Proyecto final modelado matemático: Clasificador del grado de colisión de galaxias basado en redes neuronales profundas.

César Eduardo Pachón C.

Antes de iniciar la descripción del proyeto se plantea una breve introducción a los conceptos fundamentales en referencia a la inteligencia artificial, concretamente al funcionamiento y descripción de las redes neuronales profundas.

Chapter 1

Inteligencia artifical

El término inteligencia artificial (IA) hace referencia al diseño e implementación de algoritmos que cumplen la función de "agente inteligente", i.e., que perciba su entorno y tome medidas que maximicen sus posibilidades de alcanzar sus objetivos con éxito. Una definición menos formal de IA se aplica cuando una máquina o computadora imita funciones "cognitivas" que los humanos asocian con otras mentes humanas, como "aprender" y "resolver problemas" [1]. Dentro de los múltiples subcampos de la inteligencia artificial se encuentra el aprendizaje automático (*Machine Learning*), en donde la máquina es capaz de aprender a partir de la experiencia, no por un proceso de instrucciones programadas con anterioridad. Como se muestra en la Figura 1.1, dentro del paradigma de aprendizaje automático se encuentran un subconjunto de técnicas conocidas como aprendizaje profundo (Deep Learning), cuya su principal característica es que la estructura de estos algoritmos está inspirada en las redes neuronales biológicas de los sistemas nerviosos de los animales.

De manera muy general, una red neuronal artificial (ANN: *Artificial Neural Networks*) es un conjunto de neuronas artificiales (funciones de activación tales como tanh, $sigmoide$, $softmax$, etc.) conectadas entre sí mediante pesos estadísticos w_{ij}, donde la red "aprende" a realizar tareas cuando considera ejemplos, generalmente sin estar programado con ninguna regla de tareas específicas. En el diagrama esquemático de la Figura 1.2, las neuronas se representan por circunferencias, y cada una de éstas se programa una función de activación según sea la función de la capa a la que pertenezca. R, N y S corresponden al número de entradas, neuronas ocultas y salidas, respectivamente; x es el vector de entrada de la red, iw y hw son las matrices de entrada y de peso oculto, respectivamente. hb y ob son los vectores de polarización de las capas ocultas y de salida, respectivamente; ho corresponde al vector de salida de la capa oculta y y al vector de salida de la red.

Las ANN están constituidas por una capa de entrada, una capa de salida

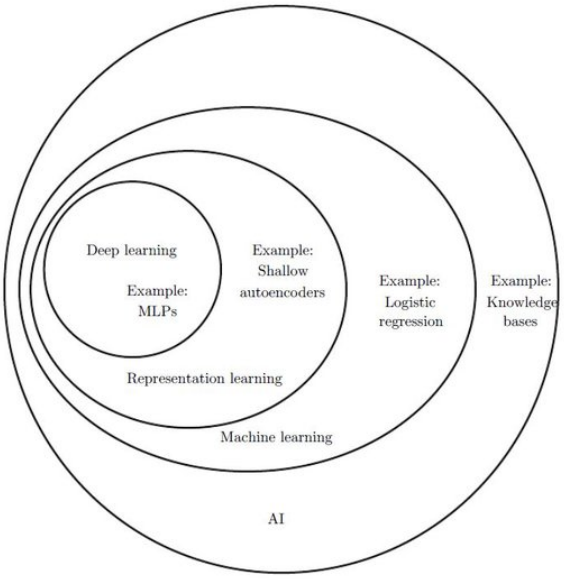


Figure 1.1: Relación entre la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo. Fuente: Ref [2].

y entre estas dos, un cierto número de capas ocultas que realizan diferentes operaciones en los datos. Más de tres capas otorgan a la red el carácter profundo y es de allí de donde viene el concepto de aprendizaje profundo (*Deep Learning*). El proceso de aprendizaje estadístico de la red viene de un proceso de optimización del error en las predicciones de la red y en cada ciclo (*epoc*) del proceso de retroalimentación de la red (*feedback*), los pesos estadísticos w_{ij} de las conexiones entre las neuronas, son recalculados. Este proceso de ajuste sobre los w_{ij} es conocido como propagación hacia atrás (*backpropagation*).

