experimentos se han realizado de forma incremental modificando diferentes parámetros y configuraciones y realizando estudios comparativos entre los mismos con el fin de obtener el mejor resultado posible.

Abstract

This paper is a research project in which a study of the technology associated with Deep Learning is carried and its subsequent application in the field of Astro-informatics by automating the morphological classification and identification of the contour of deep space galaxies.

Due to the particular needs of this field, an emphasis has been made in the study of convolutional neural networks and Fastai has been used as a framework for the development of the experiments. The experiments have been carried out incrementally, modifying different parameters and configurations and carrying out comparative studies between them in order to obtain the best possible result.

Índice general

1. Introducción y Objetivos

1.1. Motivación

Gracias a los increíbles avances tecnológicos que se están dando en los últimos años en el ámbito del aprendizaje automático empieza a ser cada vez mas viable y eficiente la aplicación de técnicas de Machine Learning como solución a un numero cada vez mayor de problemas.

Uno de estos problemas es el tratamiento de imágenes, ya sea para identificar ciertos elementos ,delimitar la imagen en diferentes segmentos o incluso la clasificacion en base a ciertas características. Una disciplina que depende mayoritariamente de datos compuestos por imágenes para su trabajo es la astro-informática.

Esta disciplina puede verse altamente beneficiada por la incorporación de técnicas de aprendizaje automático y Deep Learning al tratamiento de las imágenes que recogen del cosmos. agilizara los procesos y permitiría liberar tiempo a los investigadores.

1.2. Objetivos

Los objetivos principales a tratar en este trabajo son:

- Estudiar e investigar las tecnologías involucradas en el ámbito del Deep Learning, particularmente de las redes neuronales convolucionales, y sus medios de aplicación y frameworks donde llevar a cabo el desarrollo de modelos.
- Estudiar e investigar el ámbito de la Astro-informática, el modo en el que se opera, y los desafíos a los que se enfrentan.
- Aplicar los conocimientos adquiridos sobre Deep Learning sobre 2 problemas concretos en el ámbito de la Astro-informática, la clasificacion morfológica y la identificación del contorno de galaxias. Se intentara automatizar la tarea que se realiza generalmente de forma manual mediante el uso de redes neuronales obteniendo el mejor resultado posible.

1.3. Estructura del Trabajo

- **Capitulo 2 Planificación :**Presenta la forma en la que se ha realizado la gestión y desarrollo del trabajo, las actividades realizadas y una planificación temporal de recursos para abarcar el proyecto.
- **Capitulo 3 Astronomía e Astro-informática :** Introducción a la Astro-informática. Explica los diferentes instrumentos que se utilizan para analizar el cosmos así como conceptos básicos y desafíos de la disciplina.También incluye las herramientas utilizadas referentes a este ámbito en el trabajo.
- **Capitulo 4 Redes Neuronales y Deep Learning :** Introducción a las redes neuronales y el Deep Learning, explica el funcionamiento de sus algoritmos,diferentes tipos de redes neuronales y los elementos que las componen. También incluye las herramientas utilizadas referentes a este ámbito en el trabajo.
- **Capitulo 5 Clasificacion morfológica de galaxias en el espacio profundo** Incluye la experimentación realizada para la clasificacion morfológica de galaxias, el diseño de la investigación, cada uno de los elementos principales de cada experimento y sus resultados
- **Capitulo 6 Identificación del contorno de galaxias en el espacio profundo** Incluye la experimentación realizada para la identificación del contorno de galaxias, el diseño de la investigación, cada uno de los elementos principales de cada experimento y sus resultados
- **Capitulo 7 Conclusiones** Se presentan las conclusiones a las que se ha llegado al realizar el trabajo fin de grado y posibles lineas de investigación futura.

2. Planificación

En este capítulo se expone el plan de proyecto. En primer lugar, se hablará de la metodología seguida para el desarrollo del proyecto. Seguidamente, se expondrán las restricciones y la gestión de los riesgos. A continuación, se definirá la planificación inicial del proyecto

2.1. Metodología

La metodología utilizada en el proyecto es el Proceso Unificado con ciertas adaptaciones debido a la naturaleza del proyecto, el cual esta mas enfocado a la investigación.

2.1.1. Proceso Unificado

El **Proceso Unificado (*Unified Proccess* o UP)** es una de las metodologías de desarrollo de software más extendidas.Se define como un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software,áreas de aplicación,tipos de organizaciones y diferentes niveles de aptitud y tamaños de proyectos. Se distingue por seguir una serie de 'buenas prácticas':

- Iterativo e incremental, enfocado en minimizar los riesgos.
- Centrado en la arquitectura.
- Comprobar continuamente la calidad del software.
- Controlar cambios en el software.

El ciclo de vida de un proyecto según UP está compuesto por dos dimensiones. Verticalmente se representan las principales disciplinas del desarrollo, mientras que las fases del proyecto con respecto del tiempo están en horizontal, las cuales son:

- **Inicio:** se establece el contexto de negocio, los principales requisitos, la planificación y una primera evaluación de los riesgos.
- **Elaboración:** se desarrolla la mayor parte del análisis y se construye una arquitectura del sistema que pueda ser validada para minimizar los riesgos.
- **Construcción:** la fase más larga y con mayor número de iteraciones en la que se construye el sistema.

- **Transición:** se dispone el producto desarrollado al usuario y se valida el alcance del proyecto.

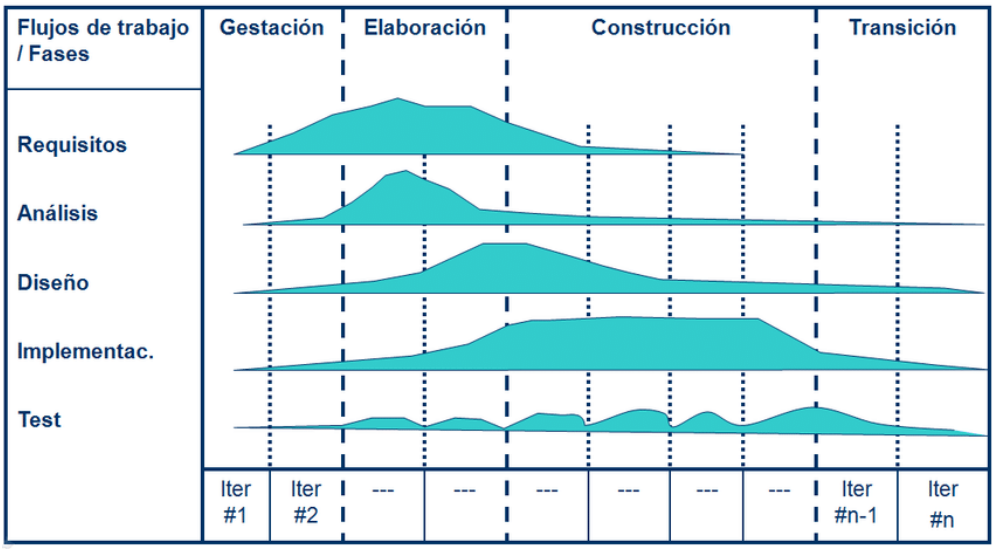


Figura 2.1.: Ciclo de vida del desarrollo de software UP

Fuente: https://www.researchgate.net/figure/Flujos-de-trabajo-del-proceso-unificado_fig1_279751761

2.2. Riesgos

R1	**Falta de disponibilidad del alumno**
Descripción	Incapacidad de continuar con el proyecto debido a una enfermedad o por trabajo
Probabilidad	Baja
Impacto	Alto
Plan de acción	Realizar nuevamente la planificación
Plan de contingencia	Realizar un seguimiento de la disponibilidad del alumno

Tabla 2.1.: Descripción del riesgo R1

R2	**Falta de disponibilidad de los recursos software**
Descripción	Incapacidad de poder continuar con el desarrollo por falta de disponibilidad de las herramientas o tecnologías utilizadas
Probabilidad	Baja
Impacto	Alto
Plan de acción	Encontrar el problema de la falta de disponibilidad y solucionarlo o utilizar otro software
Plan de contingencia	Disponer de las tecnologías actualizadas y configuradas

Tabla 2.2.: Descripción del riesgo R2

R3	**Falta de disponibilidad de los recursos hardware**
Descripción	Incapacidad de poder continuar con el desarrollo por falta de disponibilidad del equipo
Categoría	Riesgo de proyecto
Probabilidad	Baja
Impacto	Alto
Plan de acción	Continuar con el desarrollo en otro equipo
Plan de contingencia	Disponer de otro equipo preparado para poder continuar con el proyecto

Tabla 2.3.: Descripción del riesgo R3

R4	**Perdida del trabajo realizado**
Descripción	Debido a distintas circunstancias como el fallo en el hardware o software es posible que se pierdan los datos
Probabilidad	Baja
Impacto	Alto
Plan de acción	Crear copias de seguridad
Plan de contingencia	Disponer de un lugar seguro donde almacenar las copias

Tabla 2.4.: Descripción del riesgo R4

2.3. Planificación inicial

El proyecto se ha dividido en 7 iteraciones y se dedicaran unas 430 horas en total. El parón entre el 20 de Diciembre y el 8 de enero se debe a las festividades navideñas.

Iteración	Fase	Fecha inicio	Fecha fin	Horas
1	Inicio	18/10/2021	01/11/2021	50
2	Elaboración	02/11/2021	16/11/2021	40
3	Elaboración	17/11/2020	05/12/2021	60
4	Construcción	06/12/2020	20/12/2021	50
5	Construcción	08/01/2022	10/02/2022	100
6	Construcción	11/02/2022	17/03/2022	100
7	Transición	18/03/2022	05/04/2022	50
Total		18/10/2021	28/03/2022	430

Tabla 2.5.: Planificación inicial

La lista de tareas de cada iteración es la siguiente:

- **Iteración 1:**
  - Establecer la metodología de trabajo.
  - Gestión de riesgos.
  - Planificación inicial.
- **Iteración 2:**
  - Investigación sobre contexto astro-informático
  - Aprendizaje sobre tratamiento de datos astronomicos
  - Obtención de datos para la investigación
- **Iteración 3:**
  - Estudio de redes neuronales y Deep Learning
  - Aprendizaje sobre herramientas software
  - Desarrollar el diseño de la experimentación.
- **Iteración 4:**
  - Instalación y preparación de las herramientas software necesarias
  - Tratamiento del conjunto de datos a utilizar

- **Iteración 5:**
  - Experimentación con clasificacion morfológica de galaxias.
- **Iteración 6:**
  - Experimentación con obtención de contorno de galaxias mediante regresión
- **Iteración 7:**
  - Obtencion de conclusiones.
  - Escritura de la memoria.

3. Astronomia e Astro-informatica

3.1. Astronomia

3.1.1. Introduccion

La astronomía se define como el estudio de los objetos que se encuentran más allá de nuestro planeta Tierra y los procesos por los cuales estos objetos interactúan entre sí. También es el intento de la humanidad para organizar lo que aprendemos en una historia clara del universo, desde el instante de su nacimiento en el Big Bang hasta el momento presente.

En astronomía se trabaja con distancias a una escala inimaginable, lo que hace necesario tomar ciertas medidas para facilitar la comprensión y el estudio. Se utilizan dos enfoques que hacen que lidiar con números tan grandes sea un poco más fácil. Primero, se usa un sistema para escribir números grandes y pequeños llamado notación científica. En notación científica, si se desea escribir un número como 500.000.000, lo expresas como 510^8. El número pequeño elevado después del 10, llamado exponente, realiza un seguimiento del número de lugares que tuvimos que mover el punto decimal a la izquierda para convertir $500,000,000$ a 5.La segunda forma en la que se trata de mantener los números simples es usar un conjunto consistente de unidades: el métrico Sistema Internacional de Unidades.

Una unidad común que usan los astrónomos para describir distancias en el universo es un año luz, que es la distancia que la luz viaja durante un año. El hecho de que la luz siempre viaja a la misma velocidad, y su velocidad resulta ser la velocidad más rápida posible en el universo, la convierten en un buen estándar para realizar un seguimiento de las distancias.

La luz viaja a un ritmo asombroso de 3105 kilómetros por segundo (km / s), lo que hace un año luz $9,4610^12$ kilómetros. Seria intuitivo pensar que una unidad tan grande podría llegar de sobra a la estrella más cercana, pero las estrellas son mucho más remotas de lo que nuestra imaginación podría hacernos creer. Incluso la estrella más cercana está a 4,3 años luz de distancia, más de 40 billones de kilómetros. Otras estrellas visibles al ojo humano están a cientos o miles de años luz de distancia.

Figura 3.1.: Nebulosa de la Helice, de las nebulosas mas cercanas situada a 650 años luz

Fotografía obtenida en 2012 con el telescopio VISTA de 4,1 metros del ESO, en Chile

Hay otra razón por la que la velocidad de la luz es una unidad de distancia tan natural para los astrónomos. Información sobre el universo llega a nosotros casi exclusivamente a través de varias formas de luz, y toda esa luz viaja en el velocidad de la luz, es decir, 1 año luz cada año. Esto establece un límite en la rapidez con la que podemos aprender acerca de los eventos en el universo.

Si una estrella está a 100 años luz de distancia, la luz que vemos esta noche salió de esa estrella hace 100 años y es solo esta llegando ahora a nuestro planeta. Lo más pronto que podemos saber sobre cualquier cambio en esa estrella es 100 años después de que suceda. Para una estrella a 500 años luz de distancia, la luz que detectamos esta noche se fue hace 500 años.

3.1.2. La luz como onda

Gran mayoria de la información que los astrónomos utilizan para estudiar el cosmos nos llega en forma de luz. Codificado en la luz y otros tipos de radiación que nos llegan de los objetos en el universo hay una amplia gama de información sobre cómo son esos objetos y cómo funcionan.

La luz visible y otras radiaciones que recibimos de las estrellas y planetas se generan mediante procesos a nivel atómico, por cambios en la forma en que las partes de un átomo interactúan y se mueven. Un átomo típico consta de varios tipos de partículas, algunas de las cuales no solo

tienen masa sino una propiedad adicional llamada carga eléctrica. En el núcleo de cada átomo hay protones, que están cargados positivamente; fuera del núcleo estan los electrones, que tienen carga negativa.

La teoría de Maxwell se ocupa de estas cargas eléctricas y sus efectos, especialmente cuando se mueven. En la cercanía de una carga electrónica, otra carga siente una fuerza de atracción o repulsión, las cargas opuestas se atraen y las cargas iguales se repelen. Cuando las cargas no están en movimiento, solo observamos esta atracción o repulsión eléctrica. Sin embargo, si las cargas están en movimiento , entonces se mide otra fuerza llamada magnetismo.

Los experimentos con cargas eléctricas demostraron que el magnetismo era el resultado de partículas cargadas en movimiento. A veces, el movimiento es claro, como en las bobinas de alambre pesado que hacen un electro imán. Otras veces, es más sutil, como en el tipo de imán que compras en una ferretería, en el que muchos de los electrones dentro de los átomos giran aproximadamente en la misma dirección; es la alineación de los movimientos de las cargas lo que hace que el material se vuelva magnético.

Maxwell analizó lo que sucedería si las cargas eléctricas estuvieran oscilando (moviéndose constantemente hacia adelante y hacia atrás) y descubrió que el patrón resultante de campos eléctricos y magnéticos se esparciría y viajaría rápidamente a través de el espacio. Algo similar sucede cuando una gota de lluvia golpea la superficie del agua o una rana salta a un estanque,la perturbación se mueve hacia afuera y crea un patrón que llamamos onda.

Maxwell pudo calcular la velocidad a la que una perturbación electromagnética se mueve a través del espacio. Descubrió que es igual a la velocidad de la luz, que había sido medida experimentalmente. Sobre esa base, especuló que la luz formaba parte de una familia de posibles perturbaciones electromagnéticas llamadas radiaciones electromagnéticas, una conclusión que se confirmó nuevamente en experimentos de laboratorio.

Los campos eléctricos y magnéticos cambiantes en la luz son similares a las ondas que se pueden establecer en una piscina tranquila de agua. En ambos casos, la perturbación viaja rápidamente hacia afuera desde el punto de origen y puede usar su energía para perturbar objetos más lejanos. Las ondas generadas por partículas cargadas difieren de las ondas formadas en el agua en que no necesitan de un medio físico como por ejemplo el agua o el aire para transportarse y que siempre se mueven a la misma velocidad, la velocidad de la luz.No importa a partir de que o donde las ondas electromagnéticas se generan y no importa qué otras propiedades tengan, cuando están moviéndose (y no interactuando con la materia), se mueven a la velocidad de la luz.

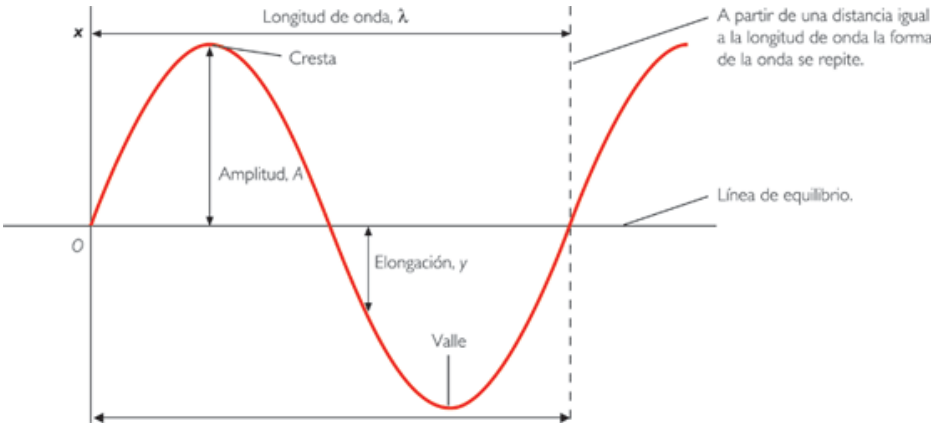


Figura 3.2.: Caracteristicas de una onda electromagnetica

Fuente: https://sites.google.com/site/elgrandiosorincondelafisica/onda

Las ondas son un fenómeno repetitivo, el patrón de perturbación se repite de forma cíclica. Por tanto, cualquier movimiento ondulatorio puede caracterizarse por una serie de crestas y valles. Pasar de una cresta a la siguiente cresta completa un ciclo. La longitud horizontal cubierta por un ciclo se llama longitud de onda. Para la luz visible, nuestros ojos perciben diferentes longitudes de onda como diferentes colores, el rojo por ejemplo es la longitud de onda visible mas larga, y el violeta la más corta.También podemos caracterizar diferentes ondas por su frecuencia, el número de ciclos de onda que pasan por segundo.

3.1.3. La luz como partícula

El modelo de onda electromagnética de la luz formulado por Maxwell fue uno de los grandes triunfos de ciencia del siglo XIX.Sin embargo, a principios del siglo XX, experimentos más sofisticados habían revelado que la luz se comporta de ciertas formas que no pueden ser explicadas por el modelo de onda. A regañadientes, los físicos tuvieron que aceptar que a veces la luz se comporta más como una partícula que como una onda. Esta partícula se conoce actualmente con el nombre de foton.

El hecho de que la luz se comporte como una onda en ciertos experimentos y como una partícula en otros fue un hecho muy sorprendente. Después de todo, nuestro sentido común dice que las ondas y las partículas son conceptos opuestos. Una onda es una perturbación repetida que, por su propia naturaleza, no está en un solo lugar, sino que se extiende. Una partícula, por otro lado, es algo que puede estar en un solo lugar en un momento dado. Por extraño que parezca innumerables experimentos ahora confirman que la radiación electromagnética a veces puede comportarse como una onda y otras veces como una partícula. La confusión que provocaba esta dualidad de luz onda-partícula finalmente se resolvió con la introducción de una teoría más complicada de ondas y partículas, ahora llamada mecánica cuántica que no trataremos en este trabajo por su gran

extensión y complejidad.

3.1.4. El espectro electromagnético

Hemos visto como la luz es un tipo de radiacion electromagnetica, estas radiaciones pueden tener longitudes de onda y frecuencias muy diferentes entre si y los objetos del universo envían una enorme gama de radiación electromagnética. Los científicos llaman a este rango el espectro electromagnético, que se ha dividido en varias categorías:

1. **Rayos Gamma:** Radiación electromagnética con las longitudes de onda más cortas, no más de 0,01 nanómetros.Debido a que los rayos gamma transportan mucha energía, pueden ser peligrosos para tejidos vivos. La radiación gamma se genera en las profundidades del interior de las estrellas, así como por algunas de las más fenómenos violentos en el universo, como la muerte de estrellas y la fusión de cadáveres estelares. Los rayos gamma que llegan a la Tierra son absorbidos por nuestra atmósfera antes de que lleguen a la superficie.

2. **Rayos X:** Radiación electromagnética con longitudes de onda entre 0,01 nanómetros y 20 nanómetros. Al ser más enérgicos que la luz visible, los rayos X pueden penetrar los tejidos blandos pero no los huesos, por lo que nos permiten hacer imágenes de las sombras de los huesos dentro de nosotros.Los rayos x son detenidos por la gran cantidad de átomos en la atmósfera de la Tierra con los que interactúan.

3. **Rayos Ultravioleta:** Radiación electromagnética con longitudes de onda entre 20 nanómetros y 400 nanómetros.La radiación ultravioleta está bloqueada principalmente por la capa de ozono de la atmósfera de la Tierra, pero una pequeña fracción de los rayos ultravioleta de nuestro Sol penetran y provocan quemaduras solares o, en casos extremos de sobreexposición, cáncer de piel en los seres humanos.

4. **Luz visible:** Radiación electromagnética con longitudes de onda entre aproximadamente 400 y 700 nanómetros. Estas son las ondas que la visión humana puede percibir, esta es también la banda del espectro electromagnético que llega más fácilmente a la superficie de la Tierra.

5. **Rayos Infrarrojos:** Radiación electromagnética con longitudes de onda entre aproximadamente 700 y 10^3 nanómetros. Una lámpara de calor irradia principalmente radiación infrarroja, y las terminaciones nerviosas de nuestra piel son sensibles a esta banda del espectro electromagnético. Las ondas infrarrojas son absorbidas por moléculas de agua y dióxido de carbono, que están más concentrados en la parte baja de la atmósfera de la Tierra.

6. **Rayos de Microondas:** Radiación electromagnética con longitudes de onda entre aproximadamente 10^6 y 10^9 nanómetros.

7. **Radio:** Radiación electromagnética con longitudes de onda superiores a 10^9 nanómetros.Hay muchos tipos de radiaciones incluidas en esta categoria. Entre las más conocidas se encuentran las ondas de radar, que son utilizadas por los oficiales de tráfico para determinar la velocidad de los vehículos, y ondas de radio AM, que fueron las primeras a ser desarrollado para radiodifusión. Las longitudes de onda de estas diferentes categorías van desde más de un metro hasta cientos de metros, y otras radiaciones de radio pueden tener longitudes de onda de varios kilómetros.

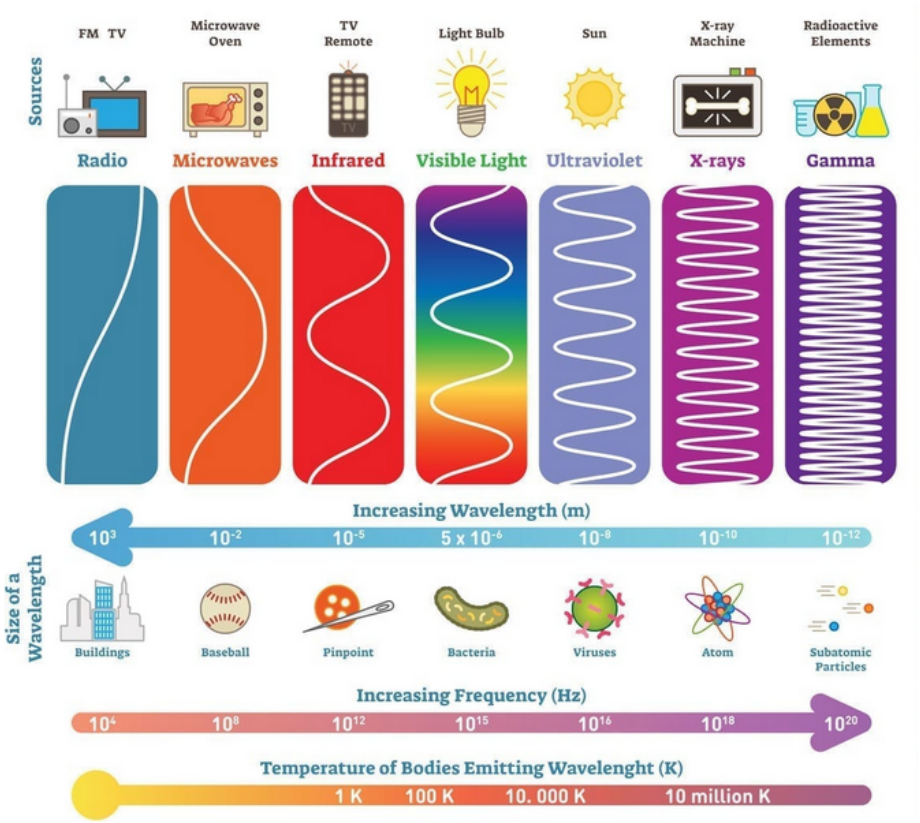


Figura 3.3.: Espectro Electromagnetico

Algunos objetos astronómicos emiten principalmente radiación infrarroja, otros principalmente luz visible y otros en su mayoría radiación ultravioleta. ¿Qué determina el tipo de radiación electromagnética emitida por el Sol, las estrellas y otros objetos astronómicos densos? La respuesta a menudo resulta ser su temperatura.

A nivel microscópico, todo en la naturaleza está en movimiento. Un sólido está compuesto por moléculas y átomos en vibración continua, se mueven hacia adelante y hacia atrás en un lugar, pero su movimiento es demasiado pequeño para que nuestros ojos lo puedan percibir.

Un gas consta de átomos y / o moléculas que vuelan libremente a alta velocidad, continuamente

chocando entre sí y bombardeando la materia circundante. Cuanto más caliente esté el sólido o el gas, más rápido el movimiento de sus moléculas o átomos. La temperatura de algo es, por tanto, una medida de la energia de movimiento media de las partículas que lo componen.

Este movimiento a nivel microscópico es responsable de gran parte de la radiación electromagnética en la Tierra y en el universo. A medida que los átomos y las moléculas se mueven y chocan, o vibran en su lugar, sus electrones emiten radiación electromagnética. Las características de esta radiación están determinadas por la temperatura de aquellos átomos y moléculas.

En un material caliente las partículas individuales vibran en su lugar o se mueven rápidamente por lo que las ondas emitidas son, en promedio, más enérgicas. En material muy frío, las partículas tienen movimientos atómicos y moleculares de baja energía y, por lo tanto, generar ondas de menor energía.

3.1.5. Instrumentos Astronómicos

Hay tres componentes básicos de un sistema moderno para medir la radiación de fuentes astronómicas. Primero, esta el telescopio, que funciona como un especie de recipiente para recolectar luz visible o radiación en otros longitudes de onda. Así como puedes atrapar más lluvia con un cubo de basura que con una taza, cuanto mayor es el tamaño del telescopio mayor es la cantidad de luz que puede captar.

En segundo lugar, hay un instrumento adjunto al telescopio que clasifica la radiación entrante por longitud de onda. A veces, la clasificación es bastante burda. Por ejemplo, simplemente podríamos querer separar la luz azul de la luz roja para poder determinar la temperatura de un estrella.Pero en otras ocasiones, queremos ver líneas espectrales individuales para determinar de qué está hecho un objeto, o para medir su velocidad.

En tercer lugar, necesitamos algún tipo de detector,un dispositivo que detecta la radiación en las regiones de longitud de onda que hemos elegido y registra permanentemente las observaciones.

3.1.5.1. Telescopios

La historia del desarrollo de los telescopios astronómicos trata sobre cómo se han aplicado nuevas tecnologías. para mejorar la eficiencia de estos tres componentes básicos: los telescopios, el dispositivo de clasificación de longitud de onda y los detectores.

Las funciones más importantes de un telescopio son recolectar la luz tenue de una fuente astronómica y enfocar toda la luz en un punto o una imagen. La mayoría de los objetos de interés para los astrónomos son extremadamente débiles, por lo que cuanta más luz podamos recolectar, mejor podremos estudiar tales objetos.

Los telescopios que recolectan radiación visible usan una lente o espejo para recolectar la luz. Otros tipos de telescopios pueden utilizamos dispositivos colectores que se ven muy diferentes de las lentes y espejos con los que estamos familiarizados, pero que cumplen la misma función. En todos los tipos de telescopios, la capacidad de captación de luz está determinada por el área del dispositivo que actúa como el recipiente de recolección de luz. Dado que la mayoría de los telescopios tienen espejos o lentes, podemos comparar su poder de captación de luz comparando el diámetro de la abertura a través de la cual la luz viaja o se refleja.

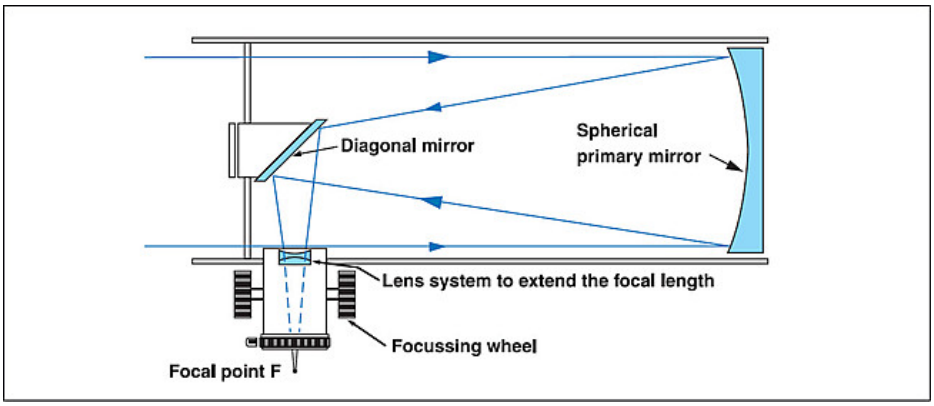


Figura 3.4.: Componentes de un telescopio

https://www.bresser.de/c/es/support/teleskop-fibel/telescopios/

Después de que el telescopio forme una imagen, necesitamos alguna forma de detectarla y registrarla para poder medir, reproducir y analizar la imagen de varias formas. Antes del siglo XIX, los astrónomos simplemente veían imágenes con sus ojos y escribían descripciones de lo que vieron. Esto fue muy ineficiente y no condujo a un registro a largo plazo muy confiable.

En el siglo XIX se generalizó el uso de la fotografía. En aquellos días, las fotografías eran un registro químico de una imagen en una placa de vidrio especialmente tratada. Hoy en día, la imagen se detecta generalmente con sensores similares a los de las cámaras digitales, grabados electrónicamente y almacenados en computadoras. Este registro permanente se puede utilizar para estudios detallados y cuantitativos. Los astrónomos profesionales rara vez miran a través del grandes telescopios que utilizan para sus investigaciones.

Además de recoger la mayor cantidad de luz posible, los astrónomos también quieren tener las imágenes más nítidas posibles. La resolución se refiere a la precisión de los detalles presentes en una imagen: es decir, las características más pequeñas que se pueden distinguido. Los astrónomos siempre están ansiosos por distinguir más detalles en las imágenes que estudian.