1.0.1 Regresión y clasificación con redes neuronales convolucionales

Para la aplicación del proceso de *backpropagation* es necesario definir el tipo de aprendizaje de la red. Para definir este aspecto principalmente se analizan los datos con los que se entrenará la red. Sí para cada conjunto de entrada x_i se posee el conocimiento a priori y_i, se dice que el paradigma de aprendizaje es supervisado. Dentro del enfoque Bayesiano de la probabilidad, la mayoría de los algoritmos de aprendizaje supervisado se basan en la estimación de la distribución de probabilidad condicional $p(y|x)$ [2]. En este punto el método de estimación por máxima verosimilitud (*maximum likeli-*

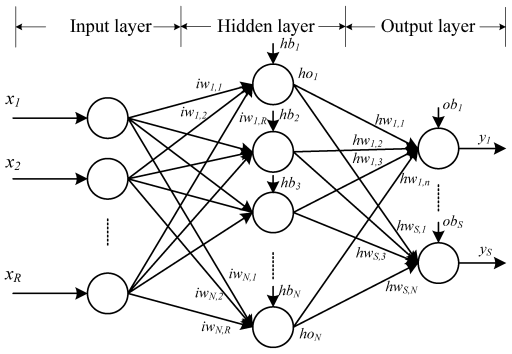


Figure 1.2: Diagrama esquemático de una red neuronal artificial. Fuente: Mdpi.

hood estimation) juega un rol fundamental, pues proporciona un mecanismo para la estimación de los parámetros del modelo estadistico, i.e., permite encontrar el mejor vector de parametros θ para una familia paramétrica de distribuciones $p(y|x;\theta)$. Para los problemas de regresion lineal, la familia de distribuciones de probabilidad condicional esta definida por

$$p(y|x;\theta) = \mathcal{N}(y;\theta^T x, I), \tag{1.1}$$

expresado como una distribución condicional Gaussiana en $\mathbf{y}^{(\text{train})}$,

$$p(\mathbf{y}^{(\text{train})}|\mathbf{X}^{(\text{train})},\theta) = \mathcal{N}(\mathbf{y}^{(\text{train})}, \mathbf{X}^{(\text{train})}\theta, \mathbf{I}) \tag{1.2}$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{y}^{(\text{train})} - \mathbf{X}^{(\text{train})}\theta)^T(\mathbf{y}^{(\text{train})} - \mathbf{X}^{(\text{train})}\theta)\right), \tag{1.3}$$

en donde la etiqueta (train) hace referencia al espacio de muestras que conforman el conocimiento a priori del modelo. Por su parte $\mathbf{X}^{(\text{train})}$ es la información o los datos suministrados para el proceso de aprendizaje, junto con su respectivo conjunto de objetivos $\mathbf{y}^{(\text{train})}$ que el algoritmo buscará producir en un nuevo conjunto de muestras $\mathbf{X}^{(\text{test})}$. El proceso de aprendizaje estadístico se enfoca en computar el vector de parámetros θ que para el caso de las ANN está relacionado con la estimación de los pesos estadísticos w_{ij} que unen a las neuronas de las diferentes capas.

Es posible generalizar el concepto de regresión lineal al escenario de los problemas de clasificación definiendo una familia diferente de distribuciones de probabilidad. Sean dos clases bien definidas, la clase a y la clase b, entonces solo es necesario especificar la probabilidad de una de estas clases. La

probabilidad de la clase b determina la probabilidad de la clase a, puesto que estos dos valores deben cumplir el axioma de probabilidad de Kolmogorov, i.e., la suma de todas las probabilidades debe ser igual a uno.

En los problemas de clasificación es precisamente en donde las redes neuronales convolucionales (CNN: *Convolutional Neural Networks*) poseen grandes capacidades en la ejecución de este tipo de tareas. Las CNN son un conjunto de algoritmos de inteligencia artificial y visión por computadora ampliamente utilizados en la actualidad en el área del procesamiento digital de imágenes. En las que las capas ocultas realizan operaciones de convolución entre diferentes transformaciones de las imágenes de entrada, producto de un proceso de ingeniería de características (*Feature Engineering*).

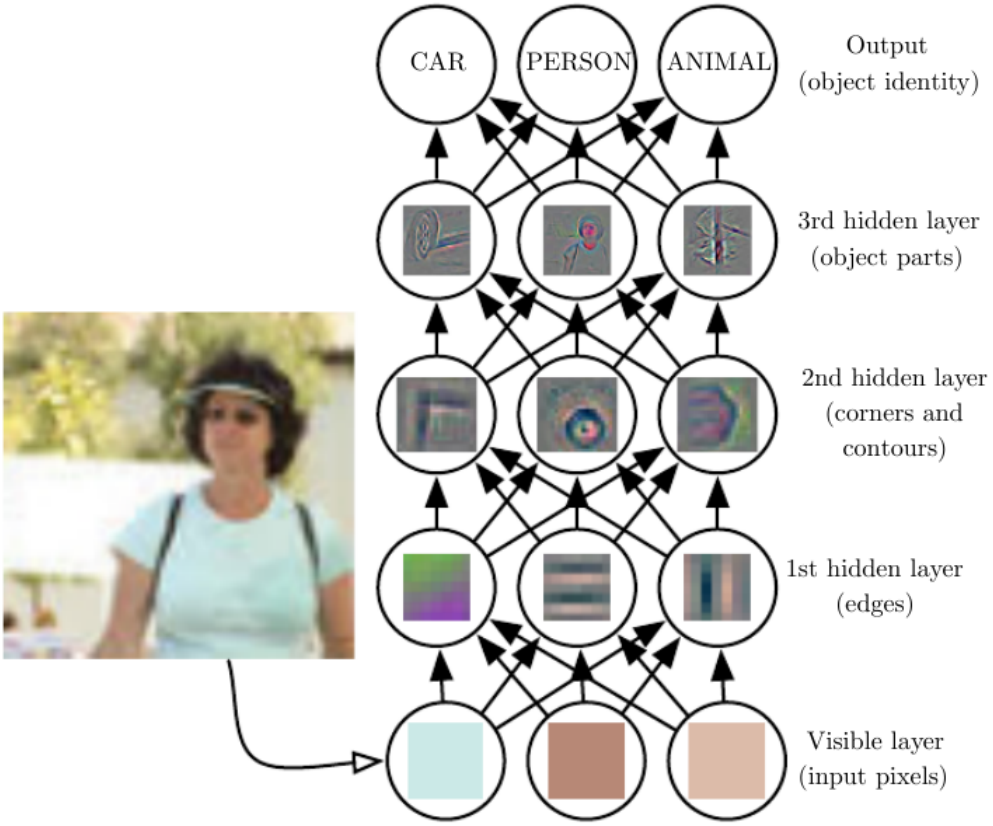


Figure 1.3: Estructura y funcionamiento de una CNN. Fuente: Ref [2].

Este proceso de transformación de características es lo que hace tan robustas a las redes neuronales convolucionales. La Figura 1.3 muestra el funcionamiento en el diagrama esquemático de una CNN, en donde la capa de entrada (**Visual Layer**) recibe la información de intensidad de la imagen en una determinada escala de color. Posteriormente, las siguientes tres capas

ocultas transforman y extraen características cada vez más abstractas de la imagen, a través de la convolución con diferentes núcleos definidos por las funciones de activación de las neuronas. Más específicamente, en el ejemplo citado, la primera capa oculta identifica los bordes en la imagen, comparando la intensidad de los píxeles vecinos. Una vez extraída la característica de los bordes de la primera capa, la segunda capa oculta puede buscar fácilmente esquinas y contornos extendidos, que se reconocen como colecciones de bordes. Ahora, a partir del resultado obtenido por la segunda capa, la tercera capa oculta puede detectar partes enteras de objetos específicos, al encontrar colecciones específicas de contornos y esquinas. Finalmente, la capa de salida asocia las colecciones especificas de contornos y esquinas al conocimiento a priori de los objetivos suministrados a la red en la etapa de entrenamiento, i.e., las diferentes clases que definen el proceso de clasificación, con lo que identifica los objetos presentes en la imagen.

Chapter 2

Planteamiento del problema

Dado un conjunto de imágenes correspondiente a diferentes galaxias, para 2 ángulos: incl_45_phi_26.4 ; incl_135_phi_153.5 y diferentes intervalos de tiempo, se busca entrenar un algoritmo que clasifique estas imágenes según su grado de colisión, este grado de colisión se determina a partir de los intervalos de tiempo, para este trabajo se han definido sólo tres clases basadas en el mapa de velocidades:

- Discos: Grado de colisión cero.