Un factor que determina qué tan buena será la resolución es el tamaño del telescopio. Aberturas más grandes producen imágenes más nítidas. Sin embargo, hasta hace muy poco tiempo, los telescopios de luz visible e infrarrojos en la superficie de la Tierra no podían producir imágenes tan nítidas como deberían.

El problema es la atmósfera de nuestro planeta, que es turbulenta. Contiene muchas gotas o cúmulos de gas a pequeña escala que varían en tamaño desde unos centimetros a varios metros. Cada cúmulo tiene una temperatura ligeramente diferente a la de su vecino, y cada cumulo actúa como una lente, refractando el camino de la luz por una pequeña cantidad. Esta flexión cambia ligeramente la posición donde cada rayo de luz finalmente llega al detector en un telescopio.

Los cúmulos de aire están en movimiento, siendo constantemente impulsados a través de la trayectoria de la luz del telescopio por vientos, a menudo en diferentes direcciones a diferentes altitudes. Como resultado, el camino seguido por la luz es constantemente cambiando.

3.1.5.2. Detectores

Después de que un telescopio recolecta radiación de una fuente astronómica, la radiación debe detectarse y medirse.Antes de que la luz llegue al detector los astrónomos normalmente utilizan algún tipo de instrumento para clasificar la luz según la longitud de onda. El instrumento puede ser algo tan simple como filtros de colores, que transmiten luz dentro de un rango específico de longitudes de onda. Un plástico rojo transparente es un ejemplo cotidiano de un filtro que transmite solo la luz roja y bloquea los otros colores. Después de que la luz pase a través de un filtro, forma una imagen que los astrónomos pueden utilizar para medir el brillo y el color de los objetos.

A lo largo de la mayor parte del siglo XX, la película fotográfica o las placas de vidrio sirvieron como el principal detector astronómico , ya sea para fotografiar espectros o imágenes directas de objetos celestes.

Los astrónomos hoy en dia tienen detectores electrónicos mucho más eficientes para registrar imágenes astronómicas más a menudo. Estos son dispositivos de carga acoplada (CCD), que son similares a los detectores utilizados en videocámaras o en cámaras digitales.

En un CCD, los fotones de radiación que golpean cualquier parte del detector generan una corriente de partículas cargadas (electrones) que se almacenan y cuentan al final de la exposición. Cada lugar donde se cuenta la radiación se llama píxel (elemento de imagen), y los detectores modernos pueden contar los fotones en millones de píxeles (megapíxeles o MP).

Debido a que los CCD normalmente registran entre el 60 y el 70 % de todos los fotones que los golpean, y los mejores CCD de silicio e infrarojos superan el 90 % de sensibilidad, podemos detectar objetos mucho más débiles. Entre estos hay muchas lunas alrededor de los planetas exteriores, planetas enanos helados más allá de Plutón y galaxias enanas.

Los CCD también proporcionan mediciones más precisas del brillo de los objetos astronómicos que la fotografía, y su salida es digital: en forma de números que pueden ir directamente a una computadora para su análisis.

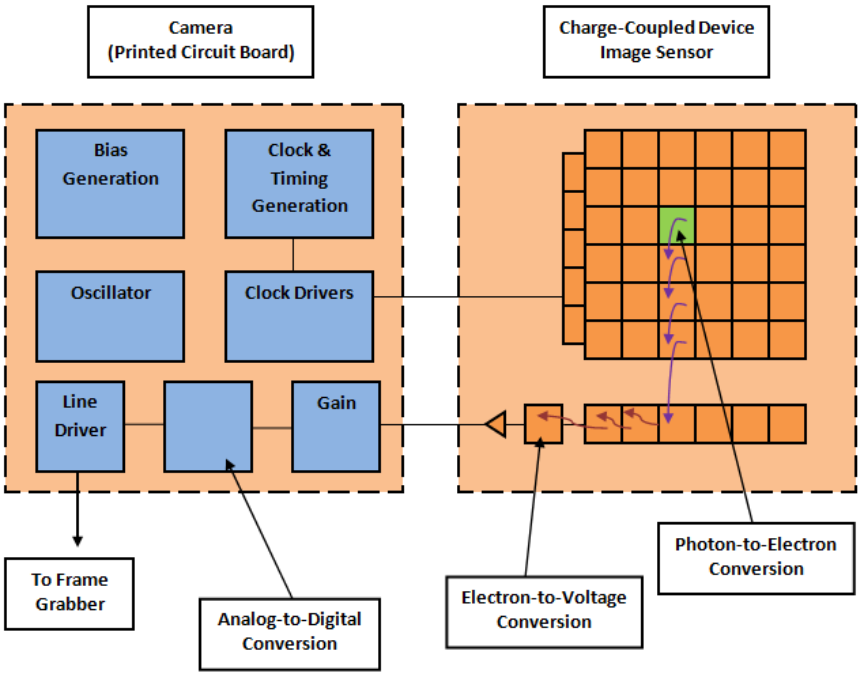


Figura 3.5.: Funcionamiento de un detector CCD

https://cecas.clemson.edu/cvel/auto/sensors/optical-image.html

3.1.5.3. Desafios en la obtencion de imagenes

Una vez obtenidas las imágenes los astrónomos se enfrentan a una serie de problemas, entre ellos están el corrimiento al rojo (redshift) y la función de dispersión de punto (PSF).

El **desplazamiento al rojo** es un fenómeno que afecta a las ondas electromagnéticas provenientes de un objeto emisor y que consiste en un enrojecimiento de la luz,es decir, las radiaciones emitidas experimentan un corrimiento hacia la parte menos energética (más roja) del espectro. Este cambio se puede deber a tres procesos físicos diferentes:

1. El emisor y el receptor se alejen entre sí (efecto Doppler)
2. El emisor se encuentre sometido a un campo gravitatorio más intenso que el receptor (desplazamiento al rojo gravitatorio).
3. La expansión del universo (desplazamiento al rojo cosmológico).

Cuando el receptor experimenta un campo gravitatorio más intenso o cuando el universo se contrae, entonces se produce el efecto contrario, el desplazamiento al azul. El desplazamiento al rojo se representa con la letra z. La variable z adopta valores positivos cuando se trata de un desplazamiento al rojo y negativos si se trata de un desplazamiento al azul.

Los astrónomos utilizan esta información para determinar la distancia a la que se encuentran los objetos estelares observados en las imágenes.

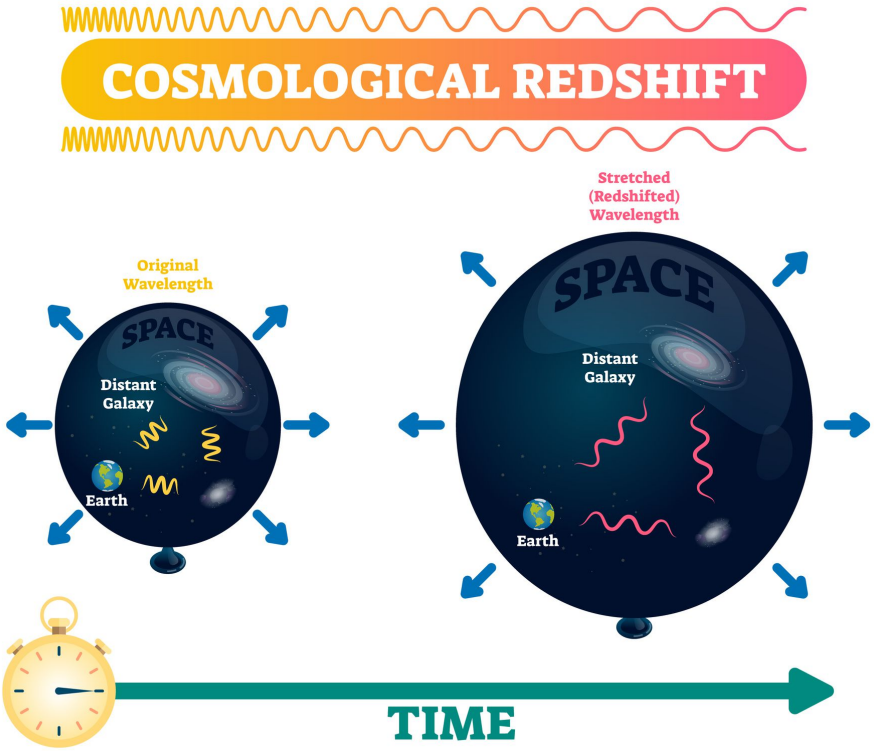


Figura 3.6.

https://www.greelane.com/es/ciencia-tecnolog%c3%ada-matem%c3%a1ticas/ciencia/what-is-redshift-3072290/

Cualquier sistema formador de imágenes introduce anomalías en el registro de la señal procedente del espacio objeto. La consecuencia más evidente es la difuminación de los contornos que definen las formas.

Cuando la fragmentación territorial ronda unas dimensiones del mismo orden de magnitud que la resolu- ción espacial del sensor utilizado, esto se traduce en una imagen confusa, en la cual los rasgos definitorios de los elementos a lo sumo se adivinan.

La función de suavizado característica del conjunto integrado por el propio sistema formador de imágenes y la atmósfera dispersante es la **Función de Dispersión Puntual PSF** (Point Spread Function).

En un sistema discreto como el de las imágenes digitales, la PSF es la imagen bidimensional de un punto del espacio objeto, que no es solamente el propio punto registrado en una celda sino un conjunto de valores asociados a las celdas vecinas.En las operaciones de restauración digital mediante deconvolución es esencial conocer la PSF que ha afectado la adquisición de la imagen.

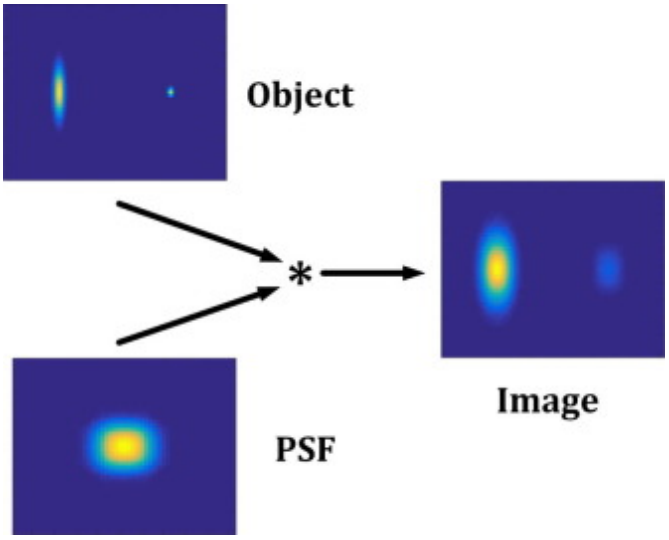


Figura 3.7.

hhttps://www.sciencedirect.com/science/article/abs/pii/S1350449516300160

3.1.5.4. Telescopio Espacial Hubble

En abril de 1990, se dio un gran salto en la astronomía con el lanzamiento del Telescopio Espacial Hubble (HST). Con una apertura de 2,4 metros, este es el telescopio más grande lanzado al espacio hasta ahora.Fue nombrado por Edwin Hubble, el astrónomo que descubrió la expansión del universo en la década de 1920.

Figura 3.8.: Telescopio espacial Hubble en orbita

https://magnet.xataka.com/un-mundo-fascinante/25-imagenes-para-celebrar-los-25-anos-del-telescopio-espacial-hubble

El HST es operado conjuntamente por el Goddard Space Flight Center de la NASA y el Space Telescope Science Institute en Baltimore. Fue el primer observatorio en órbita diseñado para ser atendido por astronautas y, a lo largo de los años desde su lanzamiento, se han realizado varias visitas para mejorar o reponer sus instrumentos iniciales y reparar algunos de los sistemas.

Con el Hubble, los astrónomos han obtenido algunas de las imágenes más detalladas de objetos astronómicos tanto de el sistema solar como de galaxias mucho mas distantes. Entre sus muchos grandes logros se encuentra el Hubble Ultra- Deep Field, una imagen de una pequeña región del cielo observada durante casi 100 horas. Contiene vistas de aproximadamente 10.000 galaxias, algunas de las cuales se formaron cuando el universo tenía solo un pequeño porcentaje de su edad actual.

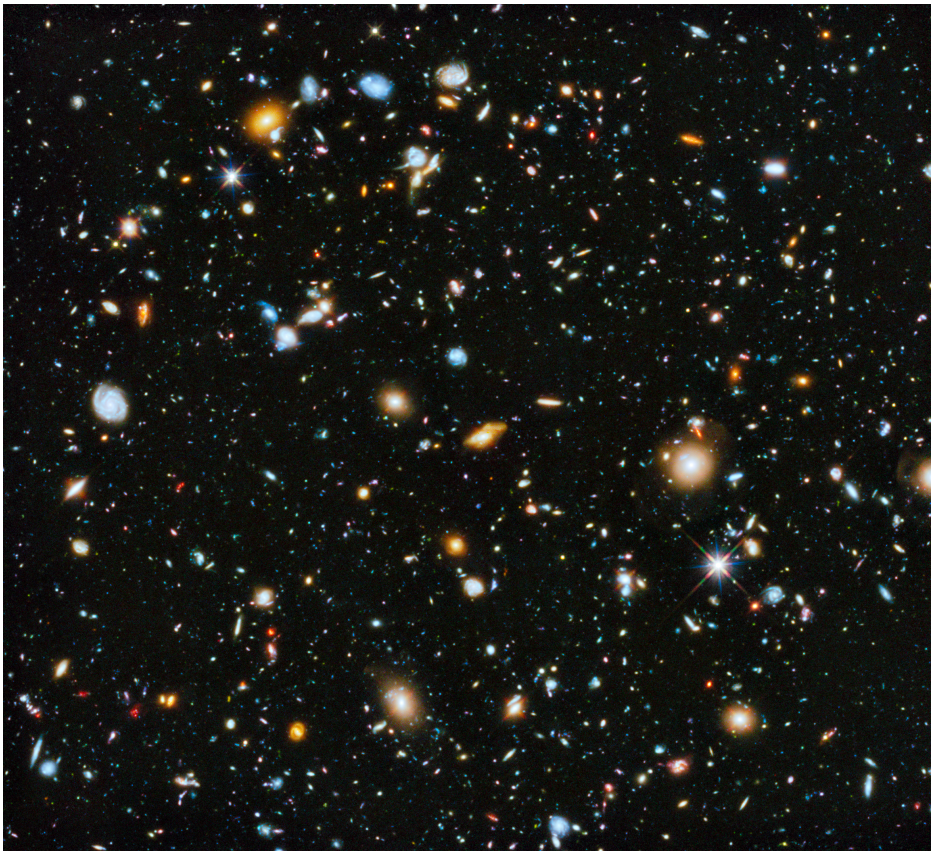


Figura 3.9.: Hubble Ultra Deep Field 2014

https://hubblesite.org/contents/media/images/2014/27/3380-Image.html

3.2. Astro-informatica

La astroinformática se centra principalmente en el desarrollo de herramientas, métodos, y aplicaciones de ciencia computacional, ciencia de datos, y estadística para la búsqueda y educación en astronomía orientada en datos.

Aunque el foco primario de la astro-informática reside en la gran colección digital mundialmente distribuida de bases de datos astronómicas, archivos de imagen, y herramientas de búsqueda, el campo reconoce también la importancia de los conjuntos de datos históricos —utilizando tecnologías modernas para preservar y analizar observaciones astronómicas históricas.

Hay muchas áreas de investigación involucradas con la astro-informática, como la minería de datos, el aprendizaje automático, la estadística, la visualización, la administración de datos científicos. La minería de datos y el aprendizaje automático juegan roles especialmente significativos.

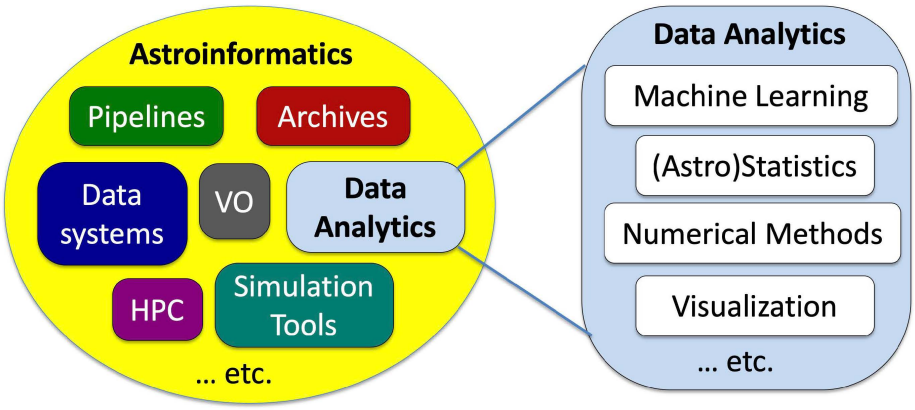


Figura 3.10.: Componentes de la Astro-informatica

http://astroinformatics.info/astroinfo

La cantidad de datos recogidos de estudios del cielo astronómico ha crecido de gigabytes a terabytes durante la década pasada y está pronosticado que crezca en la década próxima a centenares de petabytes. Este masivo incremento en la cantidad de datos nuevos tanto habilita como dificulta la realización eficaz de la investigación astronómica.

La utilización y estandarización de dichos datos, ha generado nuevos desafíos y prácticas en la astronomía junto con el advenimiento de Observatorios Virtuales agrupados en iniciativas como la Alianza del Observatorio Internacional (IVOA) para hacer frente a este nuevo escenario global.

Ello implica en la práctica, la necesidad por parte de los astrónomos de tener habilidades en la construcción de algoritmos para procesar un gran volumen de información, poder visualizar y analizar los datos en tiempo real, adoptar estrategias de análisis provenientes de diversas áreas del saber, combinar efectivamente la información proveniente de distintos observatorios astronómicos y poder almacenarlos eventualmente, entre otros.Esta infraestructura incluye bases de

datos, observatorios virtuales, computación de alto rendimiento (clusters y máquinas petaescala), computación distribuida (Grid, la nube, redes peer to peer), herramientas inteligentes de búsqueda y descubrimiento, e innovadores entornos de visualización

3.3. Herramientas para tratamiento de datos astronómicos

3.3.1. Sextractor

SExtractor (Source-Extractor) es un programa que construye un catálogo de objetos a partir de una imagen astronómica. Está particularmente orientado a la reducción de datos de estudios de galaxias a gran escala, pero también funciona bien en campos de estrellas moderadamente poblados. Alguna de sus principales características son:

1. Soporte para FITS de múltiples extensiones.
2. Velocidad: normalmente 1 Mpx / s con un procesador de 2 GHz.
3. Capacidad para trabajar con imágenes muy grandes (hasta 65k × 65k píxeles en máquinas de 32 bits, o 2G × 2G píxeles en máquinas de 64 bits), gracias al acceso a imágenes en búfer.
4. Desmezclamiento robusto de objetos extendidos superpuestos.
5. Filtrado de imágenes en tiempo real para mejorar la detectabilidad.
6. Fotometría de píxel a píxel en modo de imagen dual.
7. Manejo de mapas de peso y mapas de banderas.
8. Modo especial para escaneos fotográficos.

El análisis completo de una imagen está completamente automatizado realizándose dos pasadas a través de los datos. Durante la primera pasada, se construye un modelo del fondo del cielo y se calculan varias estadísticas globales. Durante la segunda pasada, los píxeles de la imagen se restan, filtran y segmentan sobre la marcha. Luego, las detecciones se desmembran, se podan y entran en la fase de medición. Finalmente, las cantidades medidas se escriben en el catálogo de salida, después de la comparación cruzada con una lista de entrada opcional.

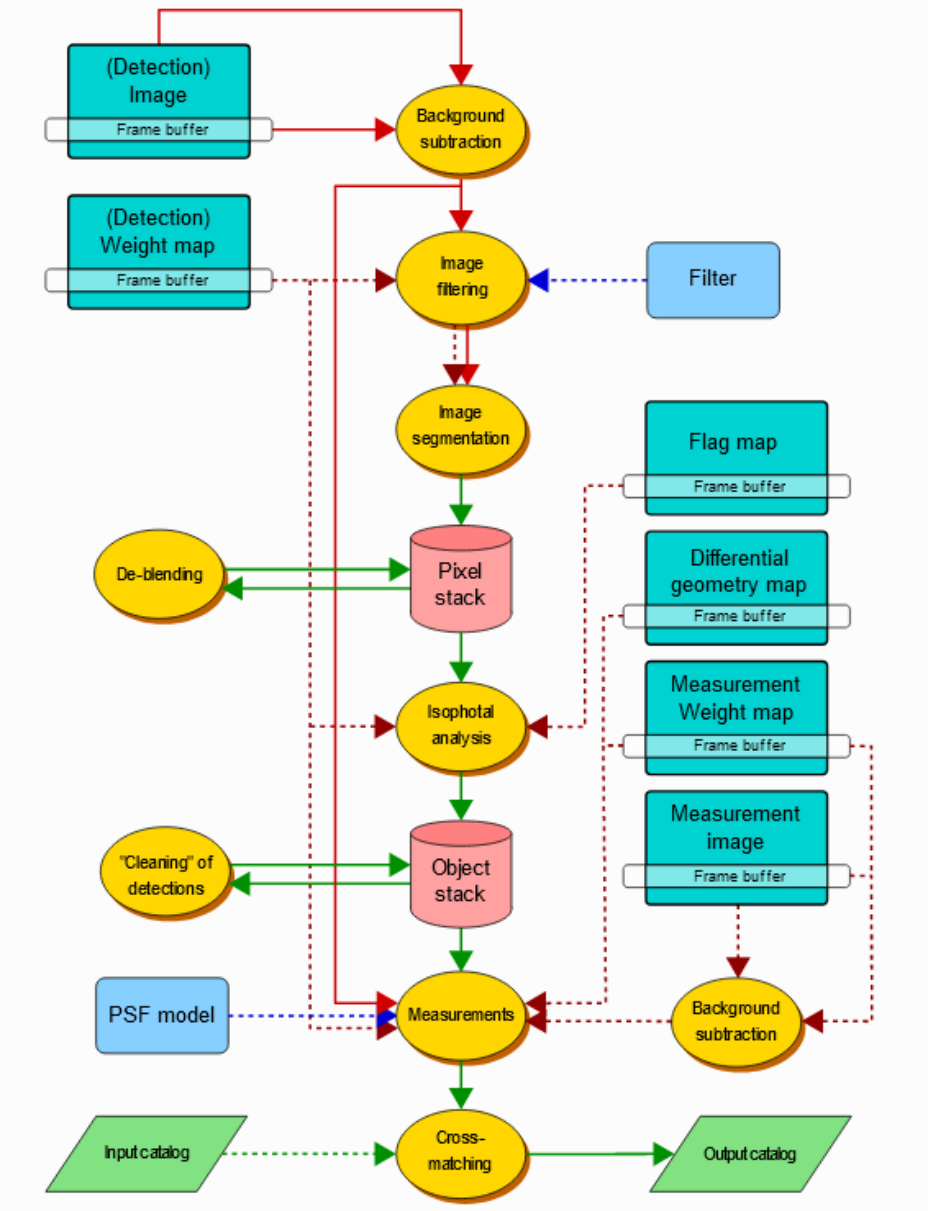


Figura 3.11.: Proceso completo de extraccion con Sextractor

Fuente: https://sextractor.readthedocs.io/en/latest/Processing.html

3.3.2. Aladin Desktop

Aladin Desktop permite al usuario visualizar y manipular imágenes astronómicas digitalizadas,superponer entradas de catálogos o bases de datos astronómicos y acceder de forma interactiva a datos e información relacionados de la base de datos de Simbad, el servicio VizieR y otros archivos de todos los objetos astronómicos conocidos en el campo.

Figura 3.12.: Logotipo Aladin Desktop

http://www.esep.pro/sites/esep/IMG/pdf/12_aladin-meudon.pdf

Creado en 1999 por CDS, Aladin Desktop se ha convertido en una herramienta de VO ampliamente utilizada capaz de abordar desafíos como localizar datos de interés, acceder y explorar conjuntos de datos distribuidos y visualizar datos de múltiples longitudes de onda.

El cumplimiento de los estándares VO existentes o emergentes, la interconexión con otras herramientas de visualización o análisis, la capacidad de comparar fácilmente datos heterogéneos son temas clave que permiten que Aladin sea una poderosa herramienta de exploración e integración de datos, así como un habilitador científico.Algunas de sus características principales son:

1. Más de 20 000 recopilaciones de datos (DSS, SDSS, PanSTARRS, Skymapper, Gaia, Simbad, NED, VizieR, ...)
2. Cualquier sistema de coordenadas (FK4, FK5, ICRS, GAL, SGAL, ECL)
3. Compatible con la mayoría de formatos astronómicos (imágenes: HiPS, FITS, PDS, mapa HEALPix, JPEG, PNG; cubos (HiPS, FITS)
4. Uso de tablas: HiPS, FITS, VOTable, S-extractor, IPAC TBL, ASCII
5. Potentes cajas de herramientas (imágenes: mapa de color, contornos, recorte, recodificación, composición de color, cálculo de píxeles, remuestreo, calibración astrométrica, mosaico, mediciones fotométricas, etc.)

4. Redes neuronales y Deep Learning

4.1. Introducción

Hemos visto como en astronomía se están recopilando cada vez mayor cantidad de datos y la gestion y clasificacion de los mismos esta suponiendo todo un desafió. Este problema esta obligando a los astrónomos a incorporar conocimientos de diferentes ramas del saber.

Ya que la mayoría de los datos obtenidos en astronomía se corresponden a imágenes un enfoque con tecnicas de aprendizaje profundo con redes neuronales convolucionales para la clasificacion e identificación de imagenes podría ser muy beneficioso.

Las Redes Neuronales Artificiales son modelos matemáticos inspirados en el comportamiento biológico de las neuronas y en la estructura del cerebro, y son utilizados para resolver una amplia variedad de problemas. Debido a su flexibidad, una misma red neuronal es capaz de realizar diversas tareas.Al igual que sucede en la estructura de un sistema neuronal biológico, los elementos esenciales de proceso de un sistema neuronal artificial son las neuronas.

Una neurona artificial es un dispositivo simple de cálculo que genera una única respuesta o salida a partir de un conjunto de datos de entrada.Las neuronas se agrupan dentro de la red formando niveles o capas. Dependiendo de su situación dentro de la red, se distinguen tres tipos de capas:

- La capa de entrada, que recibe directamente la información procedente del exterior,incorporándola a la red.
- Las capas ocultas, internas a la red y encargadas del procesamiento de los datos de entrada.
- La capa de salida, que transfiere información de la red hacia el exterior.

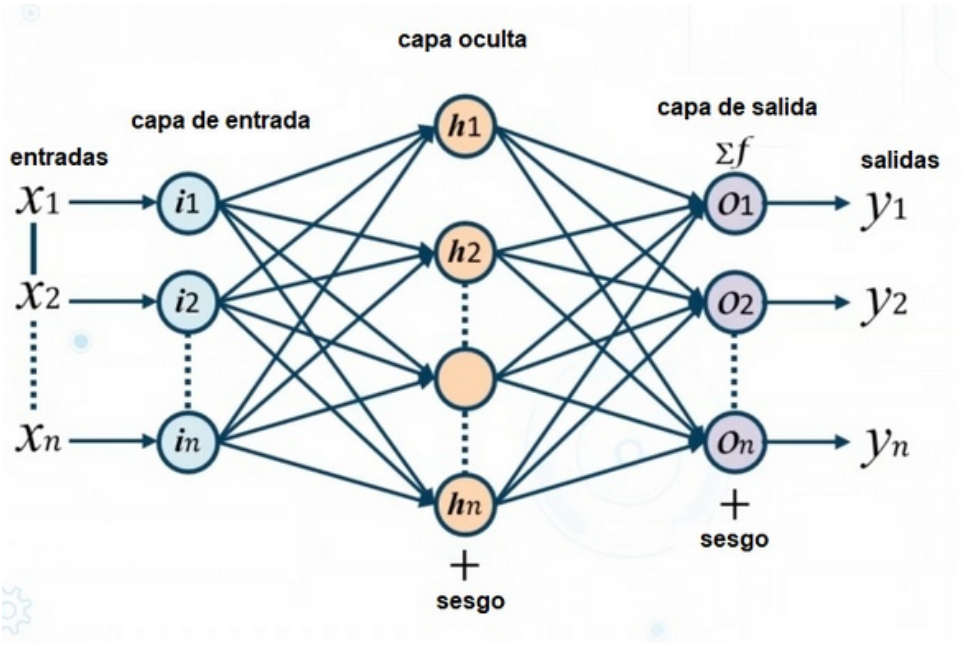


Figura 4.1.: Estructura básica de una red neuronal

Fuente: https://analisisyprogramacionoop.blogspot.com/2020/05/introduccion-redes-neuronales.html

En las redes neuronales existen diferentes topologías elegibles en función de las necesidades del diseñador. Por lo general las operaciones que se llevan a cabo durante el entrenamiento son una mezcla de cálculos lineales y no lineales.

El resultado calculado en cada neurona se lleva a cabo mediante una operación lineal por cada una de las neuronas de la capa anterior que contribuyen a esta y una posterior operación, por lo general no-lineal, aplicada al resultado llamada función de activación.

Se pueden entender las operaciones realizadas en cada una de las capas ocultas de la siguiente manera:

$$h_n = F(w^n * X + b^n) \tag{4.1}$$

Donde w^n refiere a la matriz de pesos,b^n al vector de sesgo y X a la entrada de la neurona en particular que puede ser la entrada original o las salidas de otras neuronas de capas anteriores.

4.2. Funciones de activacion

Las funciones de activación son elementos, por lo general no-lineales, que se operan al final de cada proceso en cada neurona, para definir de manera adecuada la salida de la misma. Existen

una gran variedad y se eligen en función de las necesidades del diseñador.Algunas de las funciones mas utilizadas son:

- **Sigmoide:** función delimitante no-lineal, devuelve un valor entre 0 y 1 y se utiliza principalmente en problemas de clasificación binarios.

$$G(x_i) = \frac{1}{1 + \exp(-x_i)} \tag{4.2}$$

- **Softmax:** función no-lineal similar en funcionamiento a la sigmoide, se utiliza principalmente en problemas de clasificación de mas de una clase ya que devuelve la probabilidad de pertenencia a cada clase.

$$G(x_i) = \frac{exp(x_i)}{\sum_i exp(x_i)} \tag{4.3}$$

- **ReLU (Rectifier Liner Unit):** función delimitante no-lineal, consiste en escoger el máximo valor por muestra de entre el vector entrante y 0. Típicamente se utiliza en problemas con topologías profundas ya que evita que los pesos de una capa se queden estancados durante el entrenamiento.

$$G(x_i) = \text{máx}(0, x_i) \tag{4.4}$$

4.3. Función de coste

La función de coste es la última operación que se lleva a cabo en la red neuronal antes de realizar el backpropagation y pasar a una nueva iteración.

Esta función representa la diferencia que existe entre el resultado que la red consigue, la salida, y el que se querría que fuera, el objetivo.