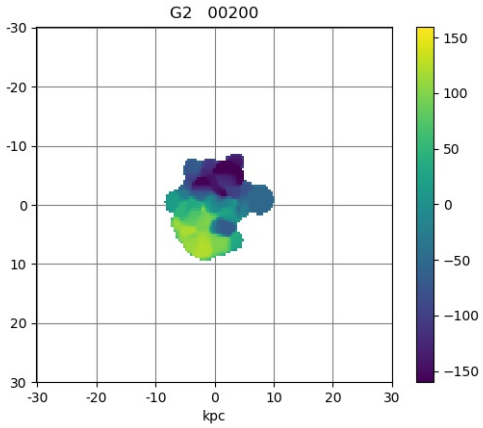
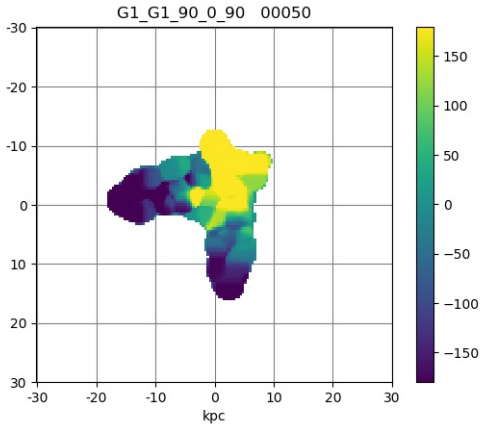


Figure 2.1: Clase Disco

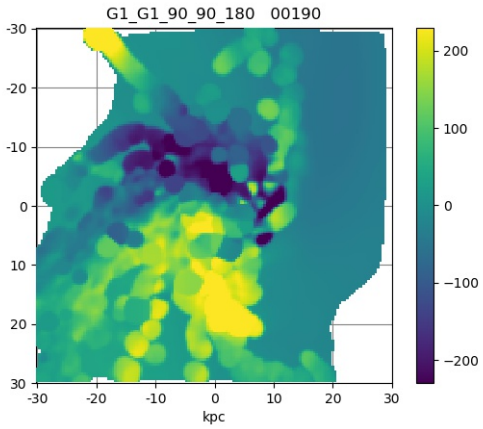
- early_merger: Colisión temprana, aquí inicia la fusión de las galaxias.

Figure 2.2: Clase Early_merger



- late_merger: Fusión de galaxias avanzado.

Figure 2.3: Clase Late_merger



Chapter 3

Objetivos

3.1 Objetivo general

Entrenar una red neuronal profunda que permita clasificar las galaxias segun el grado de colisión definido.

3.1.1 Objetivos especificos

- Estudiar el funcionamiento y la arquitectura de las redes neuronales profundas.
- Aplicar los conceptos estudiados en el curso de modelado matemático uno a la solución del problema.
- Mejorar habilidadees de programación en python.

Chapter 4

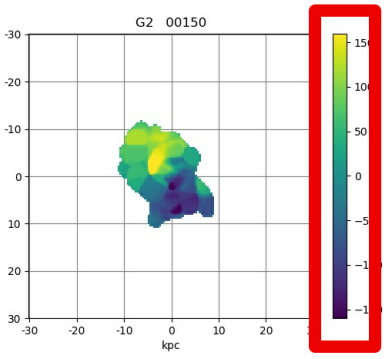
Solución

4.1 El dataset

El dataset esta conformado por tres grupos de imágenes en baja, media y alta resolución, estos a su vez contienen información de mapa de velocidades, mapa de flujo luminoso y dispersión de velocidades. Para esta primera versión del algoritmo sólo se tendrá en cuenta la información de los mapas de velocidades. Para el entrenamiento de la red se han tomado 150 imágenes que incluyen las tres resoluciones y los dos ángulos de inclinación para las tres clases, la validación del algoritmo cuenta con 120 imágenes con las mismas características del set de entrenamiento. El primer problema que aparece en el dataset es la barra del mapa de colores implementado en la simulación, que se ubica en la parte derecha de cada imagen, para eliminar este mapa de colores se plantea el siguiente procedimiento.