Esta función define la manera en la que la red neuronal va a aprender y por ello es de vital importancia elegirla adecuadamente

Algunas de las funciones mas utilizadas son:

- **MMSE(Minimum Mean Square Error):** función muy usada en problemas de regresión, dado que lo que busca es que la función de salida de la red sea lo más parecida posible al objetivo planteado.

$$C = \frac{1}{2N} \sum_{n=1}^{N} (X(n) - Y(n))^2 \tag{4.5}$$

  donde N es el número de dimensiones del vector, Y representa al vector de salida y X denota al vector objetivo.

– **Cross-entropía:** funcion usada principalmente en problemas de clasificación.Devuelve la probabilidad de pertenencia a cada una de las clases.(1 y 0 si son solo 2 clases)

$$C = \sum_{k=1}^{k} (X_k * log(y_k) + (1 - X_k) * log(1 - Y_k) \tag{4.6}$$

donde k es el numero de clases, X el vector objetivo e Y la salida de la red

4.4. Backpropagation

La retropropagacion es un método de cálculo del gradiente utilizado para entrenar redes neuronales. Una vez se ha realizado una iteracion, se han obtenido las salidas de la red neuronal y se calcula una señal de error para cada una de ellas mediante la función de coste se comienza la retropropagacion del error.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original.

Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Con esta información las neuronas ajustan sus pesos para minimizar este error.

Gracias a este proceso a medida que se entrena la red sucesivamente en diferentes iteraciones, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada.

4.5. Redes neuronales convolucionales

Las CNNs son muy similares a las redes neuronales clásicas. Están formadas por neuronas que tienen pesos y sesgos que pueden ajustarse. La diferencia entre las CNNs y las redes neuronales clásicas es que las primeras suponen explícitamente que la entrada se trata de un conjunto de datos invariante en cuanto a que no se pueden intercambiar las columnas y/o filas entre sí,como es el caso de las imágenes o los espectogramas, que al hacer dichos tipos de intercambios dejarían de ser datos útiles.

Esto permite a las CNNs codificar ciertas propiedades en la arquitectura y hace que el entrenamiento sea muchísimo más eficiente y el número de parámetros a ajustar por la red se reduzca considerablemente.

Si quisiéramos tratar una imagen en una red neuronal clásica con un tamaño de imagen de ejemplo de 200x200 se tendría un total de 40.000 píxeles por cada imagen y cada pixel tendría que estar conectado a cada neurona de la primera capa. Si disponemos una red con por ejemplo 1000 neuronas en la primera capa, se tendrían que ajustar 40.000.000 de pesos.

Sin embargo, las CNNs trabajan de forma distinta. Existen capas convolucionales, que consisten en uno o varios filtros que se van deslizando sobre la imagen, realizando operaciones de convolución. Cada uno de estos filtros están compuestos de neuronas. Si por ejemplo, en una primera capa oculta se tiene una capa convolucional de 7x7x128, esto significa que tiene 128 filtros de 7x7 neuronas, que se van a ir deslizando por la imagen realizando operaciones de convolución. En ese ejemplo, la red tendría que aprender 7x7x128 = 6.272 pesos para la primera capa. Como se ve, la diferencia con respecto al ejemplo de la red neuronal clásica en cuanto al número de pesos a aprender es muy grande.

4.5.1. Diferentes tipos de Capas en las CNNs

En la arquitectura de una CNNs hay diferentes tipos de capas que se utilizan.

Capa de convolucion:

Estas capas son las encargadas de hacer la operación más característica de las CNNs, las convoluciones. Dada la importancia de esta operación a continuación se va a realizar una breve explicación del fundamento de esta operación.

La operación de convolución consiste en el sumatorio de las multiplicaciones elemento a elemento entre dos matrices del mismo tamaño. Esta capa recibe como entrada una matriz y se realizan convoluciones sobre ella mediante un filtro que se irá deslizando por dicha matriz de entrada.

Una vez se le han hecho las convoluciones, se obtiene una nueva matriz, que será la salida de esta capa. Esta matriz de salida, luego se introduce a una capa de regulación y a la salida de esa capa de regulación se le suele llamar mapa de características. Debido a se le han aplicado filtros mediante convoluciones a la matriz de entrada, la matriz de salida contiene características de dicha entrada.

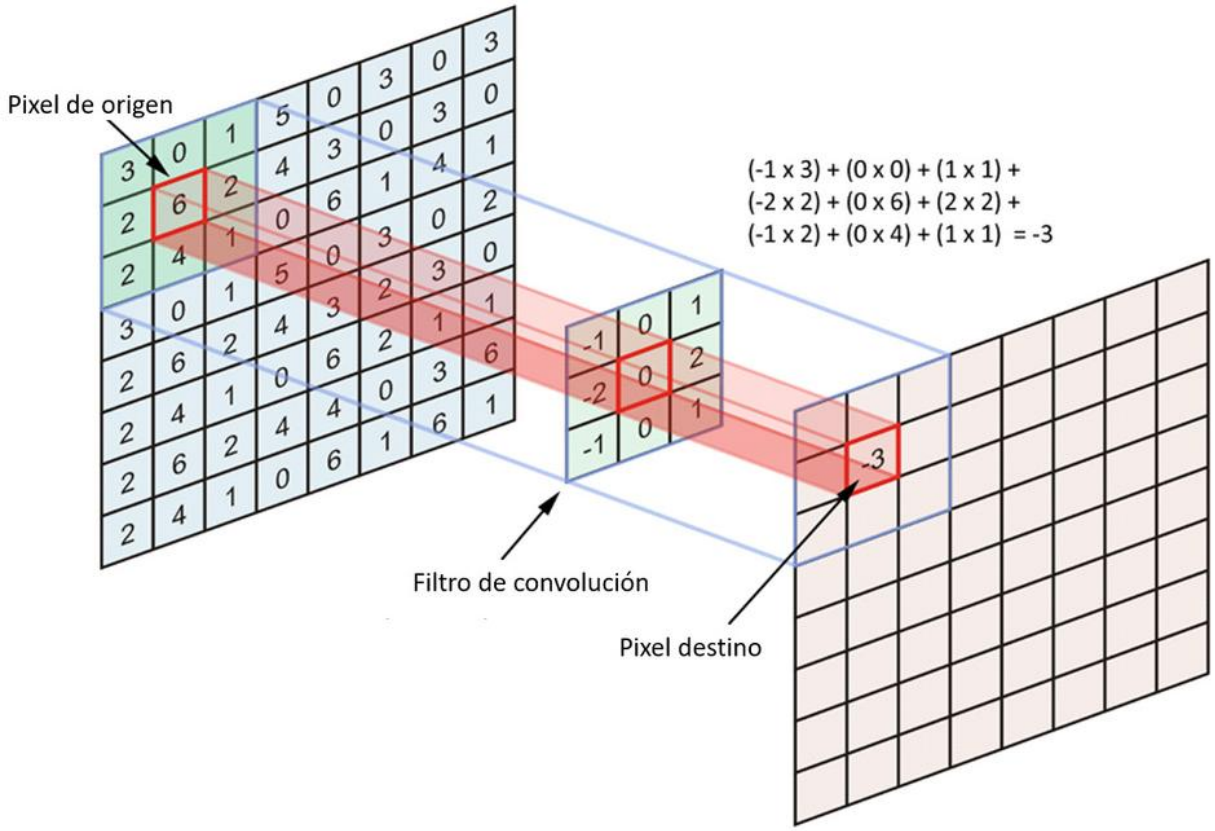


Figura 4.2.: Operación de convolución

Fuente :https://www.researchgate.net/figure/Matrix-convolution-filter-algorithm-as-a-graphic-diagram-12_fig18_304526867

Las CNNs suelen tener varias capas convolucionales seguidas . La primera de ellas tiene como entrada, en caso de que estemos ante un problema de visión artificial, la imagen a tratar. Tras la primera capa convolucional, se obtendrá un mapa de características de bajo nivel.Ese mapa de características irá pasando por varias capas convolucionales más, de forma que cada vez se va a ir obteniendo mapas de características con características de más alto nivel.

Capa de activación:

El propósito de esta capa es introducir la no linealidad a un sistema que básicamente ha estado computando las operaciones lineales durante las capas convolucionales (sólo se realizan sumas y multiplicaciones en dichas capas). En el pasado, se utilizaron funciones no lineales como la tangente hiperbólica o la sigmoide, pero los investigadores descubrieron que las capas Relu funcionan mucho mejor porque la red es capaz de entrenar un poco más rápido, sin haber una diferencia significativa en la precisión.

La función de activación Relu, también ayuda a aliviar el problema de los gradientes de fuga,

por el cual las capas inferiores de la red se forman muy lentamente porque el gradiente disminuye exponencialmente a través de las capas.

La función de activación más utilizada en la actualidad es la función ReLu aun que tambien se utilizan funciones como como la tangente hiperbólica o la sigmoide.

$$ReLu(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{4.7}$$

La función ReLu funciona pasándole como entrada una matriz, a la cual esta función cambia los números negativos que tenga por ceros y devuelve dicho resultado como salida . Esta capa aumenta las propiedades no lineales del modelo sin afectar a los campos receptivos de la capa convolucional. Generalmente se utiliza tras una capa convolucional.

Capa de agrupación(pooling):

Las capas de agrupación, también conocidas como capas de pooling,se suelen utilizar tras una capa de activación, pero también es muy común no aplicar esta capa tras haber utilizado varias veces seguidas capas de activación.

En esta categoría, al igual que en la capa de activación, también hay varias opciones de funcionamiento, siendo el Max Pooling la más popular y utilizada, la cual consiste en la aplicación de una máscara de máximos con un stride igual a la longitud de la máscara. También existen otros métodos de agrupación, como Average Pooling o Global pooling.

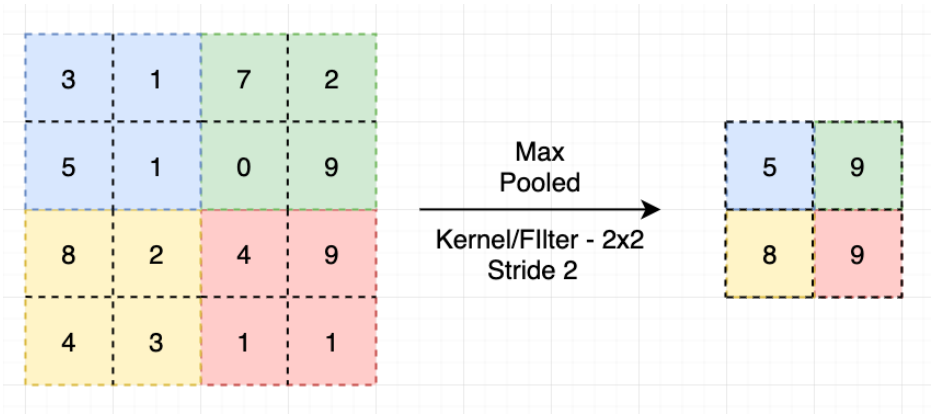


Figura 4.3.: Operación de max pooling

Fuente:https://ai.plainenglish.io/pooling-layer-beginner-to-intermediate-fa0dbdce80eb

Esta capa tiene la función de reducir el tamaño del volumen de la matriz de entrada, tratando de mantener las características más importantes. Esto es importante y necesario para que la duración del entrenamiento se reduzca y se realice en un tiempo razonable

Capa FC:

Esta capa, normalmente, se utiliza al final de la red, tras haber realizado las convoluciones que el programador ha estimado oportunas. Se trata de un perceptrón multicapa. Sus siglas FC provienen de la palabra Fully-Connected, que hace referencia a que cada neurona de una capa tiene una conexión a cada una de las neuronas de la siguiente capa.

Capa softmax:

La capa softmax se suele utilizar en la clasificación de objetos, al final de la CNN, para determinar la probabilidad de que exista un determinado objeto en la imagen. Asigna probabilidades decimales a cada clase en un caso de clases múltiples.Esas probabilidades decimales deben sumar en total 1.

Esta capa tendrá un nodo por cada clase que se haya determinado para el problema a resolver. En resumen, la salida de la función softmax es equivalente a una distribución de probabilidad categórica, que te indica la probabilidad de que alguna de las clases esté presente en la imagen.

4.5.2. Redes ResNet

Aunque las redes convolucionales profundas pueden tener un mejor rendimiento en la clasificación la mayoría de las veces, son más difíciles de entrenar principalmente por dos razones:

- **Problema del gradiente de fuga**: A veces una neurona muere durante el proceso de entrenamiento y dependiendo de su función de activación, es posible que nunca vuelva.
- **Optimización más costosa**: Cuando el modelo introduce más parámetros, se vuelve más difícil entrenar la red. Esto no es simplemente un problema de sobre-ajuste, ya que a veces, agregar más capas conduce a aún más errores de entrenamiento

Cuando apilamos múltiples capas en una red neuronal de convolución, en teoría, el error de entrenamiento debería disminuir, pero en la práctica o en la realidad, agregar más capas en la CNN a partir de cierto punto (haciendo que la CNN sea más profunda) hace que el error de entrenamiento aumente en lugar de disminuir.

Esto no sucede debido al sobre-ajuste . Porque el sobre-ajuste significa buena precisión en el entrenamiento pero mala precisión en las pruebas. El problema aquí es un problema de optimización o degradación . Los modelos más profundos son más difíciles de optimizar. Con el aumento de la profundidad de la red, la precisión se satura y luego se degrada rápidamente (aumenta el error de entrenamiento).

Por lo tanto, las CNN profundas, a pesar de tener una mejor clasificación y rendimiento, son más difíciles de entrenar. Una forma eficaz de resolver estos problemas es utilizando Redes residuales (ResNets)

Hay una simple diferencia entre ResNets y redes convolucionales comunes. El objetivo es proporcionar un camino claro a los gradientes para propagar hacia atrás a las primeras capas de la red. Esto acelera el proceso de aprendizaje al evitar la desaparición del problema del gradiente o las neuronas muertas.Esto se hace mediante los llamados bloques residuales.

Los bloques residuales se consideran el componente básico de ResNet. En el bloque residual, la entrada x se agrega directamente a la salida de la red, es decir, F (x) + x y esta ruta se conoce como conexión de salto o acceso directo .

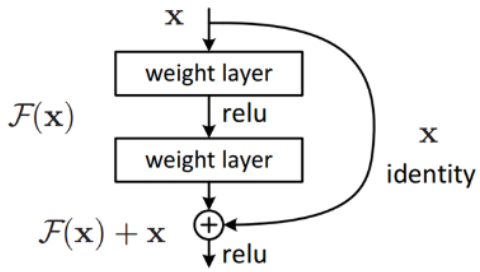


Figura 4.4.: Diseño bloque residual

https://www.jeremyjordan.me/content/images/2018/04/Screen-Shot-2018-04-16-at-6.29.19-PM.png

En los principales artículos de ResNet, los autores han sugerido diferentes configuraciones de ResNets con 18, 34, 50, 101 y 152 capas. Uno podría describir ResNets como múltiples bloques básicos que son conectados en serie entre sí y también hay conexiones de acceso directo paralelas a cada bloque básico que se agregan a sus salidas. Si el tamaño de entrada y salida para un bloque básico es igual, el acceso directo es simplemente una matriz de identidad. De lo contrario, uno puede usar la agrupación promedio (para la reducción) y el relleno cero (para ampliación) para ajustar el tamaño.

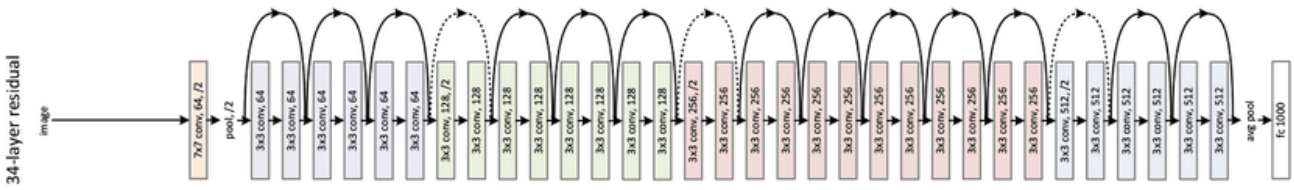


Figura 4.5.: Arquitectura ResNet34

https://www.researchgate.net/figure/ResNet-neural-network-architecture-ResNet-34-pictured-image-from-11_fig6_330400293

4.6. Herramientas utilizadas

4.6.1. Colab

Google Colaboratory (también conocido como Colab) es un servicio en la nube basado en Jupyter Notebooks para difundir la educación y la investigación sobre el aprendizaje automático. Proporciona un tiempo de ejecución completamente configurado para aprendizaje y acceso gratuito a una robusta GPU.

Figura 4.6.: Logotipo de Google Colab

https://www.marketing-branding.com/google-colaboratory-colab-guia-completa-espanol/

Características:

- Consta de servicio cloud, es decir, se ejecuta en la nube.
- Basado en Jupyter Notebook, proporciona la escritura y ejecución de código sin necesidad de realizar ninguna instalación previa.
- Permite el uso de GPUs y TPUs.
- Concede acceso a librerías como PyTorch, Keras, Fastai o TensorFlow, entre otras.
- Se enlaza con una cuenta de Google Drive, lo que posibilita interactuar con archivos que se almacenen allí.
- Los notebooks, o cuadernos, que se generan pueden compartirse con múltiples usuarios al encontrarse en Google Drive.

– Como los notebooks tienen extensión '.ipynb', pueden ejecutarse en otros entor- nos como JupyterLab, Jupyter Notebook, u otros sistemas compatibles.

Hoy en día, las soluciones en la nube son atractivas porque proporcionan hardware sobre la marcha y eliminan la necesidad de mantenimiento y configuración de recursos de hardware. Se considera que la computación paralela ejecuta el proceso de entrenamiento de redes en un tiempo factible y dado que las GPU son dispositivos masivamente paralelos, Colab parece un entorno que se ajusta a nuestras necesidades.

4.6.2. Python

Lenguaje de programación de alto nivel, interpretado y orientado a objetos. Una de sus principales virtudes es que favorece la extracción y procesamiento de datos cuan- do éstos tienen un volumen muy elevado.

Figura 4.7.: Logotipo de Python

https://www.icog.es/cursos/index.php/introduccion-a-la-programacion-en-python-para-geologos/

Características:

– Sencillo y fácil de usar.

– Multiplataforma, permitiendo que se ejecute en diferentes sistemas operativos.

– Consta de licencia de código abierto.

– Sigue un formato de código estructural.

– Incluye la biblioteca estándar de Python, ofreciendo soporte integrado para modificar ficheros, entre otros usos.

– Pueden añadirse módulos, ampliando así el lenguaje, y utilizarlos como accesos directos una vez han sido creados.

- Multiparadigma, es decir, soporta más de un modelo de desarrollo; específica- mente el imperativo, funcional y orientado a objetos.
- Pueden añadirse módulos, ampliando así el lenguaje, y utilizarlos como accesos directos una vez han sido creados.

En los últimos años se ha asociado a Data Science, ciencia centrada en el estudio de los datos, debido a su facilidad para recolectar y clasificar información; lo que junto con algunos de sus frameworks y librerías hace que sea un buen motivo para seleccionar este lenguaje como herramienta de trabajo.

4.6.3. Fastai

Fastai es una biblioteca de aprendizaje profundo que proporciona componentes de alto nivel que pueden proporcionar rápida y fácilmente resultados de vanguardia en dominios de aprendizaje profundo estándar.Cuenta con una arquitectura cuidadosamente estratificada, que expresa patrones subyacentes comunes de muchas técnicas de procesamiento de datos y aprendizaje profundo en términos de abstracciones desacopladas. Estas abstracciones se pueden expresar de forma concisa y clara aprovechando el dinamismo del lenguaje Python subyacente y la flexibilidad de la biblioteca PyTorch.

Figura 4.8.: Logotipo de fastai

https://pdito.github.io/blog/

Fastai incluye:

- Un nuevo sistema de envío de tipos para Python junto con una jerarquía de tipos semánticos para tensores.
- Una biblioteca de visión por computadora optimizada para GPU que se puede ampliar en Python puro
- Un optimizador que refactoriza la funcionalidad común de los optimizadores modernos en dos piezas básicas, lo que permite implementar algoritmos de optimización en 4-5 líneas de

código.

- Un novedoso sistema de devolución de llamada bidireccional que puede acceder a cualquier parte de los datos, el modelo o el optimizador y cambiarlo en cualquier momento durante el entrenamiento.
- Una nueva API de bloque de datos

Está construida sobre PyTorch (biblioteca de aprendizaje automático de código abierto) y contiene varios modelos pre entrenados de torchvision, un paquete que incluye datos populares, arquitecturas de modelos y transformaciones comunes de imágenes. Entre estos modelos encontramos las siguientes arquitecturas neuronales: ResNet, SqueezeNet, DenseNet, VGGNet y AlexNet.

Éstas han sido previamente entrenadas con un conjunto de datos y contienen los pesos y sesgos que representan las características de dichos datos. La ventaja de utilizar este tipo de modelos es que las características son comúnmente transferibles a datos distintos, ahorrándonos tanto tiempo como recursos computacionales.

5. Clasificacion morfológica de galaxias en el espacio profundo

5.1. Descripcion del problema

Desde la primera mitad del siglo XX las galaxias han sido clasificadas según su aspecto visual. El estudio de la formación y la evolución de las galaxias requiere de la medición de sus parámetros morfológicos. Tradicionalmente, el análisis morfológico se ha llevado a cabo principalmente a través de la extracción y selección de características, o mediante la inspección visual de expertos en un proceso que consume muchos recursos y que resulta prácticamente imposible de realizar en colecciones masivas de imágenes.

Para intentar resolver este problema pueden emplearse redes neuronales convolucionales. En el siguiente apartado se realizara una serie de experimentos sobre un conjunto de imágenes de galaxias previamente clasificadas para observar el desempeño de este tipo de redes e intentar lograr el mejor resultado posible.

5.2. Preparación dataset CANDELS

El conjunto de datos cuenta con 3412 imágenes de galaxias obtenidas de los datos del repositorio Mikluski para telescopios espaciales proporcionados por CANDELS Multi-Cycle Treasury Program junto con NASA/ESA HST.

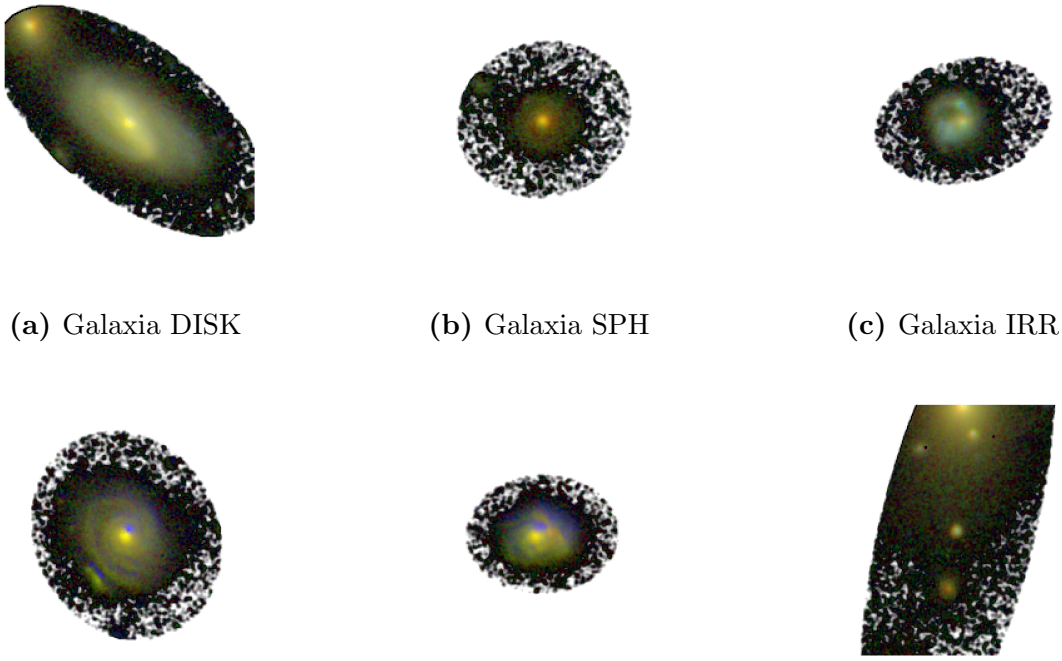
CANDELS es un estudio de imágenes del Universo profundo que se llevó a cabo con una cámara WFC3 de infrarrojo cercano y una cámara óptica ACS a bordo del Telescopio Espacial Hubble. La encuesta fue diseñada para enfocarse en dos épocas críticas en la evolución cósmica: En el .ªmanecer Cósmico", menos de mil millones de años después del Big Bang y en el "Mediodía Cósmico", 2-4 mil millones de años después del Big Bang. El equipo de CANDELS incluyó a científicos de IPAC y el procesamiento y análisis de datos del componente UV del estudio (en GOODS North) estuvo a cargo de IPAC.

Para generar el conjunto de datos se han obtenido primero los ficheros FITS para cada banda lumínica correspondientes a cada galaxia con el fin de utilizarlos para generar una imagen RGB.

A esta imagen se le ha aplicado una mascara para eliminar pixeles de la imagen que no nos proporcionan ninguna información útil a la hora de la clasificación.

Las galaxias del conjunto están clasificadas en función de 5 clases.

- DISK: la galaxia tiene forma de disco
- SPH : la galaxia tiene forma esferoidea
- IRR : la galaxia tiene forma irregular
- DISKSPH : mezcla entre disco y esfera
- DISKIRR : mezcla entre disco e irregular
- NONE : no-clasificable



(a) Galaxia DISK **(b)** Galaxia SPH **(c)** Galaxia IRR

(d) Galaxia DISKSPH **(e)** Galaxia DISKIRR **(f)** Galaxia NONE

Figura 5.1.: Ejemplos de imagenes de cada clase

5.3. Data Augmentation y Label Smothing

Ya que nuestro conjunto de datos solo cuenta con 3460 imágenes vamos a realizar lo que se conoce como **data augmentation**. Data augmentation es la generación artificial de datos por medio de perturbaciones en los datos originales. Esto nos permite aumentar tanto en tamaño

como en diversidad nuestro conjunto de datos de entrenamiento. En el computer vision, esta técnica se convirtió en un estándar de regularización, y también para mejorar el rendimiento y combatir el overfitting en CNNs.

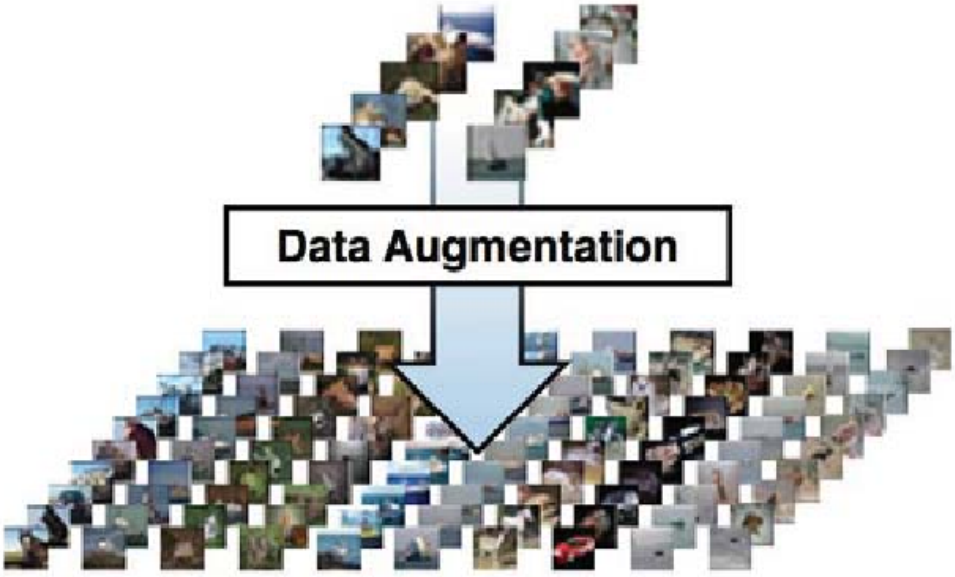


Figura 5.2.: Ilustración Data Augmentation

https://www.semanticscholar.org/paper/Improving-Deep-Learning-with-Generic-Data-Taylor-Nitschke/27c495019bae2d7ab5b607e11a47a39cd9f1c519

El **label smoothing** es una técnica sencilla que trata de paliar el numero de etiquetas de los datos de entrenamiento mal asignadas, así como de disminuir la certeza de las predicciones de la red ayudándonos a evitar el sobre-ajuste. Para ello el label smoothing aplana las etiquetas dadas asignando una distribución uniforme a todas las clases.

Asumiendo que cada imagen x se le ha asignado una distribución con formato de etiquetas tal como $y = [y_1, ..., y_i, ..., y_k]^T$, siendo K el numero de clases. Si x pertenece a la clase i entonces $y_i = 1$ y los demás serán $y_j = 0$. El label smoothing estándar se puede explicar a partir de dos pasos, primero modificamos las etiquetas usando la función

$$y_k^{LS} = y_k(1 - \alpha) + \alpha/K\alpha \in (1, 0) \tag{5.1}$$

donde $\alpha \in (1, 0).y_k = 1$ tiene valor 1 si es la clase correcta, en caso contrario tendrá valor 0. De esta forma pasamos de tener una red con la certeza al 100 % de que la imagen pertenece a una clase a que exista cierta posibilidad de que este equivocada la predicción, α indica como varia esa posibilidad.

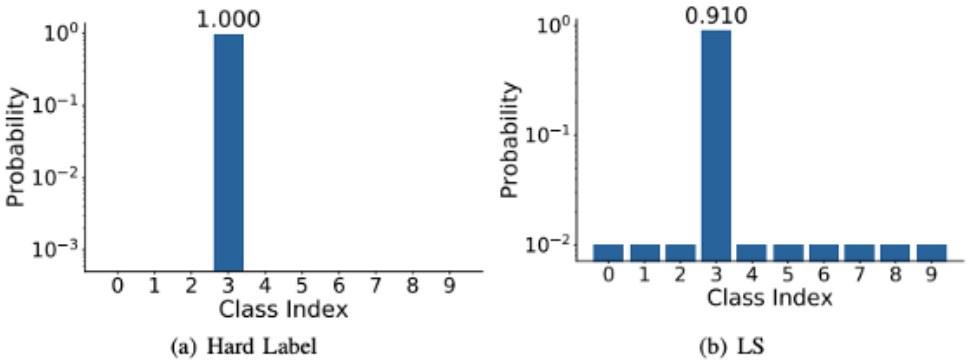


Figura 5.3.: Ejemplo label smoothing para 9 clases

https://towardsdatascience.com/label-smoothing-make-your-model-less-over-confident-b12ea6f81a9a

5.4. Experimentación

5.4.1. Diseño

El diseño de la experimentación constara de las siguientes fases: Primero se realizara un experimento base que servirá como punto de partida para realizar diferentes modificaciones.

A continuación se realizaran una serie de experimentos donde se aplicaran diferentes técnicas de data augmentation con diferente parametría y se compararan resultados con el experimento base.

A igual manera que con las técnicas de data augmentation se realizara un experimento aplicando label smoothing y se compara con el experimento base.

Finalmente se realizara un experimento final donde se tendrán en cuenta los resultados de los experimentos anteriores.En este experimento se aplicaran todas las técnicas utilizadas previamente con el fin de intentar obtener el mejor resultado posible.

Para cada una de estas etapas descritas previamente se utilizaran dos arquitecturas diferentes en cada uno de los experimentos, una ResNet18 y una ResNet34 y se compararan también resultados entre las diferentes arquitecturas.

5.4.2. Experimento Base

En fastai contamos con modelos ResNet pre-entrenados utilizando conjuntos de imágenes obtenidos en la base de datos ImageNet. Podemos ajustar estos modelos pre entrenados a nuestras necesidades utilizando la función $fine\_tune()$.

```
@delegates(Learner.fit_one_cycle)
def fine_tune(self:Learner, epochs, base_lr=2e-3, freeze_epochs=1, lr_mult=100,
              pct_start=0.3, div=5.0, **kwargs):
    "Fine tune with `Learner.freeze` for `freeze_epochs`, then with `Learner.unfreeze` for `epochs`, using discriminative LR."
    self.freeze()
    self.fit_one_cycle(freeze_epochs, slice(base_lr), pct_start=0.99, **kwargs)
    base_lr /= 2
    self.unfreeze()
    self.fit_one_cycle(epochs, slice(base_lr/lr_mult, base_lr), pct_start=pct_start, div=div, **kwargs)
```

Figura 5.4.: Codigo función Fine_tune()

fine_tune() consta de dos etapas:

En la primera etapa se entrena la ultima capa de la red. Se comienza con los pesos de el modelo previamente entrenado,se elimina la última capa y se agrega una nueva capa para nuestras clases con pesos aleatorios.Se congelan los pesos pre-entrenados ya que son pesos iniciales buenos (o al menos mejores que unos completamente aleatorios), y se entrena la última capa.

Cuando la última capa está funcionando bien, es decir, sus pesos también son "buenos"pesos iniciales, pasamos a la segunda etapa en la que descongelamos todos los pesos e intentamos entrenar todo el modelo.

Este proceso por el cual se reajustan los pesos de un modelo ya pre-entrenado se conoce como Transfer-learning.

Dividiremos los datos en dos conjuntos aleatorios para entrenamiento y validación utilizando un 80 % para entrenar y un 20 % para validar, utilizaremos el tamaño de batch por defecto que proporciona fastai que es de 64, el learning rate base por defecto es 0.002 con un multiplicador de 100.

Una vez entrenados los experimentos básicos para cada modelo por 50 épocas obtenemos los siguientes resultados:

Figura 5.5.: Entrenamiento Experimento Base

En la figura 5.5 podemos visualizar la evolución de la función de perdida tanto en entrenamiento como en validación para ambos experimentos base a lo largo de las diferente épocas. La primera gráfica corresponde a los resultados de ResNet34 y la segunda a ResNet18.

Se puede observar un claro overfiting en ambos experimentos con trayectorias muy similares entre ellos, la función de perdida en la validación aumenta ligeramente durante todas las épocas mientras que la función de perdida del conjunto de entrenamiento disminuye rápidamente. ninguno de los experimentos esta recogiendo adecuadamente las características generales de cada categoría de imágenes.

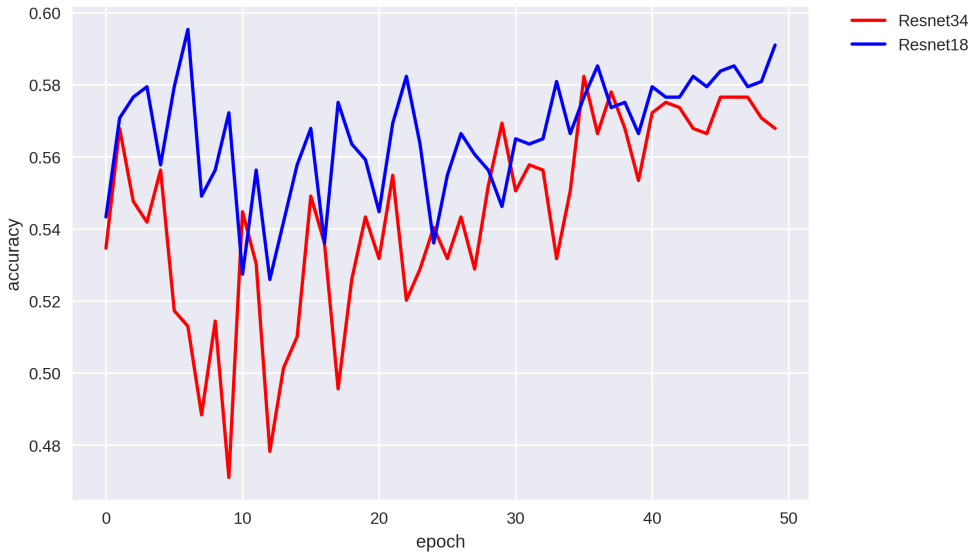


Figura 5.6.: Precisión Experimentos Base

En la figura 5.6 observamos la evolución de la precisión con la que ambos experimentos clasifican las diferentes imágenes del conjunto de validación. La precisión evoluciona de manera similar en ambos experimentos siendo un poco superior en casi todas las épocas para la ResNet18. El experimento con ResNet18 termina con una precisión aproximada de 59 % respecto a ResNet34 con 57 %

Esta precisión hace referencia al numero total de imágenes correctamente clasificadas sin hacer incapie en las diferentes categorías a las que pertenecen dichas imágenes. Para observar mas detalladamente la precisión sobre cada categoría generaremos una matriz de confusión.

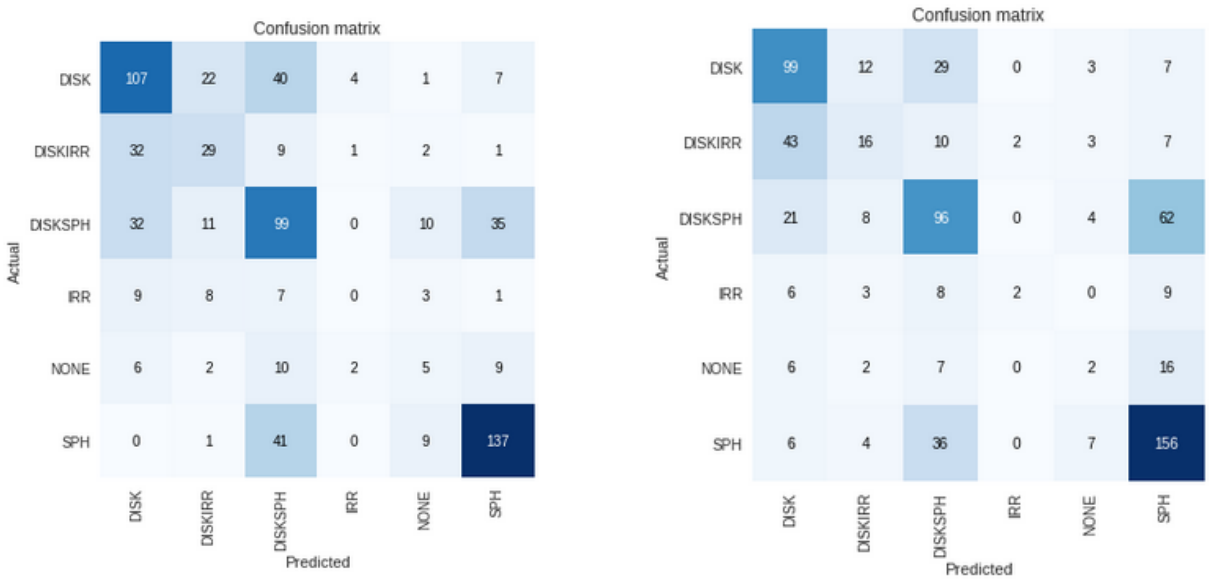


Figura 5.7.: Clasificaciones Experimento Base

En la matriz de confusión tanto para el modelo con ResNet34(izquierda) como ResNet18(derecha) vemos que las categorías mejor clasificadas son DISK ,DISKSPH y SPH, ambos modelos tienen dificultades para diferenciar entre los diferentes tipos de galaxias DISK y entre DISKSPH y SPH, las categorías NONE e IRR son las peor clasificadas.

5.4.3. Data Augmentation

Debido a las características de las imágenes de galaxias, transformaciones que modifiquen el brillo y o la forma de la imagen de forma significativa no parecen ser las mas adecuadas ya que es lo que utilizamos para diferenciar que pixeles pertenecen o no a la galaxia.

A continuación pasaremos a modificar el experimento base con 2 tipos de transformaciones, giros diédricos y la adición de ruido gaussiano, realizando un análisis de sensibilidad de diferentes parámetros de las transformaciones para descubrir como se comportan estas transformaciones respecto al experimento base.

5.4.3.1. Ruido Gaussiano

El ruido gaussiano es un ruido estadístico cuyos valores vienen determinados por una función de probabilidad normal. Para aplicar este ruido a nuestras imágenes se ha creado una función en Python que añade a cada pixel un valor perteneciente a una distribución normal de media y varianza por determinar.

```
class GaussianNoise(RandTransform):

    def __init__(self, mean=0., std=1., **kwargs):
        self.std = std
        self.mean = mean
        super().__init__(**kwargs)

    def encodes(self, x:TensorImage):
        return x + torch.randn(x.size()).cuda() * self.std + self.mean
```

Figura 5.8.: Función Python para aplicar ruido gaussiano

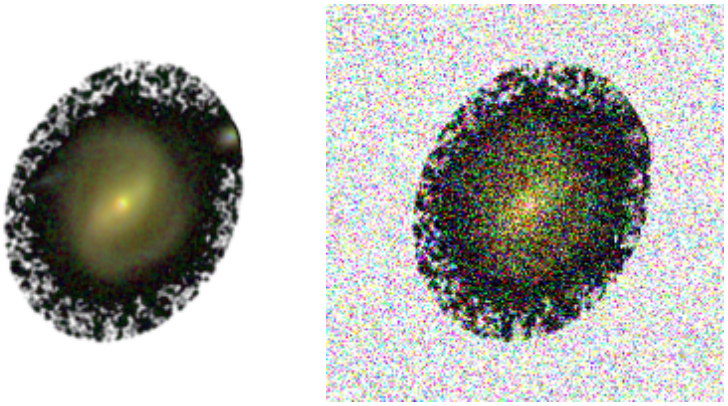


Figura 5.9.: Aplicación ruido gaussiano MEAN = 0 STD = 100

En la figura 5.9 el ruido aplicado tiene una varianza muy elevada para poder visualizar bien la transformación. Durante el entrenamiento los valores que utilizaremos serán mas moderados para no influir demasiado en la morfología de la imagen.

Realizaremos varios experimentos con diferentes valores para la varianza del ruido, compararemos la evolución de las funciones de perdida y de la precisión de dichos experimentos con el experimento base realizado previamente para ambos modelos tanto ResNet34 como ResNet18.

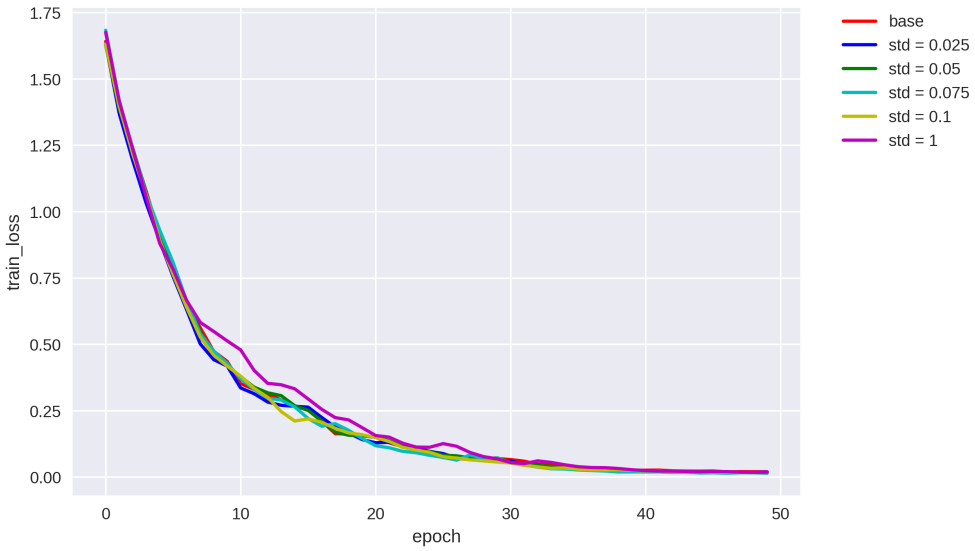


Figura 5.10.: Comparativa perdida en entrenamiento Resnet18

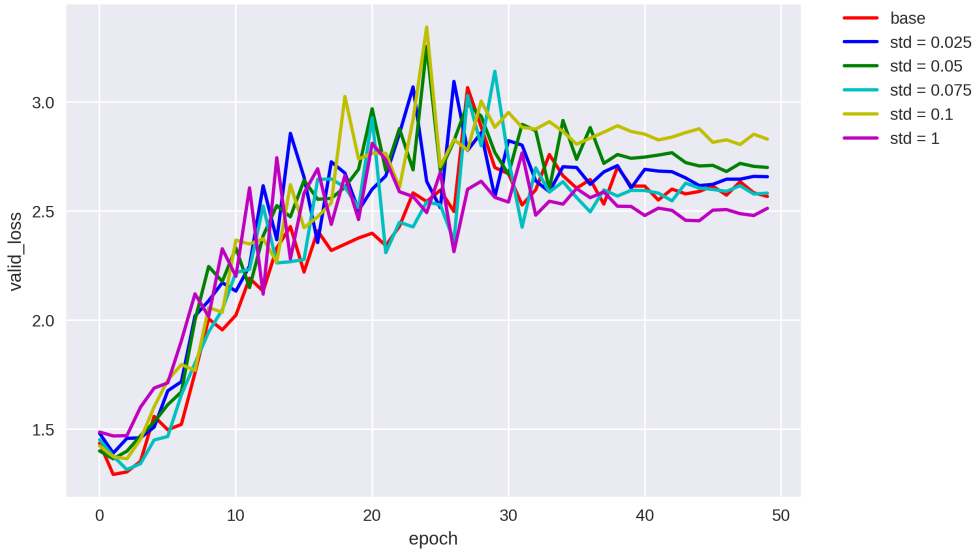


Figura 5.11.: Comparativa perdida en validación Resnet18

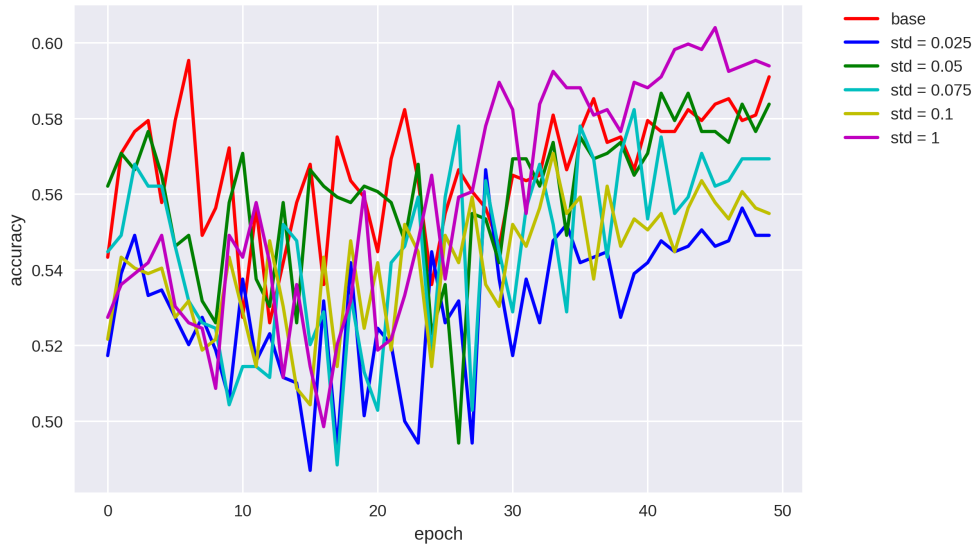


Figura 5.12.: Comparativa precisión Resnet18

El cuanto a la perdida en entrenamiento la evolución es prácticamente idéntica para todos los experimentos, decreciendo a un ritmo ligeramente inferior para el experimento con STD = 1. En cuanto a la perdida en validación sigue incrementándose en las sucesivas épocas para todos los experimentos por lo que seguimos teniendo sobre-ajuste.

En cuanto a la precisión de la clasificacion observamos que solo ha mejorado con respecto al experimento base en el caso std = 1 . El resto de casos si bien van mejorando a medida que se realizan epocas están unos cuantos puntos porcentuales por debajo a excepción de std = 0.05 que se mantiene mas o menos igual.

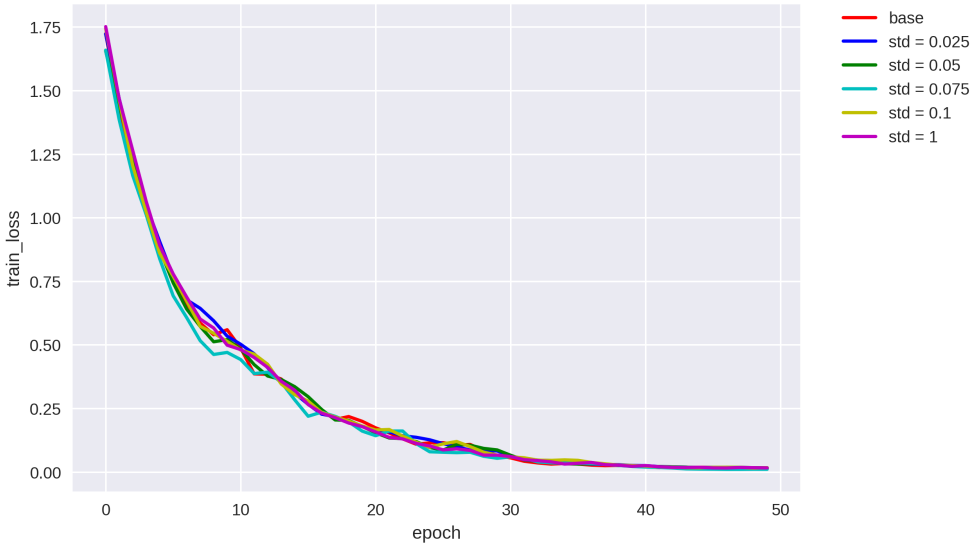


Figura 5.13.: Comparativa perdida en entrenamiento Resnet34

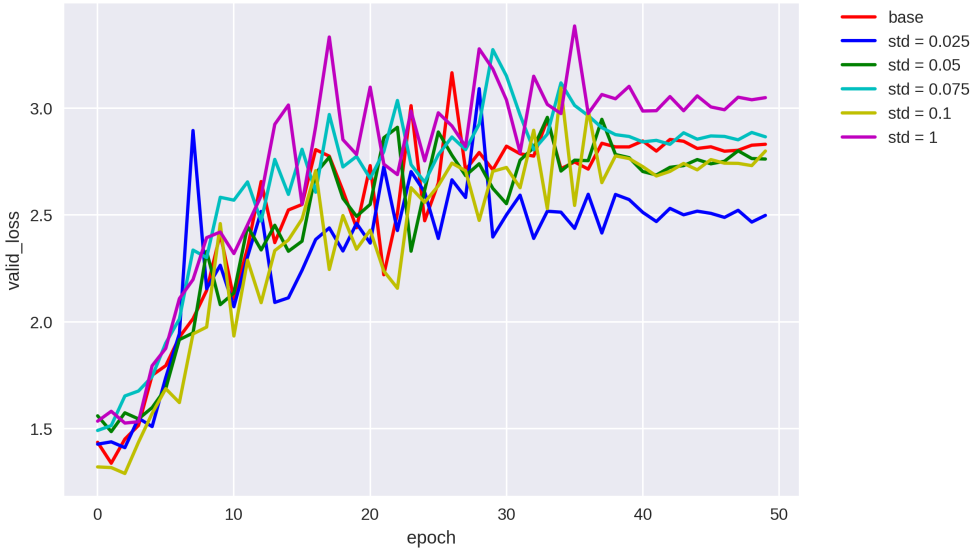


Figura 5.14.: Comparativa perdida en validación Resnet34

Figura 5.15.: Comparativa precisión Resnet34

De forma análoga al experimento realizado con ResNet18 la perdida en entrenamiento es prácticamente idéntica para todos los experimentos.La perdida en validación también continua incrementándose en las sucesivas épocas.

La precisión de la clasificacion también ha mejorado con respecto al experimento base en varios casos. El caso con la mayor mejora esta vez es std = 0.025 con una mejora algo superior a un 4 % respecto al experimento base

De entre los dos modelos elegiremos el caso en el que se haya percibido mayor mejora y los compararemos entre si. Para ResNet18 escogeremos std = 1 y para ResNet34 std = 0.025.

Figura 5.16.: Comparativa perdida std = 1 ResNet18 std = 0.025 Resnet 34

Figura 5.17.: Comparativa precisión std = 1 ResNet18 std = 0.025 Resnet 34

La diferencia entre las perdidas de entrenamiento y validación, así como la precisión son muy similares Tanto para ResNet34 como ResNet18 con sus respectivas transformaciones aunque ResNet34 obtiene una precisión ligeramente mayor superando el 60 %.

Aumentar el conjunto de datos utilizando unicamente transformaciones de ruido gaussiano no parece tener gran impacto en el entrenamiento ni en la clasificacion para ninguno de los dos modelos. Si bien se observa una ligera mejora en la precisión de las clasificaciones se sigue apreciando un claro sobre-ajuste de los datos de entrenamiento, veremos como evoluciona realizando transformaciones de distinto tipo.

5.4.3.2. Giros Diédricos

La siguiente transformación que vamos a aplicar al conjunto de datos de entrenamiento son los giros diédricos, para ello utilizaremos una de las funciones proporcionadas por fastai llamada "*Dihedral*".

```
def _dihedral(x, k:partial(uniform_int,0,7)):
    "Randomly flip `x` image based on `k`."
    flips=[]
    if k&1: flips.append(1)
    if k&2: flips.append(2)
    if flips: x = torch.flip(x,flips)
    if k&4: x = x.transpose(1,2)
    return x.contiguous()
dihedral = TfmPixel(_dihedral)
```

Figura 5.18.

Esta función genera con igual probabilidad 8 tipos de modificaciones sobre la imagen que consisten en rotaciones con ángulos múltiplos de 90 grados y transposiciones.

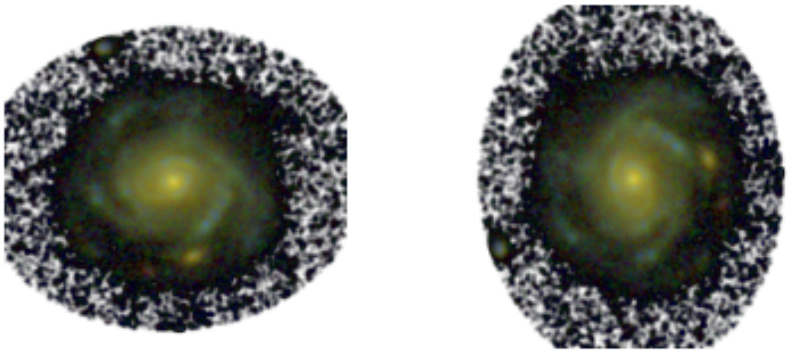


Figura 5.19.: Ejemplo transformación con Giros Diédricos

De la misma forma que se ha realizado con las transformaciones de ruido realizaremos varios experimentos con transformaciones con giros para ambos modelos y compararemos al final entre los 2 mejores experimentos para cada modelo. El parámetro que modificaremos a lo largo de los experimentos sera la probabilidad p de que a una imagen se le aplique la transformación antes de comenzar el entrenamiento.

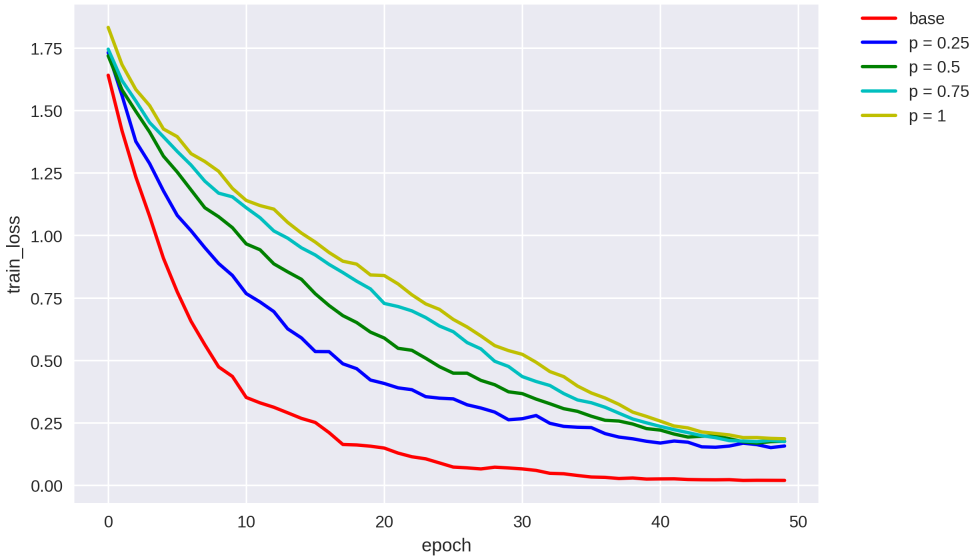


Figura 5.20.: Comparativa perdida en entrenamiento Resnet18

Figura 5.21.: Comparativa perdida en validación Resnet18

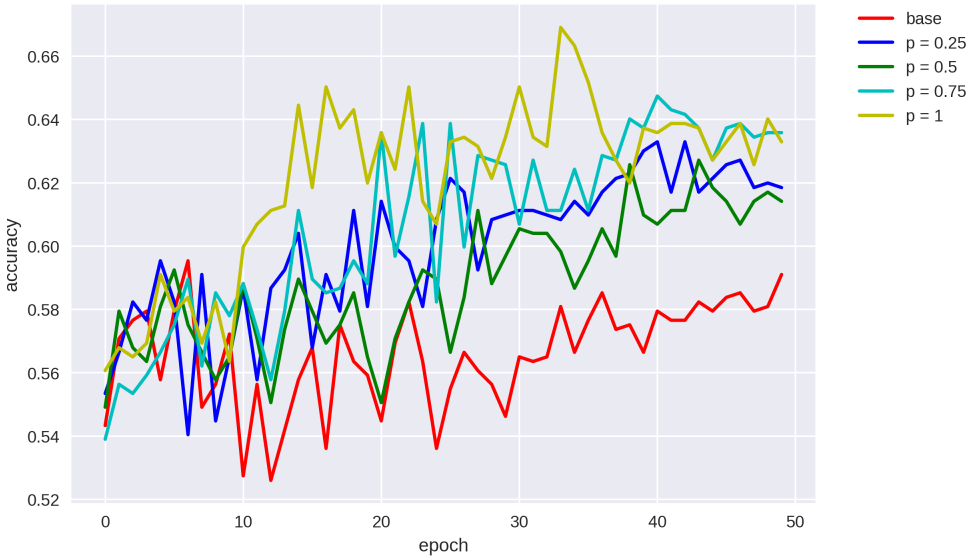


Figura 5.22.: Comparativa precisión Resnet18

Con una probabilidad p baja ya se observa una mejoría significativa tanto en las funciones de perdida como en la precisión de la clasificación, se observa un menor grado de sobre-ajuste ya que ambas funciones de perdida disminuyen de forma mas pareja.

A medida que vamos aumentado la probabilidad y por ende el numero de imágenes a las que se le aplicara la transformación parece evidente que se esta reduciendo aun mas el grado de sobre-ajuste.

La precisión en la clasificación supera el 60 % para todos los experimentos menos el base. Son los valores mas altos obtenido en los experimentos hasta ahora realizados y la trayectoria de las funciones de perdida parece indicar una bajada en el grado de sobre-ajuste al ser ambas descendientes e ir mas o menos a la par aunque aun sigue existiendo.

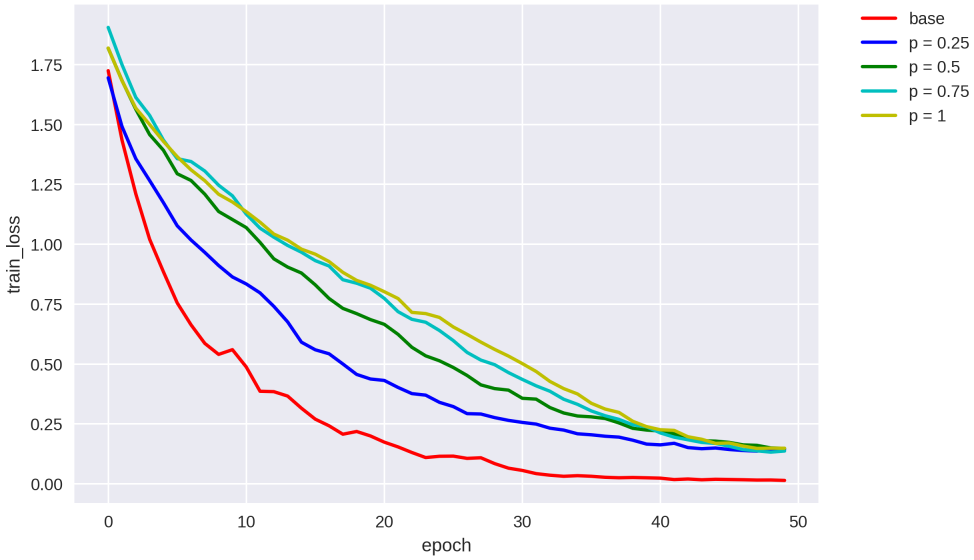


Figura 5.23.: Comparativa perdida en entrenamiento Resnet34

Figura 5.24.: Comparativa perdida en validación Resnet34

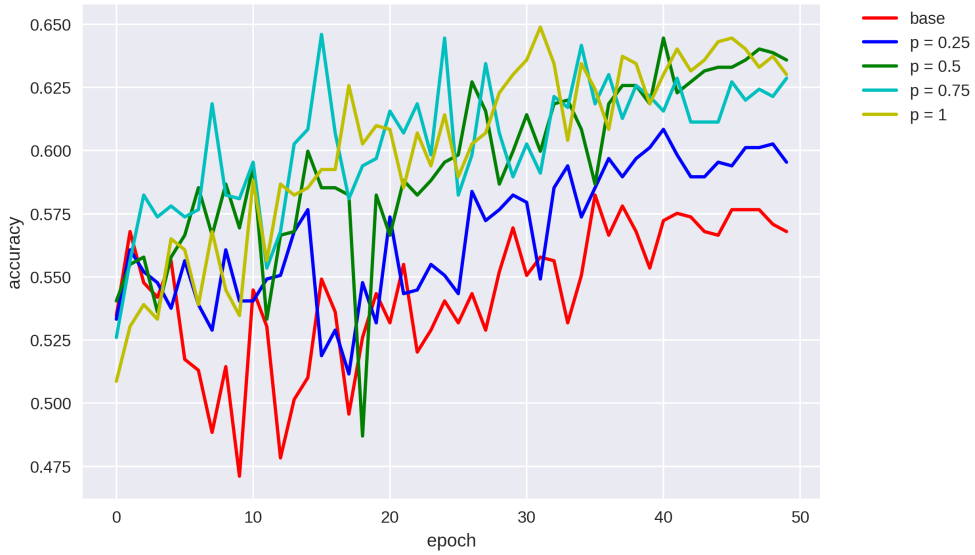


Figura 5.25.: Comparativa precisión Resnet34

Aquí también se observa una mejoría a medida que se va incrementando el valor p. Todos los experimentos acaban convergiendo con resultados similares para ambas perdidas pero son $p = 0.5$ $p = 0.75$ $p = 1$ los que obtienen la mejor precisión con resultados en torno al 63 %

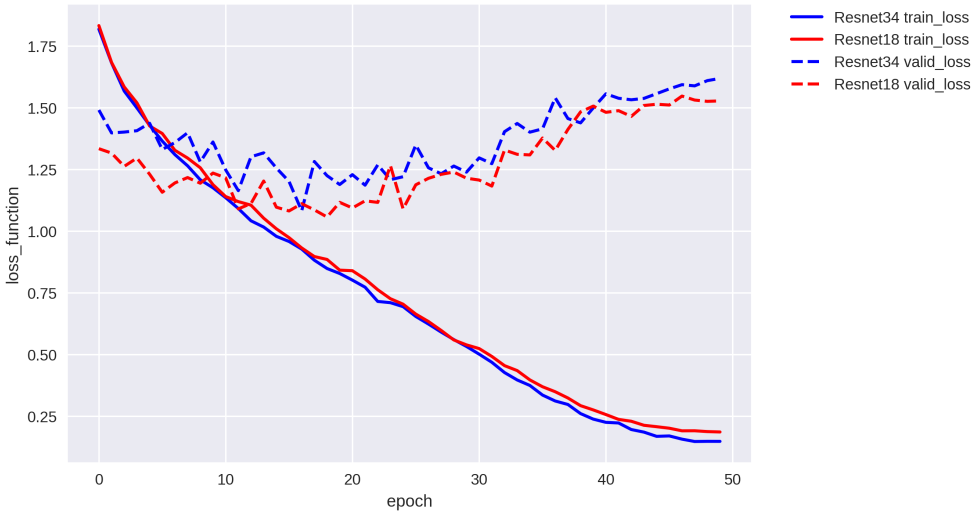


Figura 5.26.: Comparativa perdida p = 1 ResNet18 p = 1 Resnet 34

Figura 5.27.: Comparativa precisión p = 1 ResNet18 p = 1 Resnet 34

Comparando los mejores experimentos entre ambos modelos no observamos diferencias significativas entre las funciones de perdida. Ambas decrecen a ritmos similares siendo la perdida en validacion ligeramente inferior para ResNet18.