Partiendo de una imagen como la mostrada a continuación, se desea eliminar la porción que está en el cuadro rojo.

Figure 4.1: Imagen original.



Para este procedimiento se implementará una herramienta en ubuntu ImageMagick.
Instalación de ImageMagick

```
$ sudo apt-get install ImageMagick
```

Ahora se obtiene la información de la imagen usando el comando identify.

```
$ identify vel2.jpg
```

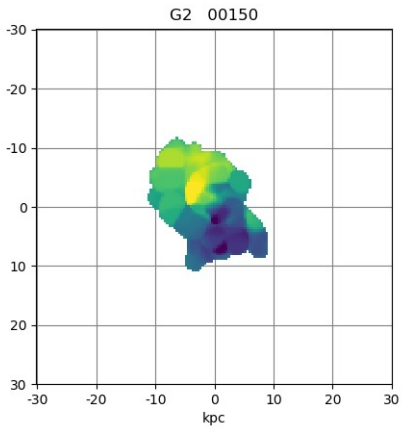
```
$ vel2.jpg JPEG 640x480 640x480+0+0 8-bit sRGB 56.8KB 0.000u 0:00.000
```

El comando anterior muestra que la imagen tiene un tamaño de 640 pixeles de ancho por 480 pixeles de alto, como el mapa de colores que queremos eliminar tiene 120 pixeles de ancho aproximadamente entonces ejecutaríamos el comando convert de la siguiente manera.

```
$ convert vel2.jpg -crop 520x682+0+0 cropped_vel2.jpg
```

La imagén quedaría así: El siguiente paso es implementar un script para

Figure 4.2: Imagen recortada.



automatizar el proceso, ya que son muchas imágenes y no es eficiente hacerlo

una por una. El siguiente script encuentra todos los ficheros jpg, jpeg, png, gif y tiff en el DIR actual, calcula el nuevo tamaño de la imagen y la recorta.

Listing 4.1: Codigo bash

```
PATTERN='.*\.(jpe?g|png|gif|tiff)$'
#
#Busque y lea todos los archivos que coincidan con
#el patron anterior en el DIR actual
#
find $(pwd) -regextype posix-egrep -iregex $PATTERN -type f
-print0 | while read -d $'' file;
do
# Obtiene la imagen de nuevo tamano.
    size=$(identify "$file" | awk '{ print $3 }' | awk -Fx
    -v to_crop=120 '{ print $1"x"$2-to_crop}')
    # nuevo nombre del archivo
    cropped_name=$(dirname $file)/cropped_$(basename "$file")
    # Cortar la imagen
    convert "$file" -crop 500x480+0+0 "$cropped_name"
    # Renombrando al archivo
    [[ $answer =~ ^[0Nn]$ ]] && [[ -e "$cropped_name" ]] &&
    mv -f "$cropped_name" "$file"
done
```

Con esto evitamos entrenar la red con información que no es necesaria.

Chapter 5

La red

En este capítulo se explica paso a paso la construcción del algoritmo y el significado de cada capa de la red. Para el desarrollo del proyecto se tiene una carpeta data que a su vez contiene dos carpetas, una de entrenamiento y otra de validación, éstas contienen tres carpetas una por cada clase definida. la Red esta construida sobre la API de Keras que asu vez usa como backend a Tensorflow.

Keras es una biblioteca de Redes Neuronales de Codigo Abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano

Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible. Inicialmente fue desarrollada como parte de los esfuerzos de investigación del proyecto ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*) Su autor principal y mantenedor ha sido el ingeniero de Google Francois Chollet.

En 2017, el equipo de TensorFlow de Google decidió ofrecer soporte a Keras en la biblioteca de core de TensorFlow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. En este proyecto se usa Python 3.7.3 (default, Mar 27 2019, 22:11:17),keras-2.2.4, tensorflow (3.0.5)

5.1 Importando librerias

```
import sys
```

```
import os
```

Estas librerias nos permiten movernos en carpetas dentro del sistema operativo.

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
```

Este paquete ayuda a preprocesar las imágenes antes de introducirlas a la red.

```
from tensorflow.python.keras import optimizers
```

Aquí esta contenido el optimizador con el que se va a entrenar la red.