En cuanto a la precisión si que se observa muy parejo para ambos experimentos terminando ambos con una precision aproximada de 63 % aunque en epocas anteriores ResNet18 ha llegado a obtener resultados de hasta 66 %.

5.4.4. Label Smoothing

Fastai nos permite aplicar label smoothing de forma muy sencilla proporcionándonos de la función de perdida cross-entropy ya modificada, de tal manera que solo tenemos que cambiarla en el learner.

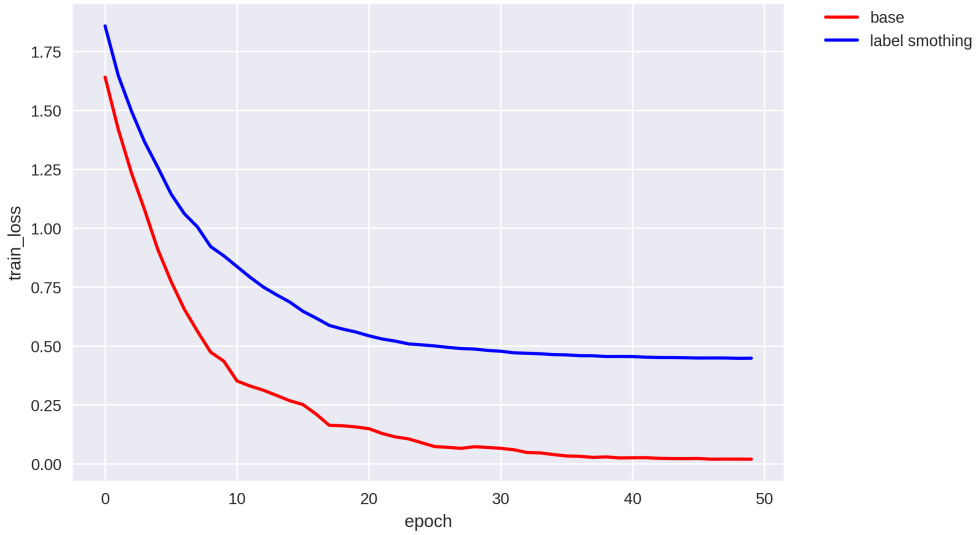


Figura 5.28.: Comparativa perdida en entrenamiento Resnet18

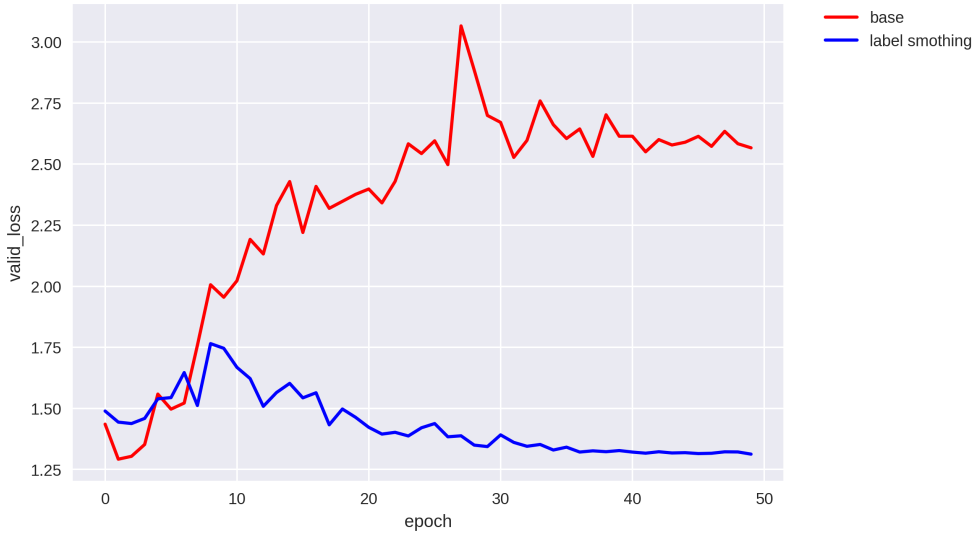


Figura 5.29.: Comparativa perdida en validación Resnet18

Comparando el modelo base con el modelo utilizando label smoothing se observa una mejora en el grado de sobre-ajuste aunque sigue existiendo, la perdida en validación es menor y con una trayectoria levemente descendente. Se aprecia también una leve mejora en la precisión de la clasificacion.

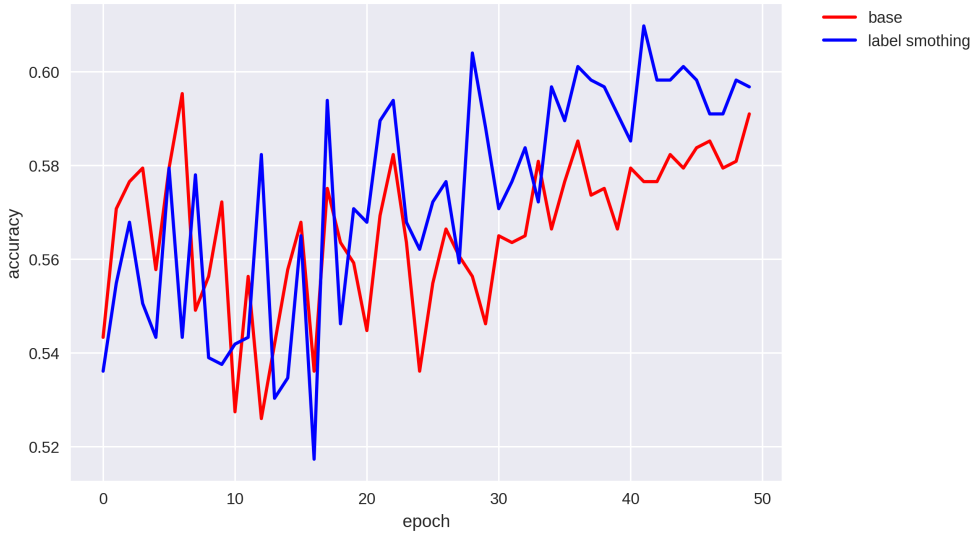


Figura 5.30.: Comparativa precisión Resnet18

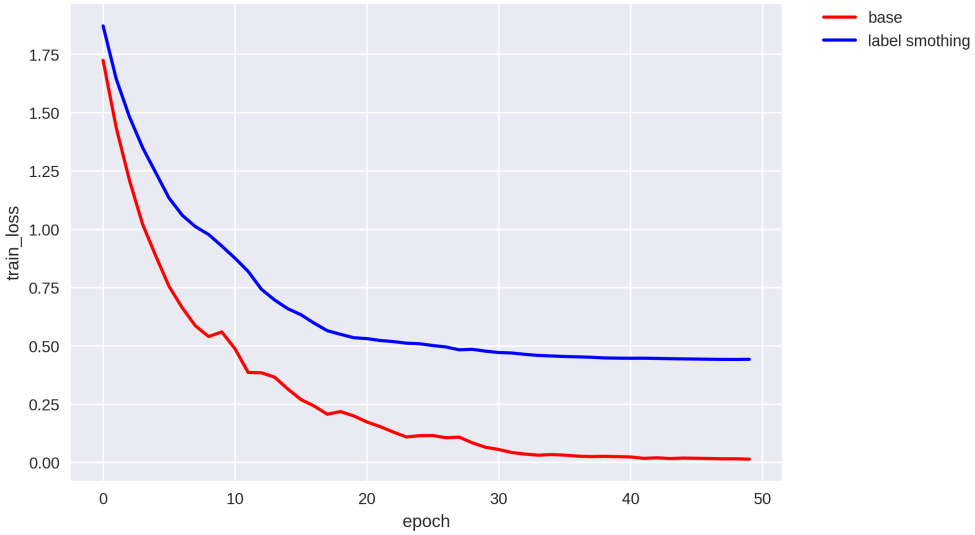


Figura 5.31.: Comparativa perdida en entrenamiento Resnet34

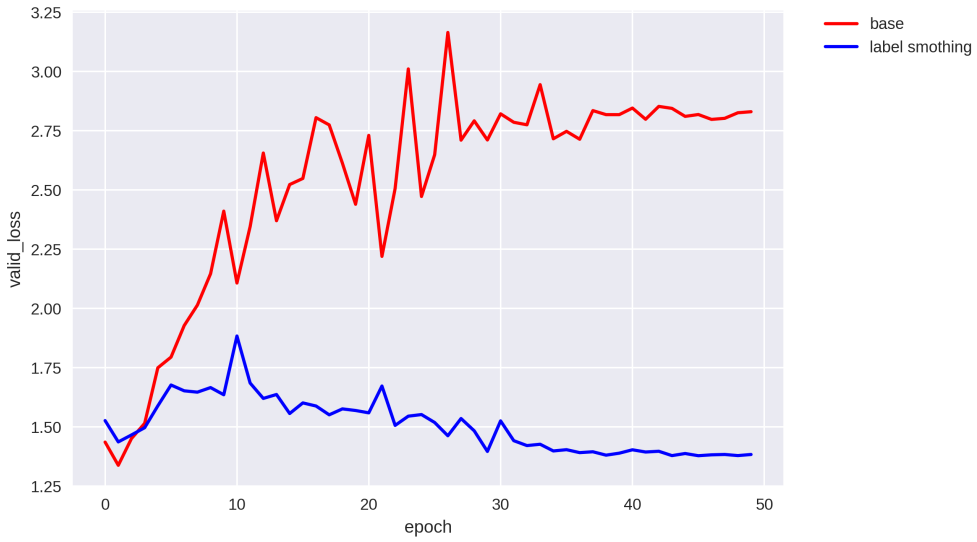


Figura 5.32.: Comparativa perdida en validación Resnet34

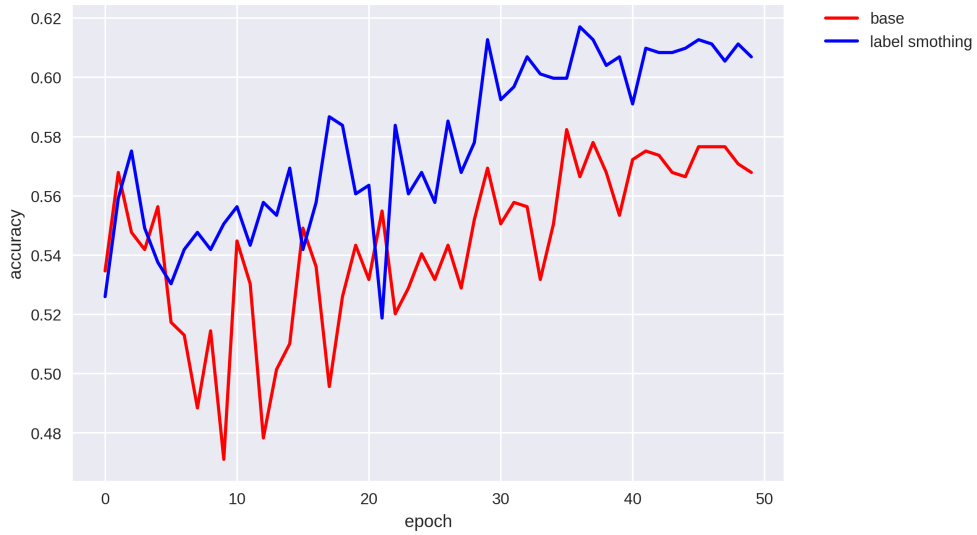


Figura 5.33.: Comparativa precisión Resnet34

Resultados bastante similares a los obtenidos utilizando ResNet18, trayectoria similar para la perdida en validación y también un incremento semejante en la precisión aunque levemente mayor

5.4.5. Experimento final

Se ha realizado data augmentation con varias transformaciones de diferentes tipos, ajustado parámetros para obtener mejores resultados y ademas realizado label smoothing para intentar crear un modelo menos confiado que pueda generalizar mejor las características de nuestras galaxias.

Todas estas técnicas se han aplicado de manera individual por si solas y se han comparado

con el modelo inicial correspondiente para determinar si se producía o no mejora. Ahora se van a aplicar estas técnicas de manera conjunta en un único experimento con los parámetros que mejor resultado proporcionaron individualmente para maximizar su efecto en la clasificacion.

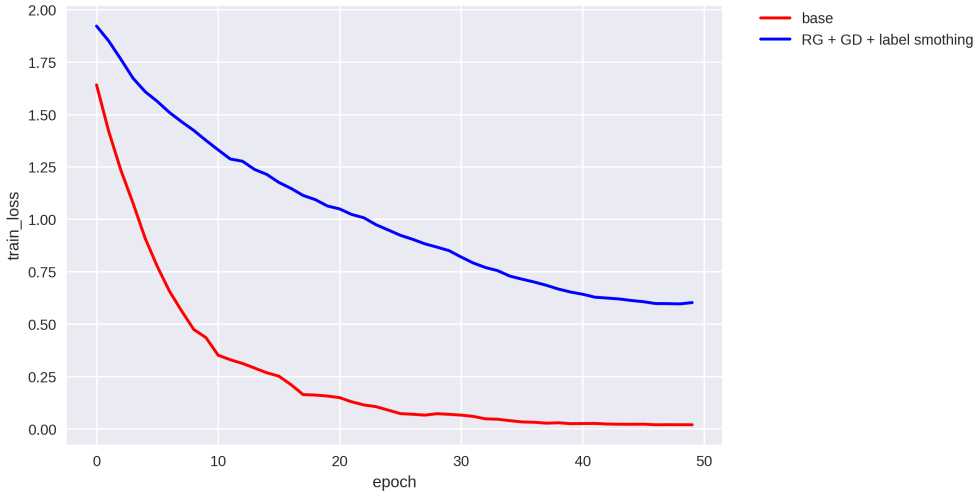


Figura 5.34.: Comparativa perdida en entrenamiento Resnet18

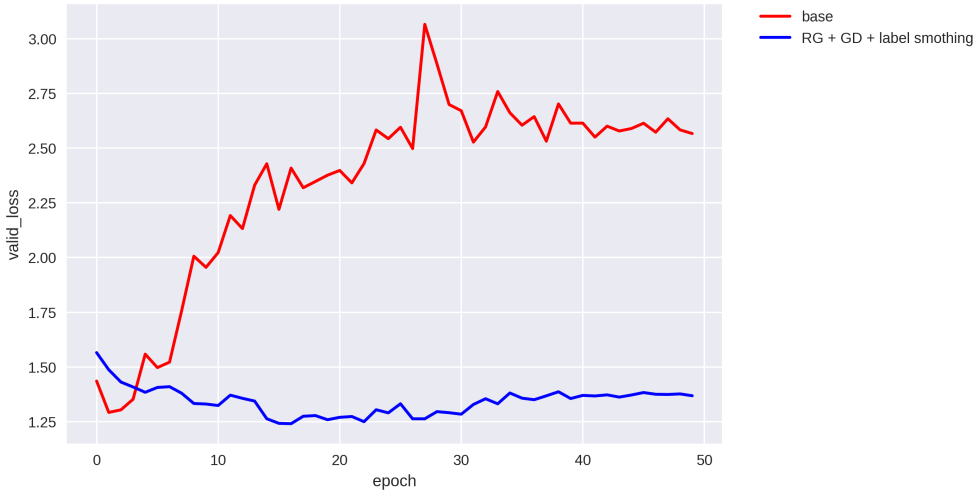


Figura 5.35.: Comparativa perdida en validación Resnet18

Figura 5.36.: Comparativa precisión Resnet18

La mejoría observada respecto al experimento base es notable tanto en la función de perdida como en la precisión de la clasificacion. La perdida en validación tiene una trayectoria descendente y evoluciona de forma bastante pareja a la perdida en entrenamiento, señal de que se ha eliminado el sobre-ajuste del experimento.

La precisión supera con creces a la precisión del experimento base superando el 65 % aunque no mejora demasiado con respecto a el experimento en el que solo se aplico la transformación de los giros a pesar de tener 3 veces mas épocas.

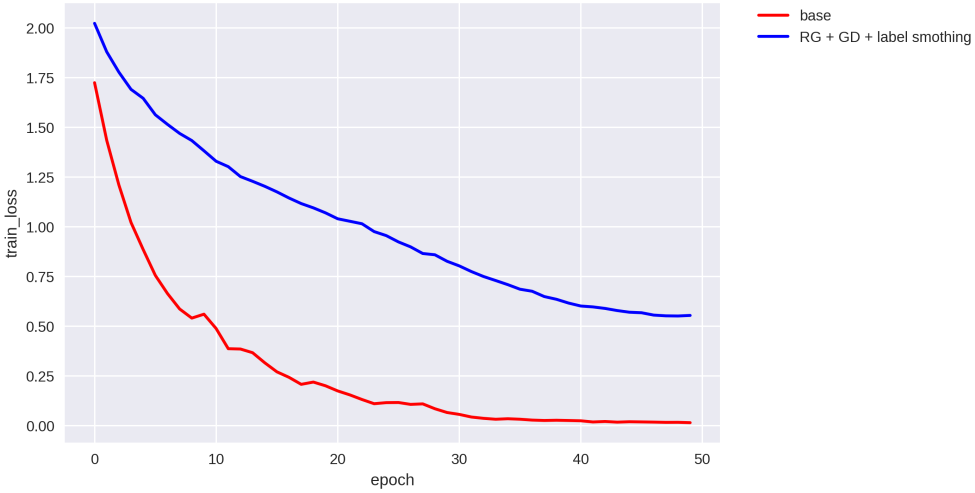


Figura 5.37.: Comparativa perdida en entrenamiento Resnet34

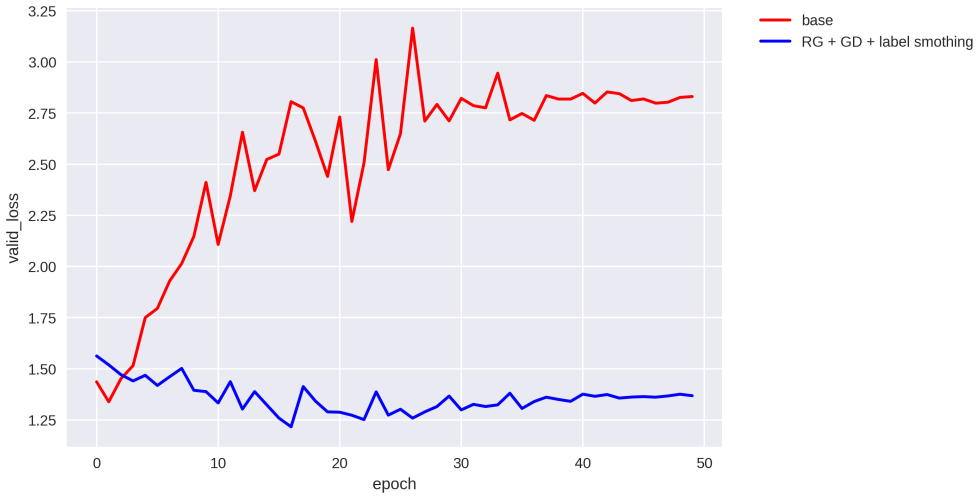


Figura 5.38.: Comparativa perdida en validación Resnet34

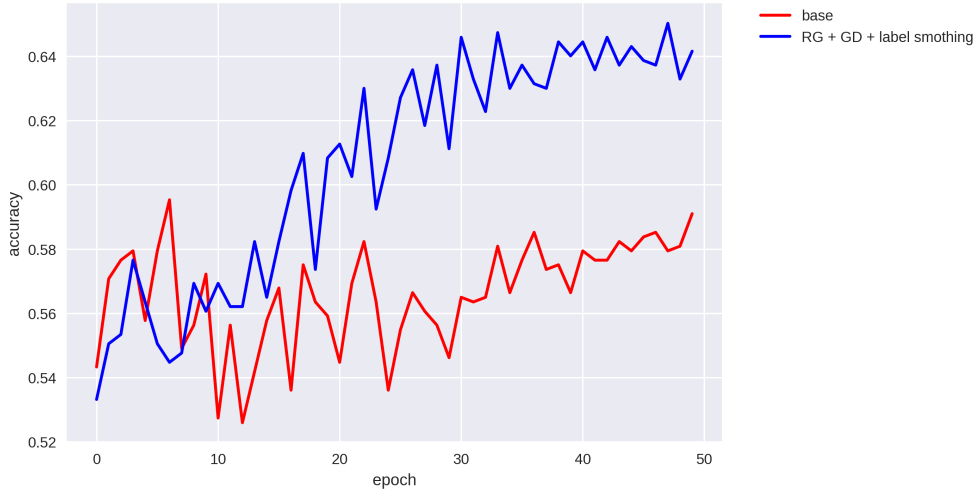


Figura 5.39.: Comparativa precisión Resnet34

Resultados bastante similares a los obtenidos utilizando ResNet18, trayectoria similar para la perdida en validación y también un incremento semejante en la precisión aunque levemente mayor. Ambos experimentos finales han proporcionado el mejor resultado de entre todos los experimentos para los dos modelos diferentes.

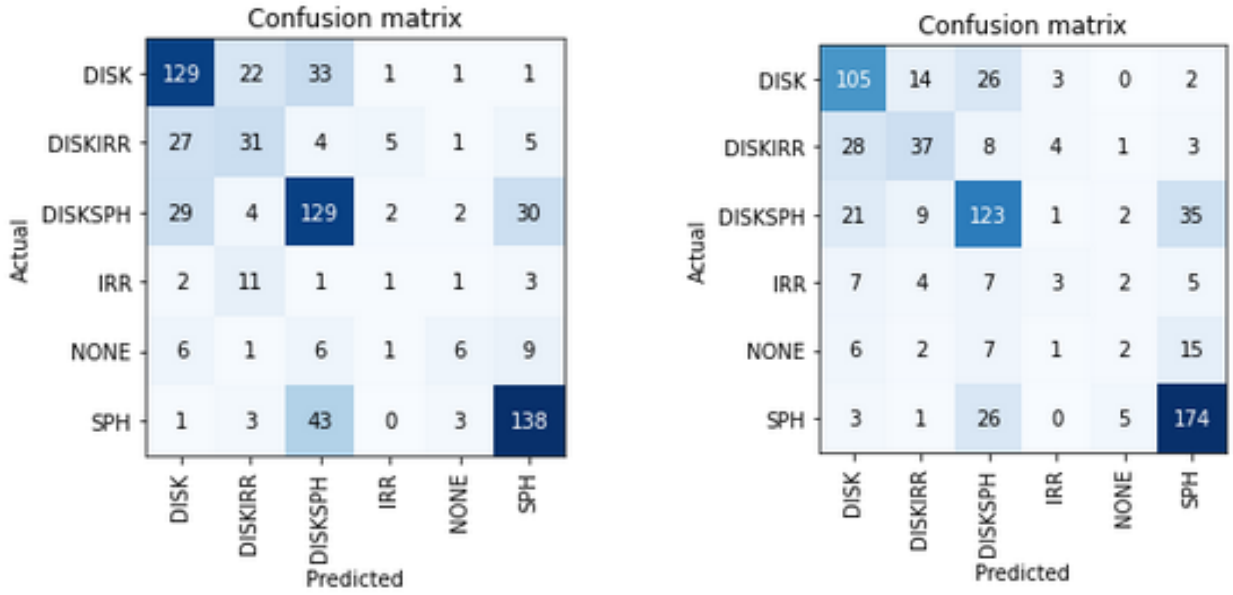


Figura 5.40.: Clasificaciones Experimento Final

Comparando los resultados de los experimentos finales para ambos modelos no se aprecia gran diferencia entre ellos, si observamos las matrices de confusión las clasificaciones se mantiene muy semejante la distribución de los resultados a la de los experimentos base siendo las mejor clasificadas SPH , DISK y DISKSPH . NONE y IRR siguen siendo muy mal clasificadas, esto es muy posiblemente debido al menor numero de imágenes pertenecientes a esta categoría en relación a las mejores clasificadas.

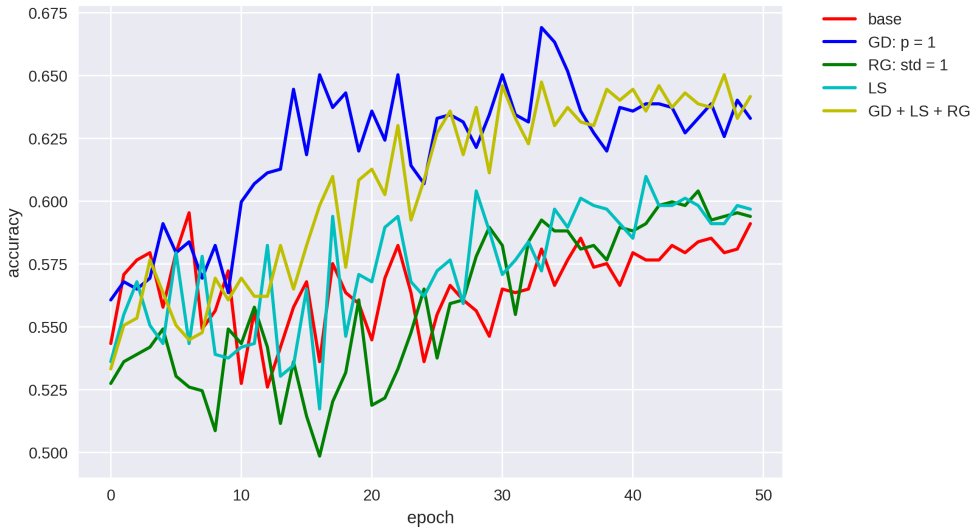


Figura 5.41.: Comparativa precisión final para los mejores experimentos ResNet18

Figura 5.42.: Comparativa precisión final para los mejores experimentos ResNet34

Observando en conjunto la precisión obtenida para los diferentes tipos de transformaciones para cada modelo identificamos que la transformación que mas a influido en el incremento de la precisión para ambos modelos son los giros diédricos. P ara ResNet18 tanto el experimento final (GD+LS+RG) como el experimento con giros (GD) obtienen resultados parecidos aunque la evolución del experimento final es mas estable debido al label smoothing.

Para ResNet34, de la misma manera, GD y GD+LS+RG alcanzan resultados semejantes aunque esta vez GD es mas estable en su evolución. Tanto el experimento con ruido gaussiano (RG) como label smoothing (LS) parecen tener un mayor efecto individualmente que para resNet18

6. Identificación del contorno de galaxias del espacio profundo

6.1. Descripcion del problema

De la misma manera que en la clasificacion morfológica, la identificación del contorno de una galaxia es un proceso que consume muchos recursos y que resulta prácticamente imposible de realizar en colecciones masivas de imágenes.

Las redes neuronales convolucionales (CNN) son arquitecturas que se desempeñan relativamente bien en otras tareas al margen de clasificación de imágenes aunque sean menos conocidas. Una de estas tareas es la regresión de imagen.

Se pueden entrenar modelos de CNNs para que dada una imagen, mediante regresión, se reconozcan ciertas características de la misma representadas de forma numérica. En nuestro caso esas características corresponderán a los parámetros con los que se puede obtener el contorno de la galaxia representada en la imagen.

La diferencia básica entre las tareas de clasificación de imágenes y regresión de imágenes es que la variable de destino (lo que estamos tratando de predecir) en la tarea de clasificación no es continua, mientras que en la tarea de regresión es numérica y continua.

Fastai nos permite realizar esta adaptación de forma sencilla modificando el tipo de datablock de salida.

6.2. Preparación de los datos

El conjunto de datos que se utilizara en esta experimentación contiene 1053 imágenes de galaxias de tipo disco seleccionadas del conjunto de imágenes anterior para las cuales se ha generado un fichero de datos csv. Este fichero csv contiene la información de 5 parámetros que dan forma a la elipse que representa el contorno de la galaxia.

Los parámetros contenidos en este archivo representan la proporción entre el eje mayor y el eje menor (q),el angulo del semieje mayor(pa), la longitud del semieje mayor(r_edge) y las coordenadas del centro(xcH,ycH).

Tabla 6.1.: Muestra datos archivo csv

gal_id	*q*	*pa*	*r_edge*	*xcH*	*ycH*
DISK_270_cosmos	0.2390	79.8405	31.859297	100.0	100.0
DISK_283_cosmos	0.2843	76.1063	17.744582	100.0	100.0
DISKSPH_886_cosmos	0.3449	-62.4947	35.659255	100.0	100.0
DISKSPH_944_cosmos	0.2000	14.0000	45.867763	100.0	100.0

En el experimento de clasificacion se utilizaron las mascaras para reducir el tamaño de las imágenes ya que solo eran necesarios los pixeles mas próximos a la galaxia y de esta forma se acelera el proceso de entrenamiento.

Para la identificación del contorno de una galaxia mediante regresión es posible que los pixeles mas alejados nos proporcionen información útil a la hora del entrenamiento por lo que utilizaremos las imágenes sin mascara aplicada.

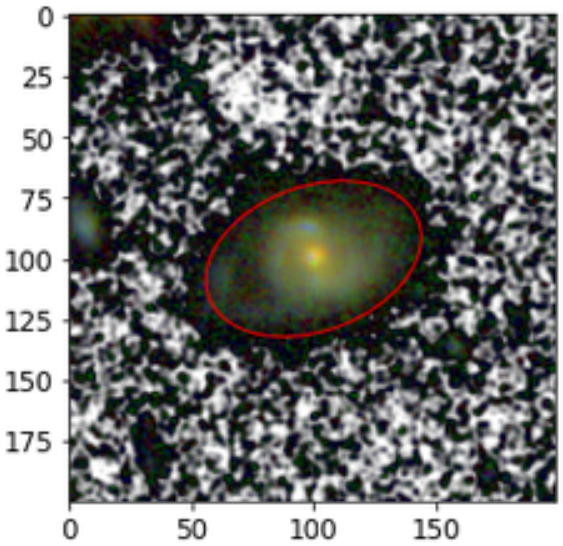


Figura 6.1.: Ejemplo imagen junto con su contorno original

6.3. Experimentación

6.3.1. Diseño

La experimentación se llevara a cabo de forma muy similar al capitulo anterior. Se realizara un experimento base sobre el que se realizaran sucesivas mejoras, primero se aplicaran técnicas de data augmentation y acto seguido se modificara el learning rate del modelo intentando ajustarlo al valor mas optimo posible para nuestro modelo particular.

Para estos experimentos utilizaremos solo la arquitectura que mejor desempeño en los experimentos de clasificación siendo esta la ResNet34.

6.3.2. Experimento base

La diferencia básica entre las tareas de clasificación de imágenes y regresión de imágenes es que la variable de destino (lo que estamos tratando de predecir) en la tarea de clasificación no es continua, mientras que en la tarea de regresión es numérica y continua.

Fastai nos permite realizar esta adaptación de forma sencilla modificando el tipo de datablock de salida.

La parametría inicial utilizada en este experimento es idéntica a la utilizada en el experimento base de clasificacion del capitulo anterior

Realizaremos 120 épocas de entrenamiento para el experimento, se ha elegido un numero alto de épocas debido a que el modelo ResNet34 fue entrenado para clasificacion por lo que necesitara mas tiempo hasta alcanzar resultados aceptables.

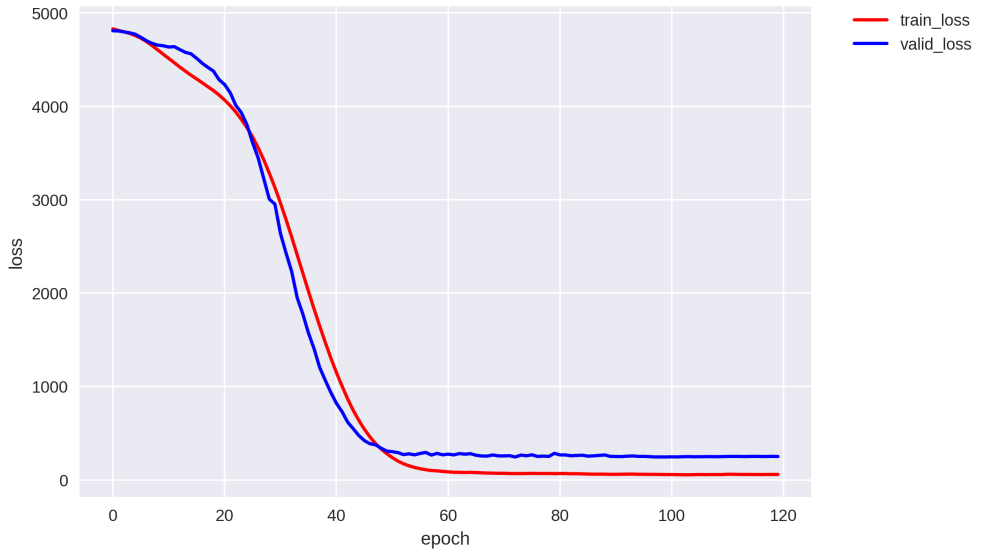


Figura 6.2.: Funciones de perdida en experimento base

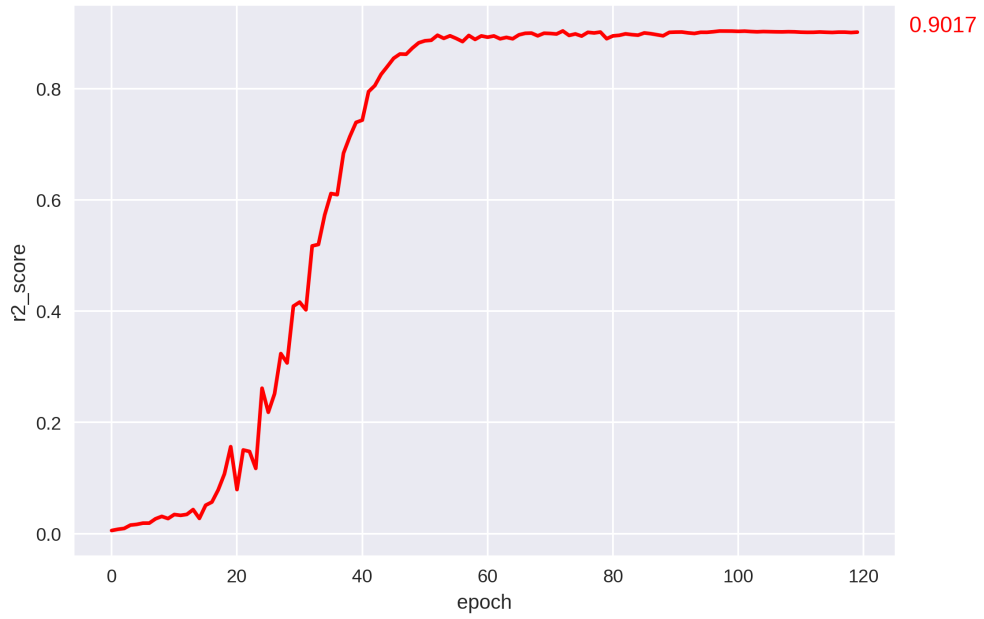


Figura 6.3.: Coeficiente de determinación en experimento base

Al no ser un problema de clasificacion no disponemos de la misma función de precisión para evaluar los resultados de nuestro modelo. Para sustituirla se han utilizado el coeficiente de determinación R^2.

En las figuras podemos observar las gráficas de la variación a lo largo de las épocas de las funciones de perdida y el coeficiente de determinación.

Se observa una función de perdida inicial con unas métricas bastante pobres que mejora rápidamente hasta estabilizarse alrededor de la época 50. A partir de este punto la función de perdida en

validación se mantiene relativamente constante con una coeficiente de determinación aproximado de 0.88

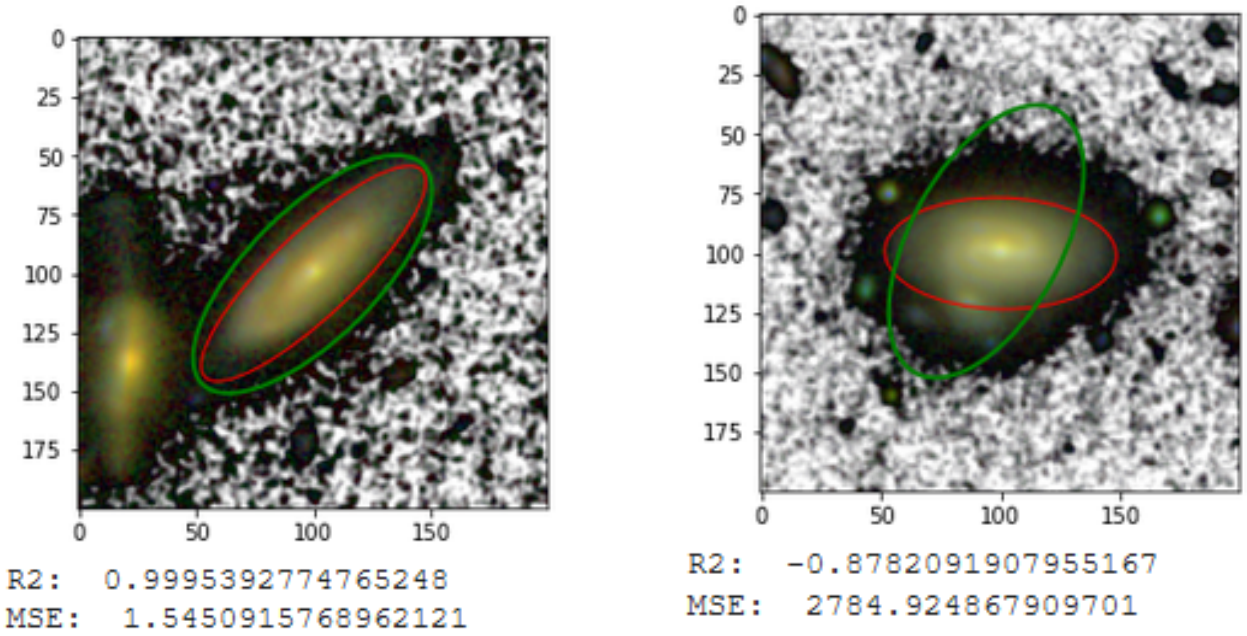


Figura 6.4.: Mejor y peor resultado experimento base

	q	*pa*	*r_edge*	*xcH*	*ycH*
Peor Original	0.4823	88.6653	48.4754	100.0	100.0
Peor Predicción	0.5089	-28.2302	62.6002	94.0264	95.0150
Mejor Original	0.2784	-46.5121	63.8312	100.0	100.0
Mejor Predicción	0.4160	-45.3469	66.3043	99.6037	100.2759

Si observamos las imágenes con el mejor y peor resultado en el peor la función del error cuadrático medio MSE utilizada para el entrenamiento y el coeficiente de determinación R2 indican un ajuste bastante pobre. La elipse roja representa el contorno original de la galaxia y la elipse verde el contorno generado después de el entrenamiento.

La gran diferencia viene dada por el valor predicho en el angulo ya que para estos valores contamos tanto con ángulos negativos como positivos.

6.3.3. Data Augmentation

A continuación se realizara data Augmentation de la misma manera que se realizo en los experimentos de clasificacion.

En este caso solo aplicaremos transformaciones de ruido gaussiano por la naturaleza de los datos de entrada. Si aplicáramos giros a las imágenes la variable dependiente dejaría de ajustarse a dicha imagen ya que representa los parámetros del contorno de la elipse que contiene a la galaxia en si.

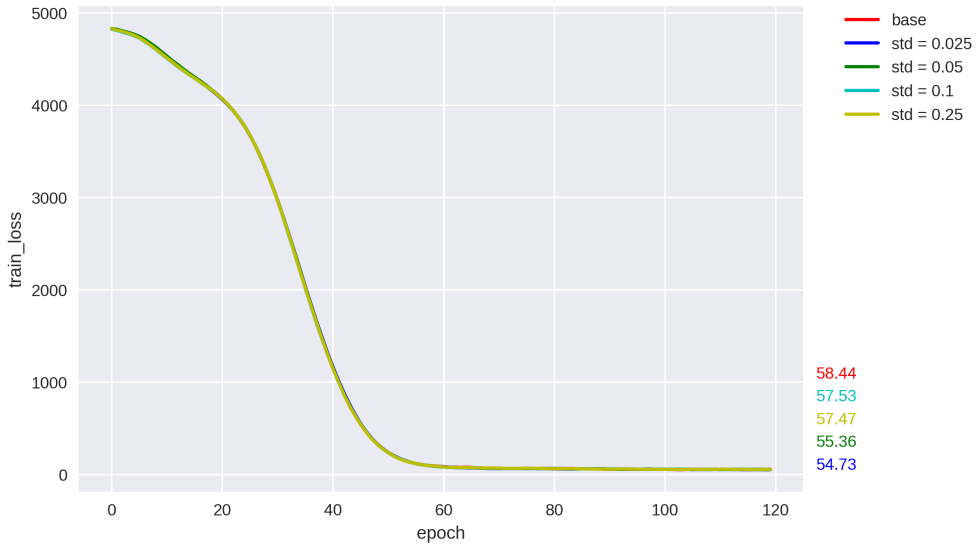


Figura 6.5.: Perdida entrenamiento experimentos con ruido

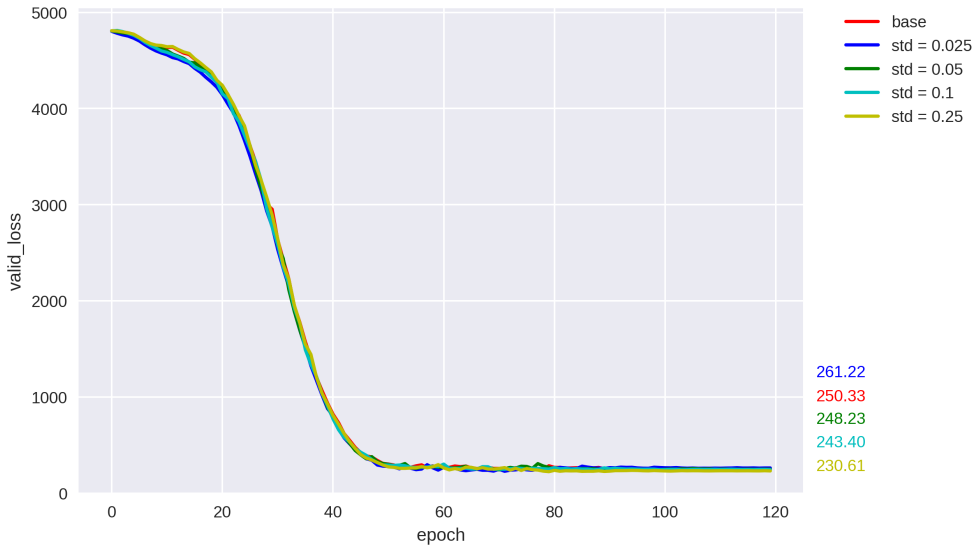


Figura 6.6.: Perdida validación experimentos con ruido

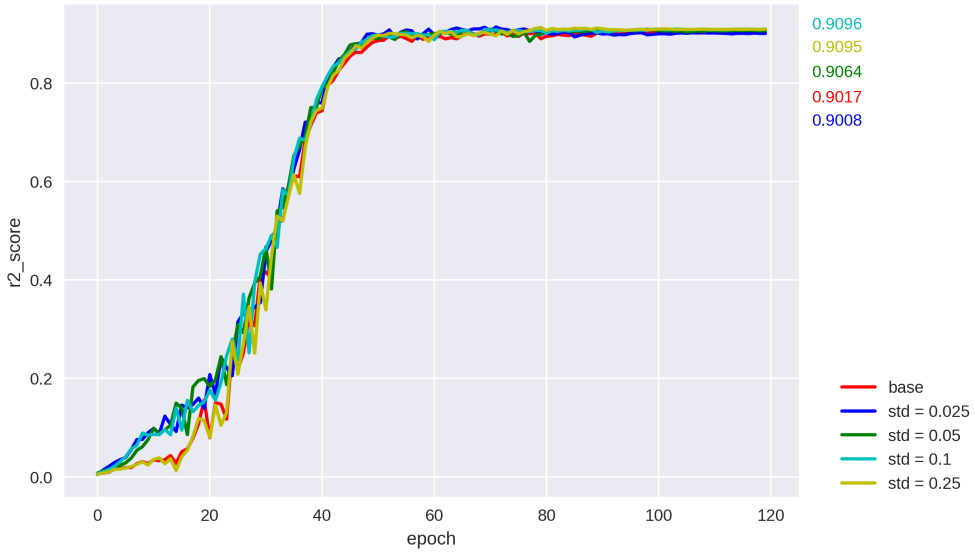


Figura 6.7.: coeficiente de determinación experimentos con ruido

A la derecha de las gráficas se observa el valor del ultimo punto de la curva con su respectivo color para facilitar su análisis. No se observa mejora aparente en el coeficiente de determinación ni en la función de perdida para ninguno de los diferentes experimentos.

Tanto la perdida en validación como en entrenamiento empiezan con valores muy altos para decrecer rápidamente hasta aproximadamente la época 50 donde el decrecimiento se ralentiza hasta estabilizarse a partir de la época 60. De igual manera el coeficiente de determinación se incrementa rápidamente hasta la 50 donde empieza a estabilizarse entorno a 0.9 para todos los experimentos.

De entre los experimentos con transformaciones de ruido gaussiano realizados, aunque sin diferencias significativas, el que mejor coeficiente de determinación a obtenido es el realizado con varianza 0.1

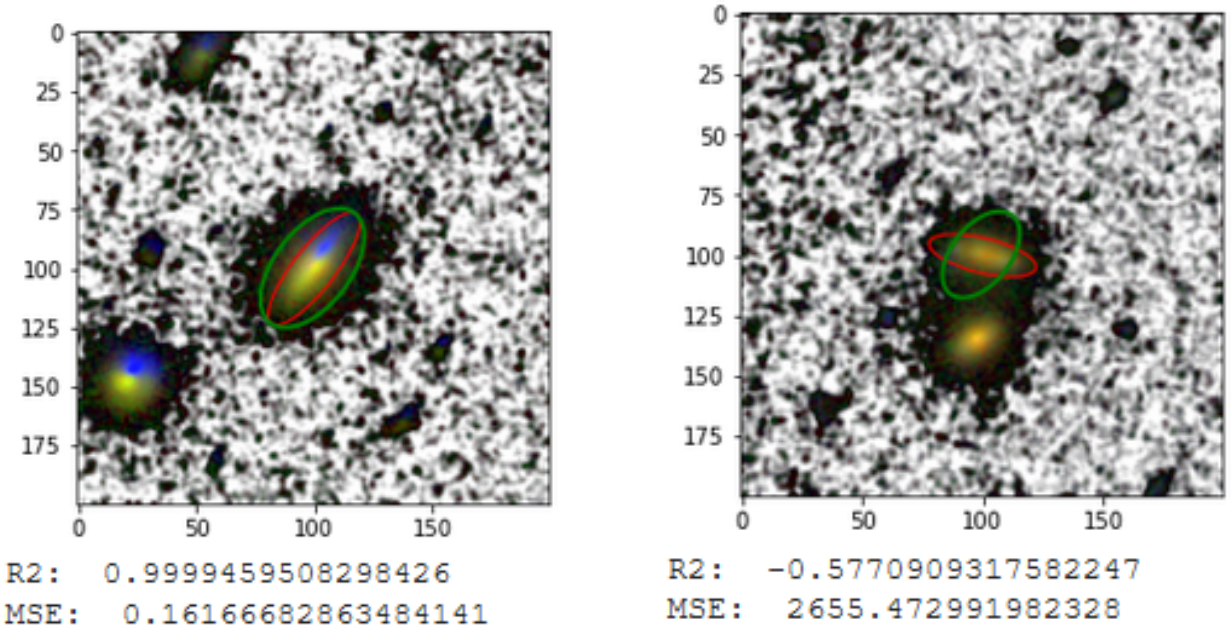


Figura 6.8.: Mejor y peor resultado experimento con ruido gaussiano std=0.1

	q	*pa*	*r_edge*	*xcH*	*ycH*
Peor Original	0.3172	76.5334	22.8040	100.0	100.0
Peor Predicción	0.5841	-38.6735	20.7667	99.3863	99.6338
Mejor Original	0.3	-38.0	29.5769	100.0	100.0
Mejor Predicción	0.5491	-37.7461	29.3594	99.2626	99.6989

6.3.4. Seleccionando la tasa de aprendizaje

La tasa de aprendizaje es el parámetro que escala la magnitud de nuestras actualizaciones del peso para minimizar la función de pérdida de la red. Dependiendo de como se seleccione pueden darse 3 tipos de escenarios, si la tasa de aprendizaje es muy pequeña las actualizaciones de los pesos serán también muy reducidas y la velocidad del entrenamiento muy lenta, por lo tanto la red tardara muchas mas épocas en llegar a al mínimo de la función de perdida.

Si por el contrario la tasa es muy elevada los saltos entre las actualizaciones son mayores y se podría llegar a dar divergencia y que a mayor numero de épocas peores resultados se obtendrían.

Figura 6.9.: Diferentes escenarios para una tasa de aprendizaje

https://www.jeremyjordan.me/nn-learning-rate/

No existe ninguna solución para averiguar cual seria la tasa de aprendizaje ideal para tu modelo pero existen métodos para obtener un rango inicial de donde obtenerla e ir mejorandola mediante prueba y error.

Cuando creamos un modelo a partir de una red previamente entrenada, fastai congela automáticamente todas las capas previamente entrenadas para nosotros. Cuando llamamos al método $fine_tune$, fastai hace dos cosas:

- Entrena las capas agregadas aleatoriamente para una época, con todas las demás capas congeladas
- Descongela todas las capas y las entrena todas para la cantidad de épocas solicitadas

Aunque este es un enfoque predeterminado razonable, es probable que para un conjunto de datos en particular se obtengan mejores resultados si hacen las cosas de manera ligeramente diferente.

Fastai nos proporciona la función $lr\_find(start\_lr = 1e{-}07, end\_lr = 10, num\_it = 100, stop\_div = True, show\_plot = True, suggestions = True)$ la cual realiza una simulación de entrenamiento en el modelo con tasas de aprendizaje en crecimiento exponencial desde $start\_lr$ hasta end_lr para un num_it y se detiene en caso de divergencia (a menos que $stop\_div = False$) y luego muestra en un gráfico las pérdidas frente a las tasas de aprendizaje en escala logarítmica.

Este método nos ofrece diferentes valores de sugerencia los cuales podemos utilizar como rango y candidatos para seleccionar nuestra tasa de entrenamiento dependiendo de la función elegida, entre ellos esta:

- El punto con el descenso de perdida mas pronunciado
- El punto con el mínimo valor de perdida antes de la divergencia dividido por 10
- Algoritmo de Novetta (slide)
- Algoritmo de ESRI(valley)

El método de Novetta se desarrolló teniendo en cuenta las tareas de procesamiento del lenguaje natural para automatizar mejor la implementación de modelos ML en la plataforma Novetta Mission Analytics. El algoritmo de ESRI se enfoca en tareas de visión informatizada.

Si ejecutamos la función para nuestro modelo anterior obtenemos los siguientes resultados:

- **minimum** = 0.0389
- **steep**= 0.0075
- **valley(ESRI)** = 0.0069
- **slide(Novetta)**= $9.9999e^{-}07$

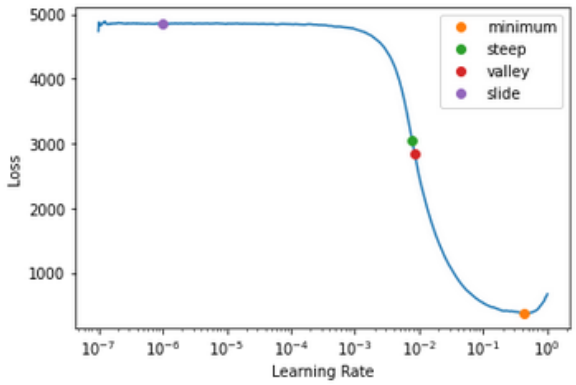


Figura 6.10.: Evolucion perdida en función del learning rate inicial

El método ESRI es el método por defecto y el que genera mejores resultados generalmente así que utilizaremos este ratio de aprendizaje inicial en nuestro modelo.Entrenaremos el modelo de forma manual utilizando la función fit_one_cycle por 10 etapas con transformación de ruido gaussiano de varianza 0.1 y veremos como reacciona.

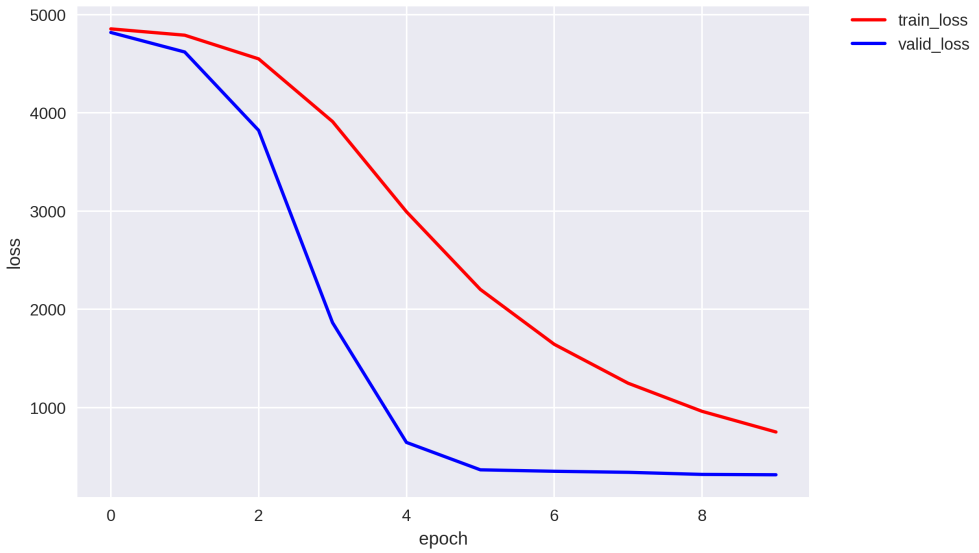


Figura 6.11.: Funciones de perdida tras 10 épocas

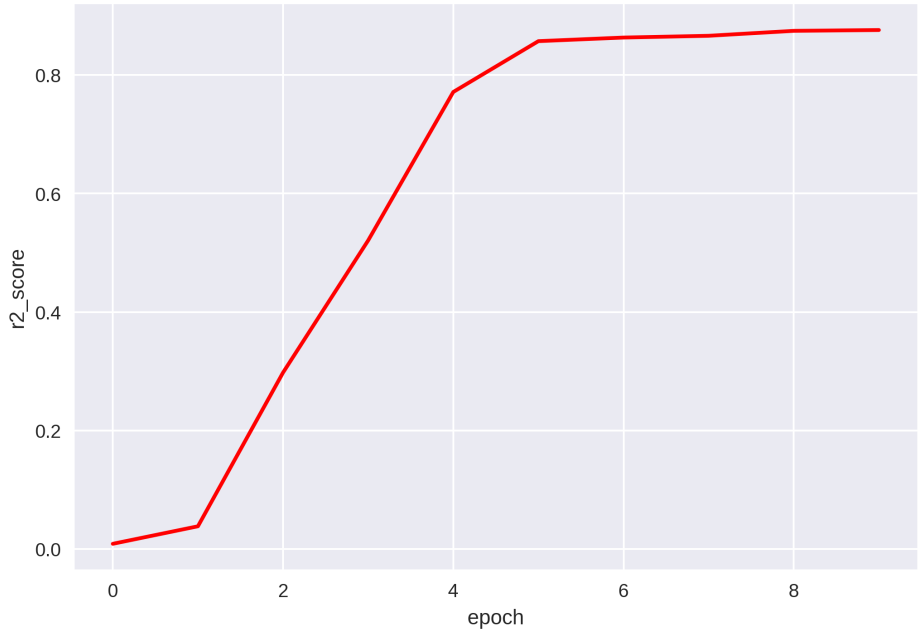


Figura 6.12.: Coeficiente de determinación tras 10 épocas

Descongelamos el modelo y ejecutamos la función lr_find de nuevo porque tener más capas para entrenar y pesos que ya se han entrenado durante tres épocas, significa que nuestra tasa de aprendizaje encontrada anteriormente ya no es apropiada.

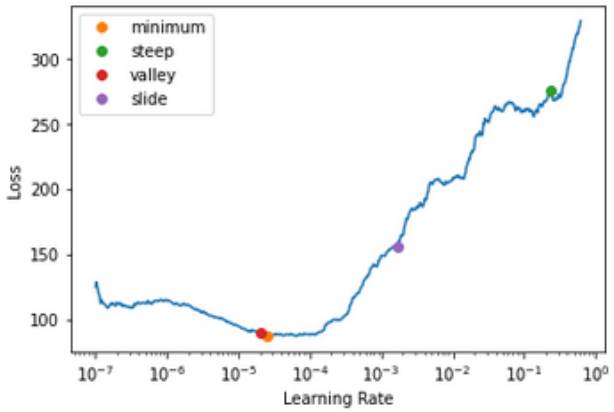


Figura 6.13.: Evolución perdida en función del learning rate tras 10 epocas

El gráfico es un poco diferente de cuando teníamos pesos aleatorios, no tenemos ese descenso brusco que indica que el modelo está entrenando. Eso es porque nuestro modelo ya ha sido entrenado. Aquí tenemos un área algo plana antes de un fuerte aumento, y deberíamos tomar un punto mucho antes de ese fuerte aumento.

Por defecto fastai utiliza tasas de aprendizaje discriminativas, una tasa de aprendizaje más baja para las primeras capas de la red neuronal y una tasa de aprendizaje más alta para las capas posteriores. La idea se basa en conocimientos desarrollados por Jason Yosinski, quien demostró en 2014 que con el aprendizaje por transferencia, diferentes capas de una red neuronal deben entrenarse a diferentes velocidades.

Continuaremos entrenando el modelo durante otras 30 épocas con un lr inicial de $1e^{-6}$ y un lr final de $1e^{-4}$

El primer valor será la tasa de aprendizaje en la capa más temprana de la red neuronal y el segundo valor será la tasa de aprendizaje en la capa final. Las capas intermedias tendrán tasas de aprendizaje que son equidistantes en todo el rango.