```
from tensorflow.python.keras.preprocessing.image import Sequential
```

Sequential básicamente nos permite hacer redes neuronales secuenciales, es decir, que las capas están en orden.

```
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
```

y

```
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
```

Tipos de capas que se implementaran.

```
from tensorflow.python.keras import backend as K
```

este nos permite "matar" otras secciones de keras que se estén corriendo en la máquina e iniciar con una máquina fresca.

5.2 Escribiendo la red

Primero se limpia la máquina

```
K.clear_session()
```

se definen dos variables que contienen las rutas de los datos de entrenamiento y validación.

```
data_ entrenamiento =  "./data/entrenamiento"
```

```
data_validacion =  "./data/validacion"
```

A continuación se definen los parámetros de la red

```
Parameters
epocas=20
longitud, altura = 150, 150
batch_size = 32
pasos = 1000
validation_steps = 300
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool = (2, 2)
clases = 3
lr = 0.0004
```

Las épocas son el número de veces que se va a iterar sobre los datos de entrenamiento, la altura y longitud es el tamaño al cual se van a procesar las imágenes antes de entregárselas a la red, el batch_size es el número de imágenes que entran a la red en cada paso, los pasos son el número de veces que se van a procesar los datos en cada época, los pasos de validación son el número de pasos que se ejecutan después de cada época pero con los datos de validación, los filtrosConv1 y filtrosConv2 son el número de filtros que se aplican en cada convolución, es decir, después de la primera convulución cada imagen tendrá una profundidad de 32 y después de la segunda convolución la profundidad será de 64. El tamaño del filtro hace referencia al tamaño de la ventana que se crea en cada imagen, para la primera convolución la altura del filtro será de 3 y la longitud será de 3, mientras que en la segunda convolución la altura será de 2 y la longitud de 2, el tamano_pool es el tamaño del filtro del MaxPooling, finalmente el número de clases que en este caso es 3, Disco early_merger y late_merger, el learning rate o la velocidad de aprendizaje o el tamaño de paso en el aprendizaje automático es un hiperparámetro que determina en qué medida la información recién adquirida anula la información antigua.

5.2.1 Pre-procesamiento de imágenes

Para pre-procesar las imágenes primero se crea un generador, el cual dice como se van a preprocesar las imágenes y luego se hace la transformación de las imágenes.

```
entrenamiento_datagen =ImageDataGenerator(
      rescale=1. / 255,
      shear_range=0.2,
      zoom_range=0.2,
       horizontal_flip=True)
test_datagen =ImageDataGenerator(rescale=1. / 255)
```

En rescale garantizamos que en la escala de las imágenes todos los valores de pixeles están entre cero y uno, es como una normalización, en el shear_range y zoom_range y horizontal_flip las imágenes se inclinan, aumentan su tamaño e invierten respectivamente, esto se hace de manera aleatoria, no para todas las imágenes. Para la validacion test_datagen sólo se reescalan las imágenes pues se quieren lo mas completas posible.

5.2.2 Tranformando las imágenes

```
entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
        data_entrenamiento,
        target_size=(altura, longitud),
        batch_size=batch_size,
        class_mode='categorical')

validacion_generador = test_datagen.flow_from_directory(
        data_validacion,
        target_size=(altura, longitud),
        batch_size=batch_size,
        class_mode='categorical')
```

en esta parte del codigo se transforman las imágenes de acuerdo a los parámetros antes definidos.

Arquitectura de la red

En esta parte se definen las capas que van a conformar la red neuronal.

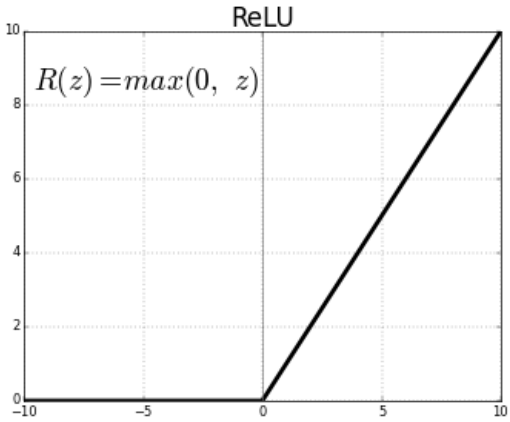
```
cnn = Sequential()
```

Ésta para que la maquina entienda que la red es secuencial, es decir, varias capas apiladas