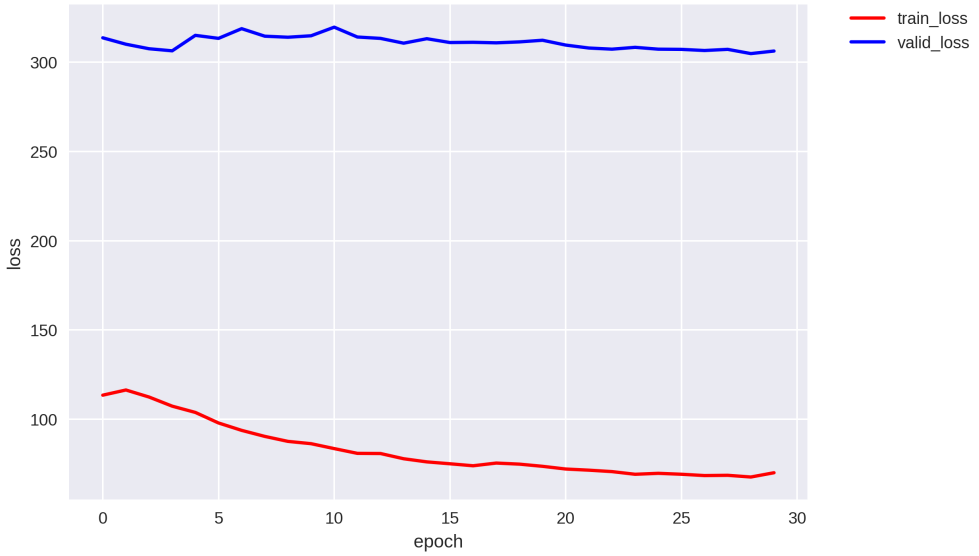


Figura 6.14.: Funciones perdida ultimas 30 épocas

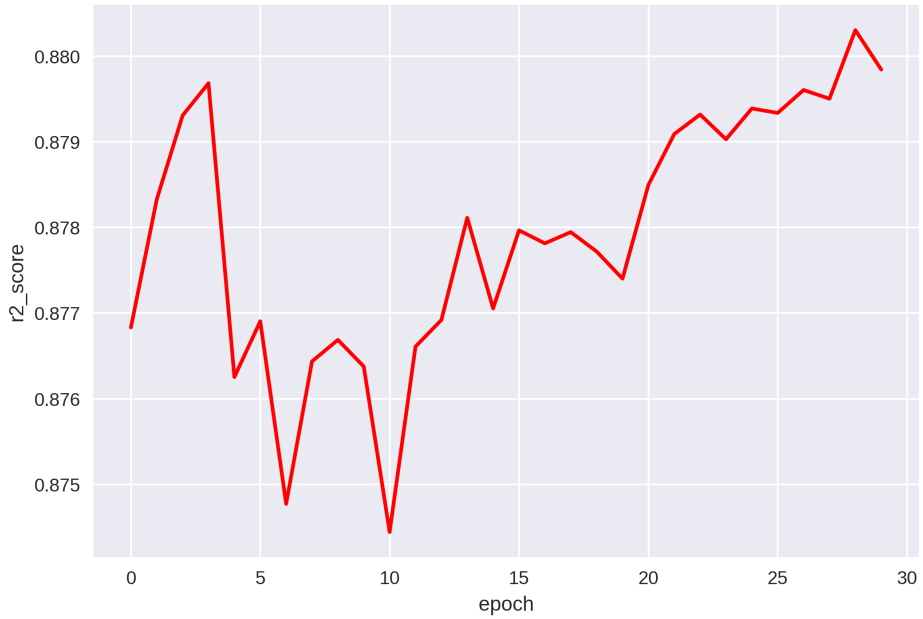


Figura 6.15.: Coeficiente de determinación ultimas 30 épocas

Los resultados no mejoran los resultados obtenidos en el experimento previo utilizando el ratio de aprendizaje estándar. El coeficiente de determinación en este caso se encuentra alrededor de 0.88 que es 0.03 puntos inferior al coeficiente previo obtenido que era de 0.91. Sin embargo el numero de épocas que se ha tenido que entrenar el modelo para conseguir resultados similares se ha visto reducido considerablemente. Se sigue apreciando un ligero incremento en el coeficiente de determinación por lo que vamos a realizar la misma técnica utilizando 120 épocas como el resto de experimentos.

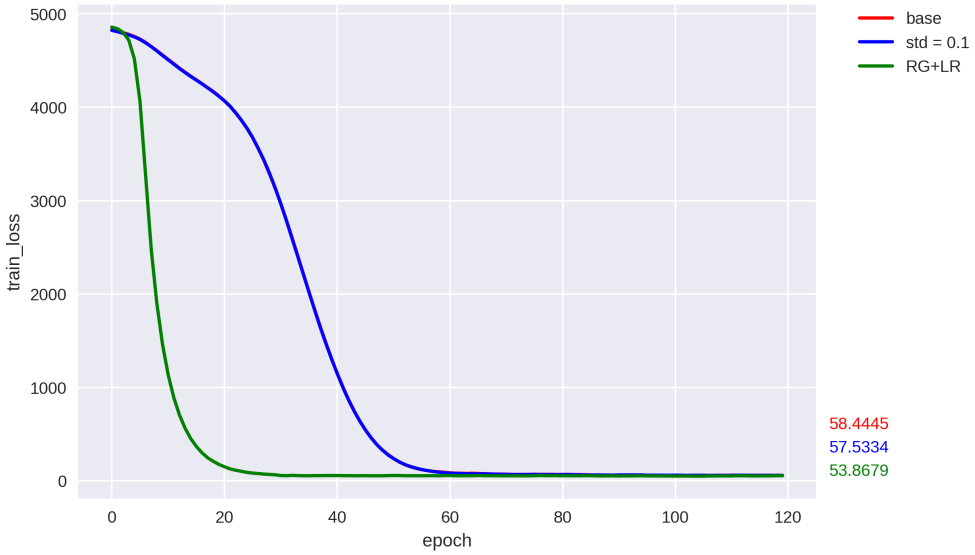


Figura 6.16.: Comparativa funciones perdida entrenamiento finales

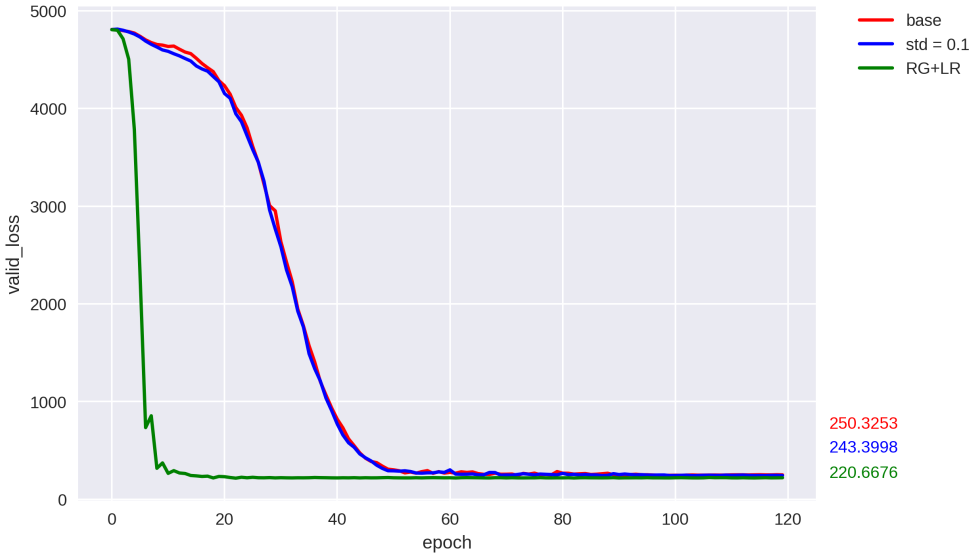


Figura 6.17.: Comparativa funciones perdida entrenamiento finales

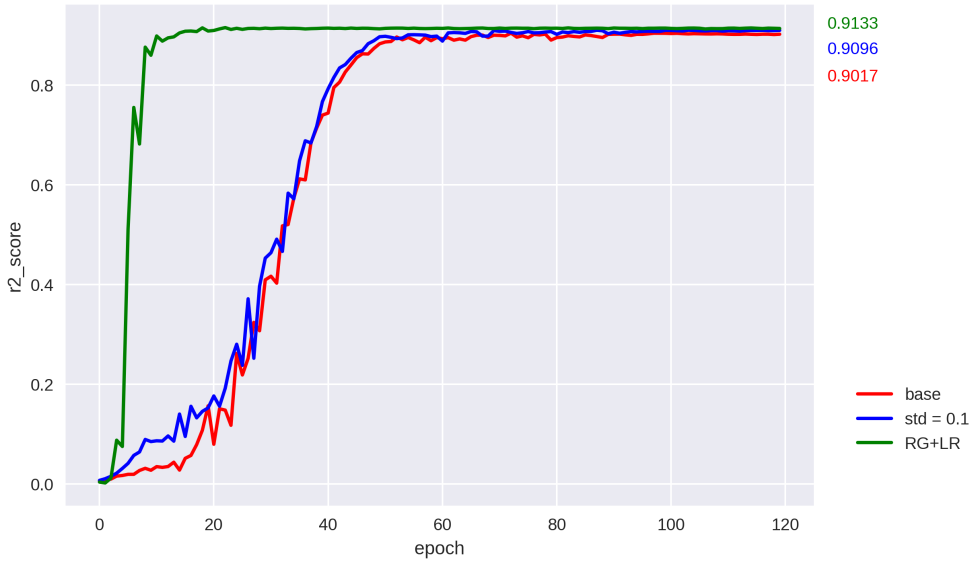


Figura 6.18.: Comparativa coeficiente de determinación final

Realizando las 120 épocas vemos que el coeficiente de determinación se estabiliza en torno a 91 % ligeramente superior tanto al experimento con ruido gaussiano como al base. la diferencia principal es el ratio de aprendizaje ya que es muy superior en el experimento final, siendo el decrecimiento de las funciones de perdida y el incremento del coeficiente de determinación mucho mas rápidos.

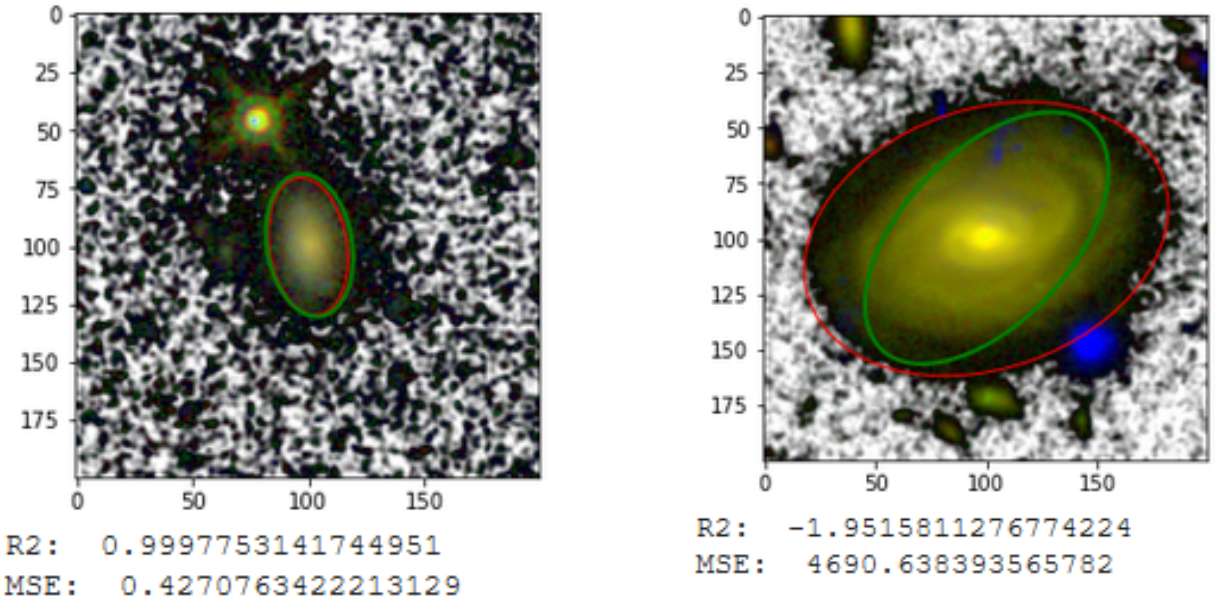


Figura 6.19.: Mejor y peor resultado experimento con ratio de aprendizaje modificado

	q	*pa*	*r_edge*	*xcH*	*ycH*
Peor Original	0.6899	109.0	84.6226	100.0	100.0
Peor Predicción	0.5531	-43.3574	69.1245	100.2431	99.6290
Mejor Original	0.5630	9.0811	29.8070	100.0	100.0
Mejor Predicción	0.5950	9.2579	31.0430	100.5879	99.5205

7. Conclusiones

En este trabajo se ha realizado una un trabajo de investigación sobre el campo de la astroinformatica y la aplicación de una metodología Deep Learning para los problemas de clasificacion e identificación de contornos de galaxias. Se ha utilizado el software de fastai para montar y entrenar las redes neuronales convolucionales.

En la clasificacion de galaxias se realizo un experimento base para 2 modelos ResNet diferentes (Resnet18 y ResNet34) a los que sucesivamente se les fue aplicando data Augmentation con giros y ruido gaussiano. Se ha realizado un análisis de sensibilidad para los parámetros principales de ambas transformaciones con el objetivo de seleccionar el experimento con mejores resultados entre ambos modelos.

Se observa cierto sobre-ajuste para los experimentos base y con ruido gaussiano que se mejora con la introducción de las transformaciones con giros. Se ha conseguido mejorar la precisión de la clasificación cerca de 10 puntos porcentuales pasando de entorno a un 54 % en el experimento base para ambos modelos a un 65 % en el ultimo experimento para el modelo ResNet18 y un 67 % para Resnet34.

Analizando la precisión de la clasificacion mas en detenimiento se observa que las imágenes peor clasificadas corresponden a las galaxias pertenecientes a tipos mixtos y que representan una proporción del total de imágenes significativamente menor.El modelo no es capaz de aprender bien las características de estas imágenes al tener muy pocas en comparación con otros tipos.

En la identificación del contorno de galaxias se ha realizado una modificación a la capa de salida en la red neuronal ResNet para ajustar un modelo de regresión que identifica los parámetros clave que conforman la elipse que rodea a la galaxia. Se ha seguido el mismo proceso con experimento base inicial con sucesivas modificaciones que en el apartado de clasificación.

Desde el experimento base ya se han obtenido resultados bastante aceptables, con un coeficiente de determinación del 0.89 que se ha incrementado ligeramente hasta el 0.91 mediante data Augmentation. Se ha incrementado ligeramente este coeficiente modificando el ratio de aprendizaje por defecto de forma manual en el experimento final.

Se han realizado visualizaciones para mostrar la elipse obtenida mediante el entrenamiento junto a la elipse obtenida con los datos originales. Las imágenes cuyos contornos están peor ajustados obtienen unos resultados muy dispares respecto al resto de imágenes que suelen obtener

resultados superiores al coeficiente de determinación medio. Esta disparidad viene dada por el angulo del semieje mayor de la elipse y se produce cuando el angulo original es negativo y el modelo después del entrenamiento asigna un angulo positivo.

Como trabajo futuro se podrían desarrollar modelos con redes neuronales convolucionales personalizadas y utilizar U-Nets mediante segmentación para la identificación del contorno de galaxias, ademas de utilizar software diferente para la construcción de los modelos como Keras o Tensorflow.

Referencias

Benne, W. (s.f.). *Source extractor for dummies.* Space Telescope Institute.

Bertin, E. (s.f.). *Sextractor draft v2.13 user's manual.* Institut Astrophysique Observatoire de Paris.

Espinosa, A., Ortiz, F., y Cereceda, T. (2019). *Astroinformática y prospección de la astronomía chilena: Sub-disciplina en el escenario global y desarrollo científico local.* Scielo.

Fastai. (s.f.). *Practical deep learning for coders.* (`https://course.fast.ai/`)

Fastai, D. (2020). *Fastai documentation.* (`http://neuralnetworksanddeeplearning.com/index.html`)

Fernique, P. (s.f.). *Aladin user's manual.* (`https://aladin.u-strasbg.fr/java/AladinManual6.pdf`)

Howard, J., y Gugger, S. (2020). *Deep learning for coders with fastai and pytorch: Ai applications without a phd.* O'Reilly Media, Incorporated.

Huertas Company M. (2015). *A catalog of visual-like morphologies in the 5 candels fields using deep-learning.* ApjS.

Mueller, Z. (2021a). *Different outputs on function lrfind.* (`https://forums.fast.ai/t/new-lr-finder-output/89236`)

Mueller, Z. (2021b). *Methods for automating learning rate finders.* (`https://www.novetta.com/2021/03/learning-rate/`)

Nielsen, M. (2019). *Neural Networks and Deep Learning.* (`http://neuralnetworksanddeeplearning.com/index.html`)

Stackoverflow.com. (s.f.). *Stack Overflow.* (`https://stackoverflow.com`)

Ujjwalkarn. (2016a). *An Intuitive Explanation of Convolutional Neural Networks.* (`https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`)

Ujjwalkarn. (2016b). *A quick intro of neuronal networks.* (https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/)

A. Anexo

CodigoExperimentoClasificacion

March 29, 2022

[ ]:
```
#hide
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

[ ]:
```
#hide
from fastbook import *
from fastai.vision.all import *
```

[ ]:
```
#Preparacion  experimento base res 34
path = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
data = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        )
df= data.dataloaders(path,bs=64)
learn = cnn_learner(df, resnet34,  metrics = accuracy
 ↪,cbs=[CSVLogger(fname="Res34Base30.csv")])
```

[ ]:
```
learn.fine_tune(50)
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento base res 18
path1 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
data1 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        )
df1 = data1.dataloaders(path1,bs=64)
```

```
learn1 = cnn_learner(df1, resnet18,  metrics = accuracy␣
 ↪,cbs=[CSVLogger(fname="Res18Base30.csv")])
```

[ ]:
```
learn1.fine_tune(50)
interp = ClassificationInterpretation.from_learner(learn1)
interp.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.25 res 34
path2 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs2 = Dihedral(p=0.25)
data2 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs2
        )
df2 = data2.dataloaders(path2,bs=64)
learn2 = cnn_learner(df2, resnet34,  metrics = accuracy,␣
 ↪cbs=[CSVLogger(fname="Res34D025.csv")])
learn2.fine_tune(50,cbs=[ShowGraphCallback()])
interp2 = ClassificationInterpretation.from_learner(learn2)
interp2.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.25 res 18
path2 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs2 = Dihedral(p=0.25)
data2 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs2
        )
df2 = data2.dataloaders(path2,bs=64)
learn2 = cnn_learner(df2, resnet18,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res18D025.csv")])
learn2.fine_tune(50,cbs=[ShowGraphCallback()])
interp2 = ClassificationInterpretation.from_learner(learn2)
interp2.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.5 res 34
path3 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs3 = Dihedral(p=0.5)
data3 = DataBlock(
```

```
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs3
        )
df3 = data3.dataloaders(path3,bs=64)
learn3 = cnn_learner(df3, resnet34,  metrics =
 ↪accuracy,cbs=[CSVLogger(fname="Res34D05.csv")])
learn3.fine_tune(50,cbs=[ShowGraphCallback()])
interp3 = ClassificationInterpretation.from_learner(learn3)
interp3.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.5 res 18
path3 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs3 = Dihedral(p=0.5)
data3 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs3
        )
df3 = data3.dataloaders(path3,bs=64)
learn3 = cnn_learner(df3, resnet18,  metrics =
 ↪accuracy,cbs=[CSVLogger(fname="Res18D05.csv")])
learn3.fine_tune(50,cbs=[ShowGraphCallback()])
interp3 = ClassificationInterpretation.from_learner(learn3)
interp3.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.75 res 34
path4 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs4 = Dihedral(p=0.75)
data4 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs4
        )
df4 = data4.dataloaders(path4,bs=64)
learn4 = cnn_learner(df4, resnet34,  metrics =
 ↪accuracy,cbs=[CSVLogger(fname="Res34D075.csv")])
learn4.fine_tune(50,cbs=[ShowGraphCallback()])
```

```
interp4 = ClassificationInterpretation.from_learner(learn4)
interp4.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 0.75 res 18
path4 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs4 = Dihedral(p=0.75)
data4 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs4
        )
df4 = data4.dataloaders(path4,bs=64)
learn4 = cnn_learner(df4, resnet18,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res18D075.csv")])
learn4.fine_tune(50,cbs=[ShowGraphCallback()])
interp4 = ClassificationInterpretation.from_learner(learn4)
interp4.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 1 res 34
path5 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs5 = Dihedral(p=1.0)
data5 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs5
        )
df5 = data5.dataloaders(path5,bs=64)
learn5 = cnn_learner(df5, resnet34,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res34D1.csv")])
learn5.fine_tune(50,cbs=[ShowGraphCallback()])
interp5 = ClassificationInterpretation.from_learner(learn5)
interp5.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Rotacion Dihedral p = 1 res 18
path5 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs5 = Dihedral(p=1.0)
data5 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
```

```
        get_y = parent_label,
        batch_tfms = tmfs5
        )
df5 = data5.dataloaders(path5,bs=64)
learn5 = cnn_learner(df5, resnet18,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res18D1.csv")])
learn5.fine_tune(50,cbs=[ShowGraphCallback()])
interp5 = ClassificationInterpretation.from_learner(learn5)
interp5.plot_confusion_matrix()
```

[ ]:
```
class GaussianNoise(RandTransform):

    def __init__(self, mean=0., std=1., **kwargs):
        self.std = std
        self.mean = mean
        super().__init__(**kwargs)

    def encodes(self, x:TensorImage):
        return x + torch.randn(x.size()).cuda() * self.std + self.mean
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.025 res 34
path6 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs6 = [GaussianNoise(mean=0., std =0.025)]
data6 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs6
        )
df6 = data6.dataloaders(path6,bs=64)
learn6 = cnn_learner(df6, resnet34,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res34RG0025.csv")])
learn6.fine_tune(50,cbs=[ShowGraphCallback()])
interp6 = ClassificationInterpretation.from_learner(learn6)
interp6.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.025 res 18
path6 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs6 = [GaussianNoise(mean=0., std =0.025)]
data6 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
```

```
        batch_tfms = tmfs6
        )
df6 = data6.dataloaders(path6,bs=64)
learn6 = cnn_learner(df6, resnet18,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res18RG0025.csv")])
learn6.fine_tune(50,cbs=[ShowGraphCallback()])
interp6 = ClassificationInterpretation.from_learner(learn6)
interp6.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.05 res 34
path7 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs7 = [GaussianNoise(mean=0., std =0.05)]
data7 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs7
        )
df7 = data7.dataloaders(path7,bs=64)
learn7 = cnn_learner(df7, resnet34,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res34RG005.csv")])
learn7.fine_tune(50,cbs=[ShowGraphCallback()])
interp7 = ClassificationInterpretation.from_learner(learn7)
interp7.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.05 res 18
path7 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs7 = [GaussianNoise(mean=0., std =0.05)]
data7 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs7
        )
df7 = data7.dataloaders(path7,bs=64)
learn7 = cnn_learner(df7, resnet18,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="Res18RG005.csv")])
learn7.fine_tune(50,cbs=[ShowGraphCallback()])
interp7 = ClassificationInterpretation.from_learner(learn7)
interp7.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.075 res 34
path8 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs8 = [GaussianNoise(mean=0., std =0.075)]
data8 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs8
        )
df8 = data8.dataloaders(path8,bs=64)
df8.show_batch()
learn8 = cnn_learner(df8, resnet34,  metrics = accuracy,cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res34RG0075.csv")])
learn8.fine_tune(50,cbs=[ShowGraphCallback()])
interp8 = ClassificationInterpretation.from_learner(learn8)
interp8.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.075 res 18
path8 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs8 = [GaussianNoise(mean=0., std =0.075)]
data8 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs8
        )
df8 = data8.dataloaders(path8,bs=64)
df8.show_batch()
learn8 = cnn_learner(df8, resnet18,  metrics = accuracy,cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res18RG0075.csv")])
learn8.fine_tune(50,cbs=[ShowGraphCallback()])
interp8 = ClassificationInterpretation.from_learner(learn8)
interp8.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.1 res 34
path9 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs9 = [GaussianNoise(mean=0., std =0.1)]
data9 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
```

```
        batch_tfms = tmfs9
        )
df9 = data9.dataloaders(path9,bs=64)
df9.show_batch()
learn9 = cnn_learner(df9, resnet34,  metrics = accuracy,cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res34RG01.csv")])
learn9.fine_tune(50,cbs=[ShowGraphCallback()])
interp9 = ClassificationInterpretation.from_learner(learn9)
interp9.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 0.1 res 18
path9 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs9 = [GaussianNoise(mean=0., std =0.1)]
data9 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs9
        )
df9 = data9.dataloaders(path9,bs=64)
df9.show_batch()
learn9 = cnn_learner(df9, resnet18,  metrics = accuracy,cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res18RG01.csv")])
learn9.fine_tune(50,cbs=[ShowGraphCallback()])
interp9 = ClassificationInterpretation.from_learner(learn9)
interp9.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 1 res 34
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs10 = [GaussianNoise(mean=0., std = 1)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
learn10 = cnn_learner(df10, resnet34,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="/content/gdrive/MyDrive/csv2/Res34RG1.csv")])
learn10.fine_tune(50,cbs=[ShowGraphCallback()])
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 1 Ruido gaussiano mean = 0 std = 1 res 18
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs10 = [GaussianNoise(mean=0., std = 1)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
learn10 = cnn_learner(df10, resnet34,  metrics =␣
 ↪accuracy,cbs=[CSVLogger(fname="/content/gdrive/MyDrive/csv2/Res18RG1.csv")])
learn10.fine_tune(50,cbs=[ShowGraphCallback()])
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento label smoothing res 34
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
#tmfs10 = [GaussianNoise(mean=0., std = 1)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label
        #batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
learn10 = cnn_learner(df10, resnet34,  metrics =␣
 ↪accuracy,loss_func=LabelSmoothingCrossEntropy(),cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res34LS.csv")])
learn10.fine_tune(50,cbs=[ShowGraphCallback()])
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento label smoothing res 18
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
#tmfs10 = [GaussianNoise(mean=0., std = 1)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
```

```
        get_y = parent_label
        #batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
learn10 = cnn_learner(df10, resnet18,  metrics =␣
 ↪accuracy,loss_func=LabelSmoothingCrossEntropy(),cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res18LS.csv")])
learn10.fine_tune(50,cbs=[ShowGraphCallback()])
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 3 tipos tfms res 34
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs10 = [Dihedral(p=1),GaussianNoise(mean=0., std = 0.075)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
learn10 = cnn_learner(df10, resnet34,  metrics =␣
 ↪accuracy,loss_func=LabelSmoothingCrossEntropy(),cbs=[CSVLogger(fname="/
 ↪content/gdrive/MyDrive/csv2/Res34Triple.csv")])
learn10.fine_tune(50)
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Preparacion  experimento 3 tipos tfms res 18
path10 = Path('/content/gdrive/MyDrive/Clasificacion/rgb_images_200')
tmfs10 = [Dihedral(p=1),GaussianNoise(mean=0., std = 0.025)]
data10 = DataBlock(
        blocks = (ImageBlock,CategoryBlock),
        get_items = get_image_files,
        #splitter = GrandparentSplitter(valid_name = "val"),
        splitter = RandomSplitter(0.2),
        get_y = parent_label,
        batch_tfms = tmfs10
        )
df10 = data10.dataloaders(path10,bs=64)
df10.show_batch()
```

```
learn10 = cnn_learner(df10, resnet18,  metrics =␣
↪accuracy,loss_func=LabelSmoothingCrossEntropy(),cbs=[CSVLogger(fname="/
↪content/gdrive/MyDrive/csv2/Res18Triple.csv")])
learn10.fine_tune(50)
interp10 = ClassificationInterpretation.from_learner(learn10)
interp10.plot_confusion_matrix()
```

[ ]:
```
#Creacion de graficas
#Lectura de datos asociadasa los experimentos con giros diedricos

dataBase18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18Base30.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])
dataBase34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34Base30.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])

data_D_025_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18D025.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_D_025_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34D025.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])

data_D_05_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18D05.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_D_05_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34D05.csv",␣
↪delimiter=",",
                ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
↪names=["epoch", "train_loss","valid_loss","accuracy"])

data_D_075_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18D075.csv",␣
↪delimiter=",",
```

```
                ␣
  ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
  ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_D_075_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34D075.csv",␣
  ↪delimiter=",",
                ␣
  ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
  ↪names=["epoch", "train_loss","valid_loss","accuracy"])

data_D_1_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18D1.csv",␣
  ↪delimiter=",",
                ␣
  ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
  ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_D_1_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34D1.csv",␣
  ↪delimiter=",",
                ␣
  ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
  ↪names=["epoch", "train_loss","valid_loss","accuracy"])

mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['train_loss'],
                 dataBase18['epoch'],data_D_025_18['train_loss'],
                 dataBase18['epoch'],data_D_05_18['train_loss'],
                 dataBase18['epoch'],data_D_075_18['train_loss'],
                 dataBase18['epoch'],data_D_1_18['train_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_18_train_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['train_loss'],
                 dataBase34['epoch'],data_D_025_34['train_loss'],
                 dataBase34['epoch'],data_D_05_34['train_loss'],
                 dataBase34['epoch'],data_D_075_34['train_loss'],
                 dataBase34['epoch'],data_D_1_34['train_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
```

```
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_34_train_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['valid_loss'],
                 dataBase18['epoch'],data_D_025_18['valid_loss'],
                 dataBase18['epoch'],data_D_05_18['valid_loss'],
                 dataBase18['epoch'],data_D_075_18['valid_loss'],
                 dataBase18['epoch'],data_D_1_18['valid_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_18_valid_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['valid_loss'],
                 dataBase34['epoch'],data_D_025_34['valid_loss'],
                 dataBase34['epoch'],data_D_05_34['valid_loss'],
                 dataBase34['epoch'],data_D_075_34['valid_loss'],
                 dataBase34['epoch'],data_D_1_34['valid_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_34_valid_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['accuracy'],
                 dataBase34['epoch'],data_D_025_34['accuracy'],
                 dataBase34['epoch'],data_D_05_34['accuracy'],
                 dataBase34['epoch'],data_D_075_34['accuracy'],
                 dataBase34['epoch'],data_D_1_34['accuracy'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_34_accuracy.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['accuracy'],
                 dataBase18['epoch'],data_D_025_18['accuracy'],
                 dataBase18['epoch'],data_D_05_18['accuracy'],
                 dataBase18['epoch'],data_D_075_18['accuracy'],
                 dataBase18['epoch'],data_D_1_18['accuracy'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "p = 0.25")
plt.setp(l3, color='g', linewidth=2.0, label = "p = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "p = 0.75")
plt.setp(l5, color='y', linewidth=2.0, label = "p = 1")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/D_18_accuracy.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
# #Lectura de datos asociadasa los experimentos con ruido gaussiano
data_RG_0025_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18RG0025.csv",␣
 ↪delimiter=",",
                                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_0025_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34RG0025.csv",␣
 ↪delimiter=",",
                                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
```

```
data_RG_005_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18RG005.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_005_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34RG005.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])

data_RG_0075_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18RG0075.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_0075_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34RG0075.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_01_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18RG01.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_01_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34RG01.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_1_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18RG1.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_RG_1_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34RG1.csv",␣
 ↪delimiter=",",
                ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])

mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['accuracy'],
                 data_RG_0025_34['epoch'],data_RG_0025_34['accuracy'],
                 data_RG_005_34['epoch'],data_RG_005_34['accuracy'],
```

```
                data_RG_0075_34['epoch'],data_RG_0075_34['accuracy'],
                data_RG_01_34['epoch'],data_RG_01_34['accuracy'],
                data_RG_1_34['epoch'],data_RG_1_34['accuracy'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_34_accuracy.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['accuracy'],
                data_RG_0025_18['epoch'],data_RG_0025_18['accuracy'],
                data_RG_005_18['epoch'],data_RG_005_18['accuracy'],
                data_RG_0075_18['epoch'],data_RG_0075_18['accuracy'],
                data_RG_01_18['epoch'],data_RG_01_18['accuracy'],
                data_RG_1_18['epoch'],data_RG_1_18['accuracy'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_18_accuracy.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['valid_loss'],
                dataBase34['epoch'],data_RG_0025_34['valid_loss'],
                dataBase34['epoch'],data_RG_005_34['valid_loss'],
                dataBase34['epoch'],data_RG_0075_34['valid_loss'],
                dataBase34['epoch'],data_RG_01_34['valid_loss'],
                dataBase34['epoch'],data_RG_1_34['valid_loss'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
```

```
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_34_valid_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['valid_loss'],
                 dataBase18['epoch'],data_RG_0025_18['valid_loss'],
                 dataBase18['epoch'],data_RG_005_18['valid_loss'],
                 dataBase18['epoch'],data_RG_0075_18['valid_loss'],
                 dataBase18['epoch'],data_RG_01_18['valid_loss'],
                 dataBase18['epoch'],data_RG_1_18['valid_loss'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_18_valid_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['train_loss'],
                 dataBase34['epoch'],data_RG_0025_34['train_loss'],
                 dataBase34['epoch'],data_RG_005_34['train_loss'],
                 dataBase34['epoch'],data_RG_0075_34['train_loss'],
                 dataBase34['epoch'],data_RG_01_34['train_loss'],
                 dataBase34['epoch'],data_RG_1_34['train_loss'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
```

```
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_34_train_loss.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['train_loss'],
                 dataBase18['epoch'],data_RG_0025_18['train_loss'],
                 dataBase18['epoch'],data_RG_005_18['train_loss'],
                 dataBase18['epoch'],data_RG_0075_18['train_loss'],
                 dataBase18['epoch'],data_RG_01_18['train_loss'],
                 dataBase18['epoch'],data_RG_1_18['train_loss'])
l1,l2,l3,l4,l5,l6 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.075")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.1")
plt.setp(l6, color='m', linewidth=2.0, label = "std = 1")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_18_train_loss.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
#Lectura de datos asociadasa los experimentos con label smoothing
data_LS_18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18LS.csv",␣
 ↪delimiter=",",
                           ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data_LS_34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34LS.csv",␣
 ↪delimiter=",",
                           ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['valid_loss'],
                 dataBase34['epoch'],data_LS_34['valid_loss'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_valid_34.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['valid_loss'],
                 dataBase18['epoch'],data_LS_18['valid_loss'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_valid_18.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['train_loss'],
                 dataBase34['epoch'],data_LS_34['train_loss'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_train_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['train_loss'],
                 dataBase18['epoch'],data_LS_18['train_loss'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_train_18.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['accuracy'],
                 dataBase34['epoch'],data_LS_34['accuracy'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_accuracy_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['accuracy'],
                 dataBase18['epoch'],data_LS_18['accuracy'])
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "label smothing")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/LS_accuracy_18.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
#Lectura de datos asociadasa los experimentos finales
data30Base18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18Base30.csv",␣
 ↪delimiter=",",
                             ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
dataTriple18 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res18Triple.csv",␣
 ↪delimiter=",",
                             ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
data30Base34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34Base30.csv",␣
 ↪delimiter=",",
                             ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
dataTriple34 = np.genfromtxt("/content/gdrive/MyDrive/csv2/Res34Triple.csv",␣
 ↪delimiter=",",
                             ␣
 ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('accuracy','f8')],␣
 ↪names=["epoch", "train_loss","valid_loss","accuracy"])
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base18['epoch'],data30Base18['accuracy'],
                 data30Base18['epoch'],dataTriple18['accuracy']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_accuracy_18.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base34['epoch'],data30Base34['accuracy'],
                 data30Base34['epoch'],dataTriple34['accuracy']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_accuracy_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base18['epoch'],data30Base18['train_loss'],
                 data30Base18['epoch'],dataTriple18['train_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_train_loss_18.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base34['epoch'],data30Base34['train_loss'],
                 data30Base34['epoch'],dataTriple34['train_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_train_loss_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base18['epoch'],data30Base18['valid_loss'],
                 data30Base18['epoch'],dataTriple18['valid_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('valid_loss')
```

```
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_valid_loss_18.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data30Base34['epoch'],data30Base34['valid_loss'],
                 data30Base34['epoch'],dataTriple34['valid_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "RG + GD + label smothing")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Triple_valid_loss_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['train_loss'],
                 dataBase34['epoch'],dataBase34['valid_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "train_loss")
plt.setp(l2, color='b', linewidth=2.0, label = "valid_loss")
plt.ylabel('Loss-value')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/base_34.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['train_loss'],
                 dataBase18['epoch'],dataBase18['valid_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "train_loss")
plt.setp(l2, color='b', linewidth=2.0, label = "valid_loss")
plt.ylabel('Loss-value')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/base_18.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['accuracy'],
                 dataBase18['epoch'],dataBase18['accuracy']
```

```
                    )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "Resnet34")
plt.setp(l2, color='b', linewidth=2.0, label = "Resnet18")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/accuracy_3418.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],data_RG_0025_34['train_loss'],
                 dataBase18['epoch'],data_RG_1_18['train_loss'],
                 dataBase34['epoch'],data_RG_0025_34['valid_loss'],
                 dataBase18['epoch'],data_RG_1_18['valid_loss']
                 )
l1,l2,l3,l4 = lines
plt.setp(l1, color='b', linewidth=2.0,linestyle = '-', label = "Resnet34␣
 ↪train_loss")
plt.setp(l2, color='r', linewidth=2.0,linestyle = '-', label = "Resnet18␣
 ↪train_loss")
plt.setp(l3, color='b', linewidth=2.0,linestyle = '--', label = "Resnet34␣
 ↪valid_loss")
plt.setp(l4, color='r', linewidth=2.0,linestyle = '--', label = "Resnet18␣
 ↪valid_loss")
plt.ylabel('loss_function')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/RG_final_loss.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],data_RG_0025_34['accuracy'],
                 dataBase18['epoch'],data_RG_1_18['accuracy']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "Resnet34")
plt.setp(l2, color='b', linewidth=2.0, label = "Resnet18")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/accuracy_RG_final.
 ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
from google.colab import files
```

```
!zip -r /content/ClasificacionGraficas.zip /content/gdrive/MyDrive/
  ↪ClasificacionGraficas
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['accuracy'],
                 dataBase18['epoch'],data_D_1_18['accuracy'],
                 dataBase18['epoch'],data_RG_1_18['accuracy'],
                 dataBase18['epoch'],data_LS_18['accuracy'],
                 dataBase18['epoch'],dataTriple18['accuracy'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "GD: p = 1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG: std = 1")
plt.setp(l4, color='c', linewidth=2.0, label = "LS")
plt.setp(l5, color='y', linewidth=2.0, label = "GD + LS + RG")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Merge_18_accuracy.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['train_loss'],
                 dataBase18['epoch'],data_D_1_18['train_loss'],
                 dataBase18['epoch'],data_RG_1_18['train_loss'],
                 dataBase18['epoch'],data_LS_18['train_loss'],
                 dataBase18['epoch'],dataTriple18['train_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "D: p = 1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG: std = 0.5")
plt.setp(l4, color='c', linewidth=2.0, label = "LS")
plt.setp(l5, color='y', linewidth=2.0, label = "D + LS + RG")
plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Merge_18_train_loss.
  ↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase18['epoch'],dataBase18['valid_loss'],
                 dataBase18['epoch'],data_D_1_18['valid_loss'],
                 dataBase18['epoch'],data_RG_1_18['valid_loss'],
                 dataBase18['epoch'],data_LS_18['valid_loss'],
                 dataBase18['epoch'],dataTriple18['valid_loss'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
```

```
plt.setp(l2, color='b', linewidth=2.0, label = "D: p = 1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG: std = 1")
plt.setp(l4, color='c', linewidth=2.0, label = "LS")
plt.setp(l5, color='y', linewidth=2.0, label = "GD + LS + RG")
plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Merge_18_valid_loss.
↪png",dpi=300, bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase34['epoch'],dataBase34['accuracy'],
                 dataBase34['epoch'],data_D_1_34['accuracy'],
                 dataBase34['epoch'],data_RG_0025_34['accuracy'],
                 dataBase34['epoch'],data_LS_34['accuracy'],
                 dataBase34['epoch'],dataTriple34['accuracy'])
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "GD: p = 1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG: std = 0.025")
plt.setp(l4, color='c', linewidth=2.0, label = "LS")
plt.setp(l5, color='y', linewidth=2.0, label = "GD + LS + RG")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/gdrive/MyDrive/ClasificacionGraficas/Merge_34_accuracy.
↪png",dpi=300, bbox_inches = "tight")
```

[16]:
```
%cd /content/gdrive/MyDrive/Colab\ Notebooks
```

[17]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-generic-recommended
```

[18]:
```
!jupyter nbconvert --to pdf --TemplateExporter.exclude_output=True ␣
↪CodigoExperimentoClasificacion.ipynb
```

CodigoExperimentoIdentificacionContorno

March 29, 2022

[ ]:
```
#hide
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

[ ]:
```
#hide
from fastbook import *
from fastai.vision.all import *
from matplotlib.patches import Ellipse
import numpy.random as rnd
```

[ ]:
```
df = pd.read_csv('/content/drive/MyDrive/Regression/benja_ppales_params_low_v4.
 ↪csv')
df.head()
```

[ ]:
```
from sklearn.metrics import mean_squared_error,r2_score

#Funcion que representa la ellipse inicial con la elipse predicha en una misma␣
 ↪imagen junto a metricas

def show_results_elipse(learn,n):
  img =  learn.dls.valid_ds[n][0]

  #angle_pred =learn.get_preds(1)[0][n].numpy()[1]
  ##semieje_pred = learn.get_preds(1)[0][n].numpy()[2]
  factor_pred = learn.get_preds(1)[0][n].numpy()[0]
  #x_pred = learn.get_preds(1)[0][n].numpy()[3]
  #y_pred = learn.get_preds(1)[0][n].numpy()[4]
  #ells_pred = Ellipse(xy=[x_pred,y_pred], width=2*semieje_pred*factor_pred,␣
 ↪height=2*semieje_pred, angle= -angle_pred␣
 ↪,edgecolor='green',facecolor='none',linewidth=2)

  angle_orig = learn.get_preds(1)[1][n].numpy()[1]
  semieje_orig = learn.get_preds(1)[1][n].numpy()[2]
  factor_orig = learn.get_preds(1)[1][n].numpy()[0]
  x_orig = learn.get_preds(1)[1][n].numpy()[3]
  y_orig = learn.get_preds(1)[1][n].numpy()[4]
```

```
  ells_orig = Ellipse(xy=[x_orig,y_orig], width=2*semieje_orig*factor_orig,␣
↪height=2*semieje_orig, angle= -angle_orig␣
↪,edgecolor='red',facecolor='none',linewidth=1)

  fig,ax = plt.subplots(1)
  ax.add_patch(ells_orig)
  ax.add_patch(ells_pred)
  ax.imshow(img)
  print("Original: ",learn.get_preds(1)[1][n].tolist())
  print("Prediccion: " ,learn.get_preds(1)[0][n].tolist())
  plt.show()
  print("R2: " ,r2_score(learn.get_preds(1)[1][n].tolist(), learn.
↪get_preds(1)[0][n].tolist()))
  print("MSE: " ,mean_squared_error(learn.get_preds(1)[1][n].tolist(), learn.
↪get_preds(1)[0][n].tolist()))
```

[ ]:
```
#Funcion que representa la mejor y la peor clasificafion con la ellipse inicial␣
↪y la elipse predicha en una misma imagen junto a metricas
def show_best_worst_results_elipse(learn):
  index_max = 0
  r2_max = 0
  index_min = 0
  r2_min = 100
  for n in range(0,len(learn.get_preds(1)[0])):

    angle_pred =learn.get_preds(1)[0][n].numpy()[1]
    semieje_pred = learn.get_preds(1)[0][n].numpy()[2]
    factor_pred = learn.get_preds(1)[0][n].numpy()[0]
    x_pred = learn.get_preds(1)[0][n].numpy()[3]
    y_pred = learn.get_preds(1)[0][n].numpy()[4]

    angle_orig = learn.get_preds(1)[1][n].numpy()[1]
    semieje_orig = learn.get_preds(1)[1][n].numpy()[2]
    factor_orig = learn.get_preds(1)[1][n].numpy()[0]
    x_orig = learn.get_preds(1)[1][n].numpy()[3]
    y_orig = learn.get_preds(1)[1][n].numpy()[4]

    r2 = r2_score(learn.get_preds(1)[1][n].tolist(), learn.get_preds(1)[0][n].
↪tolist())
    if r2 < r2_min:
      index_min = n
      r2_min = r2

    if r2 > r2_max:
      index_max = n
      r2_max = r2
    print("ITERACION",n )
```

```
  print("MEJOR RESULTADO")
  show_results_elipse(learn,index_max)
  print("PEOR RESULTADO")
  show_results_elipse(learn,index_min)
```

[ ]:
```
data_raw = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
 ↪rgb_images_regresion/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = Normalize.from_stats(*imagenet_stats)
)
dloader_raw = data_raw.dataloaders(df)
```

[ ]:
```
data_mask = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
 ↪rgb_images_regresion_mask/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = Normalize.from_stats(*imagenet_stats)
)
dloader_mask = data_mask.dataloaders(df)
```

[ ]:
```
#Experimento base
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw = cnn_learner(dloader_raw, resnet34,␣
 ↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="BaseNoMas
 ↪csv")])
learn_raw.fine_tune(120,cbs=[ShowGraphCallback()])
```

[ ]:
```
show_results_elipse(learn_raw,1)
show_results_elipse(learn_raw,2)
show_results_elipse(learn_raw,3)
show_results_elipse(learn_raw,4)
show_results_elipse(learn_raw,5)
```

[ ]:
```
show_best_worst_results_elipse(learn_raw)
```

[ ]:
```
class GaussianNoise(RandTransform):

    def __init__(self, mean=0., std=1., **kwargs):
        self.std = std
        self.mean = mean
```

```
        super().__init__(**kwargs)

    def encodes(self, x:TensorImage):
        return x + torch.randn(x.size()).cuda() * self.std + self.mean
```

[ ]:
```
#Experimentos con data augmentation  utilizando ruido gausiano
tmfs1 = [Normalize.from_stats(*imagenet_stats),GaussianNoise(mean=0., std =0.
↪025)]
data_raw_noise1 = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
↪rgb_images_regresion/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = tmfs1
)
dloader_raw_noise1 = data_raw_noise1.dataloaders(df)
```

[ ]:
```
tmfs2 = [Normalize.from_stats(*imagenet_stats),GaussianNoise(mean=0., std =0.
↪05)]
data_raw_noise2 = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
↪rgb_images_regresion/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = tmfs2
)
dloader_raw_noise2 = data_raw_noise2.dataloaders(df)
```

[ ]:
```
tmfs3 = [Normalize.from_stats(*imagenet_stats),GaussianNoise(mean=0., std =0.1)]
data_raw_noise3 = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
↪rgb_images_regresion/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = tmfs3
)
dloader_raw_noise3 = data_raw_noise3.dataloaders(df)
```

[ ]:
```
tmfs4 = [Normalize.from_stats(*imagenet_stats),GaussianNoise(mean=0., std =0.
↪25)]
data_raw_noise4 = DataBlock(
        blocks = (ImageBlock,RegressionBlock()),
```

```
        get_x = ColReader('gal_id', pref='/content/drive/MyDrive/Regression/
 ↪rgb_images_regresion/', suff='_rgb_image.png'),
        get_y = ColReader(['q','pa','r_edge','xcH','ycH']),
        splitter = RandomSplitter(0.2),
        batch_tfms = tmfs4
)
dloader_raw_noise4 = data_raw_noise4.dataloaders(df)
```

[ ]:
```
# experimento std 0.025
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw_noise1 = cnn_learner(dloader_raw_noise1, resnet34,␣
 ↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRG0025
 ↪csv")])
learn_raw_noise1.fine_tune(120,cbs=[ShowGraphCallback()])
```

[ ]:
```
show_results_elipse(learn_raw_noise1,1)
show_results_elipse(learn_raw_noise1,2)
show_results_elipse(learn_raw_noise1,3)
show_results_elipse(learn_raw_noise1,4)
show_results_elipse(learn_raw_noise1,5)
```

[ ]:
```
show_best_worst_results_elipse(learn_raw_noise1)
```

[ ]:
```
# experimento std 0.05
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw_noise2 = cnn_learner(dloader_raw_noise2, resnet34,␣
 ↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRG005.
 ↪csv")])
learn_raw_noise2.fine_tune(120,cbs=[ShowGraphCallback()])
```

[ ]:
```
show_results_elipse(learn_raw_noise2,1)
show_results_elipse(learn_raw_noise2,2)
show_results_elipse(learn_raw_noise2,3)
show_results_elipse(learn_raw_noise2,4)
show_results_elipse(learn_raw_noise2,5)
```

[ ]:
```
show_best_worst_results_elipse(learn_raw_noise2)
```

[ ]:
```
# experimento std 0.1
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw_noise3 = cnn_learner(dloader_raw_noise3, resnet34,␣
 ↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRG01.
 ↪csv")])
```

```
learn_raw_noise3.fine_tune(120,cbs=[ShowGraphCallback()])
```

[ ]:
```
show_results_elipse(learn_raw_noise3,1)
show_results_elipse(learn_raw_noise3,2)
show_results_elipse(learn_raw_noise3,3)
show_results_elipse(learn_raw_noise3,4)
show_results_elipse(learn_raw_noise3,5)
```

[ ]:
```
show_best_worst_results_elipse(learn_raw_noise3)
```

[ ]:
```
# experimento std 0.25
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw_noise4 = cnn_learner(dloader_raw_noise4, resnet34,␣
↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRG025.
↪csv")])
learn_raw_noise4.fine_tune(120,cbs=[ShowGraphCallback()])
```

[ ]:
```
show_results_elipse(learn_raw_noise4,3)
show_results_elipse(learn_raw_noise4,2)
show_results_elipse(learn_raw_noise4,3)
show_results_elipse(learn_raw_noise4,4)
show_results_elipse(learn_raw_noise4,5)
```

[ ]:
```
show_best_worst_results_elipse(learn_raw_noise4)
```

[ ]:
```
explainedVariance = ExplainedVariance()
r2score = R2Score()
learn_raw_noise_lr = cnn_learner(dloader_raw_noise3, resnet34,␣
↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRGLR1.
↪csv")])
lr = learn_raw_noise_lr.lr_find(end_lr=1000,num_it=1000,suggest_funcs=(minimum,␣
↪steep, valley, slide))
```

[ ]:
```
learn_raw_noise_lr = cnn_learner(dloader_raw_noise3, resnet34,␣
↪loss_func=MSELossFlat(),metrics=[explainedVariance,r2score],cbs=[CSVLogger(fname="RegRGLR2.
↪csv")])
```

[ ]:
```
learn_raw_noise_lr.fit_one_cycle(30,0.
↪008511380292475224,cbs=[ShowGraphCallback(),CSVLogger(fname="RegRGLR3.csv")])
```

[ ]:
```
learn_raw_noise_lr.unfreeze()
lr2 = learn_raw_noise_lr.
↪lr_find(end_lr=1000,num_it=1000,suggest_funcs=(minimum, steep, valley,␣
↪slide))
```

[ ]:
```
learn_raw_noise_lr.fit_one_cycle(90,␣
↪lr_max=slice(1e-6,1e-4),cbs=[ShowGraphCallback(),CSVLogger(fname="RegRGLR4.
↪csv")])
```

[ ]:
```
len(learn_raw_noise_lr.get_preds(1)[0])
```

[ ]:
```
show_best_worst_results_elipse(learn_raw_noise_lr)
```

[ ]:
```
# Lectura de resultados para creacion de graficas de experimentos con ruido␣
↪gaussiano
dataBase = np.genfromtxt("/content/drive/MyDrive/CSV/BaseNoMask.csv",␣
↪delimiter=",",
                         ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
↪names=["epoch", "train_loss","valid_loss","r2_score"])
data_RG_0025 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRG0025.csv",␣
↪delimiter=",",
                         ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
↪names=["epoch", "train_loss","valid_loss","r2_score"])
data_RG_005 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRG005.csv",␣
↪delimiter=",",
                         ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
↪names=["epoch", "train_loss","valid_loss","r2_score"])
data_RG_01 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRG01.csv",␣
↪delimiter=",",
                         ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
↪names=["epoch", "train_loss","valid_loss","r2_score"])
data_RG_025 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRG025.csv",␣
↪delimiter=",",
                         ␣
↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
↪names=["epoch", "train_loss","valid_loss","r2_score"])

mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['r2_score'],
                 data_RG_0025['epoch'],data_RG_0025['r2_score'],
                 data_RG_005['epoch'],data_RG_005['r2_score'],
                 data_RG_01['epoch'],data_RG_01['r2_score'],
                 data_RG_025['epoch'],data_RG_025['r2_score']
                 )
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
```

```
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.1")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.25")

plt.annotate('%0.4f' % dataBase['r2_score'][-1], xy=(1,␣
 ↪dataBase['r2_score'][-1]), xytext=(8, -42),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.4f' % data_RG_0025['r2_score'][-1], xy=(1,␣
 ↪data_RG_0025['r2_score'][-1]), xytext=(8, -56),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.4f' % data_RG_005['r2_score'][-1], xy=(1,␣
 ↪data_RG_005['r2_score'][-1]), xytext=(8, -28),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')
plt.annotate('%0.4f' % data_RG_01['r2_score'][-1], xy=(1,␣
 ↪data_RG_01['r2_score'][-1]), xytext=(8, 0),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='c')
plt.annotate('%0.4f' % data_RG_025['r2_score'][-1], xy=(1,␣
 ↪data_RG_025['r2_score'][-1]), xytext=(8, -14),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='y')

plt.ylabel('r2_score')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 0.25), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/RG_R2.png",dpi=300, bbox_inches =␣
 ↪"tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['valid_loss'],
                data_RG_0025['epoch'],data_RG_0025['valid_loss'],
                data_RG_005['epoch'],data_RG_005['valid_loss'],
                data_RG_01['epoch'],data_RG_01['valid_loss'],
                data_RG_025['epoch'],data_RG_025['valid_loss']
                )
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.1")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.25")
```

```
plt.annotate('%0.2f' % dataBase['valid_loss'][-1], xy=(1,␣
 ↪dataBase['valid_loss'][-1]), xytext=(8, 42),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.2f' % data_RG_0025['valid_loss'][-1], xy=(1,␣
 ↪data_RG_0025['valid_loss'][-1]), xytext=(8, 56),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.2f' % data_RG_005['valid_loss'][-1], xy=(1,␣
 ↪data_RG_005['valid_loss'][-1]), xytext=(8, 28),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')
plt.annotate('%0.2f' % data_RG_01['valid_loss'][-1], xy=(1,␣
 ↪data_RG_01['valid_loss'][-1]), xytext=(8, 14),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='c')
plt.annotate('%0.2f' % data_RG_025['valid_loss'][-1], xy=(1,␣
 ↪data_RG_025['valid_loss'][-1]), xytext=(8, 0),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='y')

plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/RG_Valid.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['train_loss'],
                 data_RG_0025['epoch'],data_RG_0025['train_loss'],
                 data_RG_005['epoch'],data_RG_005['train_loss'],
                 data_RG_01['epoch'],data_RG_01['train_loss'],
                 data_RG_025['epoch'],data_RG_025['train_loss']
                 )
l1,l2,l3,l4,l5 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.025")
plt.setp(l3, color='g', linewidth=2.0, label = "std = 0.05")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.1")
plt.setp(l5, color='y', linewidth=2.0, label = "std = 0.25")

plt.annotate('%0.2f' % dataBase['train_loss'][-1], xy=(1,␣
 ↪dataBase['train_loss'][-1]), xytext=(8, 56),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
```

```
plt.annotate('%0.2f' % data_RG_0025['train_loss'][-1], xy=(1,␣
 ↪data_RG_0025['train_loss'][-1]), xytext=(8, 0),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.2f' % data_RG_005['train_loss'][-1], xy=(1,␣
 ↪data_RG_005['train_loss'][-1]), xytext=(8, 14),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')
plt.annotate('%0.2f' % data_RG_01['train_loss'][-1], xy=(1,␣
 ↪data_RG_01['train_loss'][-1]), xytext=(8, 42),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='c')
plt.annotate('%0.2f' % data_RG_025['train_loss'][-1], xy=(1,␣
 ↪data_RG_025['train_loss'][-1]), xytext=(8, 28),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='y')

plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/RG_Train.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['train_loss'],
                 dataBase['epoch'],dataBase['valid_loss']
                 )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "train_loss")
plt.setp(l2, color='b', linewidth=2.0, label = "valid_loss")
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Base_Loss.png",dpi=300,␣
 ↪bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['r2_score']
                 )
l1 = lines
plt.setp(l1, color='r', linewidth=2.0)
plt.ylabel('r2_score')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Base_R2.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

```
[ ]: #Lectura de resultados para creacion de graficas de experimentos con learning␣
      ↪rate modificado
     data_LR1 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRGLR3.csv",␣
      ↪delimiter=",",
                 ␣
      ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
      ↪names=["epoch", "train_loss","valid_loss","r2_score"])
     data_LR2 = np.genfromtxt("/content/drive/MyDrive/CSV/RegRGLR4.csv",␣
      ↪delimiter=",",
                 ␣
      ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
      ↪names=["epoch", "train_loss","valid_loss","r2_score"])

     data_LRMerge = np.genfromtxt("/content/drive/MyDrive/CSV/RegRGLR5.csv",␣
      ↪delimiter=",",
                 ␣
      ↪dtype=[('epoch','i8'),('train_loss','f8'),('valid_loss','f8'),('r2_score','f8')],␣
      ↪names=["epoch", "train_loss","valid_loss","r2_score"])


     mpl.style.use("seaborn")
     lines = plt.plot(data_LR1['epoch'],data_LR1['train_loss'],
                      data_LR1['epoch'],data_LR1['valid_loss']
                      )
     l1,l2 = lines
     plt.setp(l1, color='r', linewidth=2.0, label = "train_loss")
     plt.setp(l2, color='b', linewidth=2.0, label = "valid_loss")
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
     plt.savefig("/content/drive/MyDrive/Graficas/LR1_loss.png",dpi=300, bbox_inches␣
      ↪= "tight")
```

```
[ ]: mpl.style.use("seaborn")
     lines = plt.plot(data_LR1['epoch'],data_LR1['r2_score']
                      )
     l1 = lines
     plt.setp(l1, color='r', linewidth=2.0)
     plt.ylabel('r2_score')
     plt.xlabel('epoch')
     plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
     plt.savefig("/content/drive/MyDrive/Graficas/LR1_R2.png",dpi=300, bbox_inches =␣
      ↪"tight")
```

```
[ ]: mpl.style.use("seaborn")
     lines = plt.plot(data_LR2['epoch'],data_LR2['train_loss'],
```

```
                data_LR2['epoch'],data_LR2['valid_loss']
                )
l1,l2 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "train_loss")
plt.setp(l2, color='b', linewidth=2.0, label = "valid_loss")
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/LR2_loss.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(data_LR2['epoch'],data_LR2['r2_score']
                )
l1 = lines
plt.setp(l1, color='r', linewidth=2.0)
plt.ylabel('r2_score')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/LR2_R2.png",dpi=300, bbox_inches =␣
 ↪"tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['r2_score'],
                data_RG_01['epoch'],data_RG_01['r2_score'],
                dataBase['epoch'],data_LRMerge['r2_score']
                )
l1,l2,l3 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l4, color='c', linewidth=2.0, label = "std = 0.1")
plt.setp(l5, color='b', linewidth=2.0, label = "RG + LR")

plt.annotate('%0.2f' % dataBase['r2_score'][-1], xy=(1,␣
 ↪dataBase['r2_score'][-1]), xytext=(8, -28),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.2f' % data_RG_01['r2_score'][-1], xy=(1,␣
 ↪data_RG_01['r2_score'][-1]), xytext=(8, -14),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='c')
plt.annotate('%0.2f' % data_LRMerge['r2_score'][-1], xy=(1,␣
 ↪data_LRMerge['r2_score'][-1]), xytext=(8, -0),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')

plt.ylabel('r2_score')
```

```
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/RG_Train.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['r2_score'],
                 dataBase['epoch'],data_RG_01['r2_score'],
                 dataBase['epoch'],data_LRMerge['r2_score']
                 )
l1,l2,l3 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG+LR")

plt.annotate('%0.4f' % dataBase['r2_score'][-1], xy=(1,␣
 ↪dataBase['r2_score'][-1]), xytext=(8,-28),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.4f' % data_RG_01['r2_score'][-1], xy=(1,␣
 ↪data_RG_01['r2_score'][-1]), xytext=(8,-14),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.4f' % data_LRMerge['r2_score'][-1], xy=(1,␣
 ↪data_LRMerge['r2_score'][-1]), xytext=(8,0),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')

plt.ylabel('r2_score')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 0.25), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Final_R2.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['r2_score'],
                 dataBase['epoch'],data_RG_01['r2_score'],
                 dataBase['epoch'],data_LRMerge['r2_score']
                 )
l1,l2,l3 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG+LR")

plt.annotate('%0.4f' % dataBase['r2_score'][-1], xy=(1,␣
 ↪dataBase['r2_score'][-1]), xytext=(8,-28),
```

```
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.4f' % data_RG_01['r2_score'][-1], xy=(1,␣
 ↪data_RG_01['r2_score'][-1]), xytext=(8,-14),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.4f' % data_LRMerge['r2_score'][-1], xy=(1,␣
 ↪data_LRMerge['r2_score'][-1]), xytext=(8,0),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')

plt.ylabel('r2_score')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 0.25), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Final_R2.png",dpi=300, bbox_inches␣
 ↪= "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['train_loss'],
                dataBase['epoch'],data_RG_01['train_loss'],
                dataBase['epoch'],data_LRMerge['train_loss']
                )
l1,l2,l3 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG+LR")

plt.annotate('%0.4f' % dataBase['train_loss'][-1], xy=(1,␣
 ↪dataBase['train_loss'][-1]), xytext=(8,28),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.4f' % data_RG_01['train_loss'][-1], xy=(1,␣
 ↪data_RG_01['train_loss'][-1]), xytext=(8,14),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.4f' % data_LRMerge['train_loss'][-1], xy=(1,␣
 ↪data_LRMerge['train_loss'][-1]), xytext=(8,0),
                xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')

plt.ylabel('train_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Final_train.png",dpi=300,␣
 ↪bbox_inches = "tight")
```

[ ]:
```
mpl.style.use("seaborn")
lines = plt.plot(dataBase['epoch'],dataBase['valid_loss'],
                 dataBase['epoch'],data_RG_01['valid_loss'],
                 dataBase['epoch'],data_LRMerge['valid_loss']
                 )
l1,l2,l3 = lines
plt.setp(l1, color='r', linewidth=2.0, label = "base")
plt.setp(l2, color='b', linewidth=2.0, label = "std = 0.1")
plt.setp(l3, color='g', linewidth=2.0, label = "RG+LR")

plt.annotate('%0.4f' % dataBase['valid_loss'][-1], xy=(1,␣
 ↪dataBase['valid_loss'][-1]), xytext=(8,28),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='r')
plt.annotate('%0.4f' % data_RG_01['valid_loss'][-1], xy=(1,␣
 ↪data_RG_01['valid_loss'][-1]), xytext=(8,14),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='b')
plt.annotate('%0.4f' % data_LRMerge['valid_loss'][-1], xy=(1,␣
 ↪data_LRMerge['valid_loss'][-1]), xytext=(8,0),
                 xycoords=('axes fraction', 'data'), textcoords='offset␣
 ↪points',color='g')

plt.ylabel('valid_loss')
plt.xlabel('epoch')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.savefig("/content/drive/MyDrive/Graficas/Final_valid.png",dpi=300,␣
 ↪bbox_inches = "tight")
```

[1]:
```
%cd /content/drive/MyDrive/Colab\ Notebooks
```

[2]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-generic-recommended
```

[ ]:
```
!jupyter nbconvert --to pdf --TemplateExporter.exclude_output=True ␣
 ↪CodigoExperimentoIdentificacionContorno.ipynb
```