```
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding =
    "same",input_shape= (longitud, altura, 3), activation= 'relu'))
```

ésta es nuestra primera capa de convolución con los parámetros antes definidos 32 filtros, longitud y altura definidos y una función de activacion ReLU que es una de las más implementadas y que comentaremos antes de continuar con la arquitectura de la red. Tanto en las redes neuronales artificiales como biológicas, una neurona no sólo transmite la entrada que recibe, existe un paso adicional, una función de activación, que es análoga a la tasa de potencial de acción disparado en el cerebro. La función de activación ReLU (rectified linear unit) a pesar del nombre complicado, se define simplemente como R(z)=max(0,z).

Figure 5.1: Función de activación ReLU.



En otras palabras, las ReLUs permiten el paso de todos los valores positivos sin cambiarlos, pero asigna todos los valores negativos a 0. Aunque existen funciones de activación aún más recientes, la mayoría de las redes neuronales de hoy utilizan ReLU o una de sus variantes.

```
cnn.add (MaxPooling2D(pool_size= tamano_pool))
```

En esta capa se busca reducir una matriz (o matrices) creada por una capa convolucional anterior a una matriz más pequeña. Por lo general, la reducción implica tomar el valor máximo o promedio en el área a ser reducida. Seguido se aplican más capas con diferentes parámetros.

```
cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
cnn.add (Flatten())
```

Ahora esta capa Flatten toma las imágenes que son muy profundas y las aplana, es decir, se tiene toda la información en una dimensión.

```
cnn.add (Dense(256, activation= 'relu'))
```

Aquí se establece la cantidad de neuronas de la red, todas estas neuronas están conectadas con las de la capa pasada

```
cnn.add (Dropout(0.5))
```

El drpout a la capa densa durante el entrenamiento se le apagan el el 50 % de las neuronas a cada paso, ya que si todas las neuronas estan activadas la red puede aprender un camino específico, con esto se garantiza que la red aprenda caminos alternos y se adapte mejor a información nueva

```
cnn.add (Dense(clases, activation= 'softmax'))
```

Esta última capa tiene 3 neuronas, las correspondientes a las clases y la función de activacion softmax.

La función de activacion softmax es una función que proporciona probabilidades para cada clase posible en un modelo de clasificación de clases múltiples. Las probabilidades suman exactamente 1.0. Por ejemplo, softmax puede determinar la probabilidad de que una imagen en particular sea un perro en 0.9, un gato en 0.08 y un caballo en 0.02.

Ahora se introduce la última capa para optimización, aquí se le dice a la red que vea que tan bien o mal esta entrenada a partir de categorical_crossentropy y una métrica accuracy, esto es lo que se intenta mejorar.

```
cnn.compile(loss='categorical_crossentropy',
            optimizer=optimizers.Adam(lr=lr),
            metrics=['accuracy'])
```

posteriormente, se le dice cómo se va a entrenar la red y las epocas en que lo va a hacer

```
cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validation_steps)
```

finalmente se exporta el modelo

Listing 5.1: Exportar el modelo

```
target_dir = './modelo/'
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
cnn.save('./modelo/modelo.h5')
cnn.save_weights('./modelo/pesos.h5')
```

Este código crea una carpeta dentro del directorio llamada modelo, y exporta el modelo y los pesos por separado.

5.2.3 Predictor

Finalmente escribimos el segundo código, que llama al modelo y los pesos almacenados anteriormente y genera la predicción por clases, la ejecución en la terminal sigue el orden: llamado al compilador, predictor y nombre de la imagen que se quiere clasificar (python predict.py imagen.jpg)

Listing 5.2: Predictor

```
#cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict_classes(x)
  print(array)
  result = array[0]
  answer = np.argmax(result)
  if answer == 0:
    print("pred: Disco")
  elif answer == 1:
    print("pred: earlymerger")
  elif answer == 2:
    print("pred: latemerger")
  return answer
  filename=str(sys.argv[1])

if __name__=='__main__':
  predict(filename)
```

Chapter 6

Conclusiones y perspectivas

Se logro implementar una red neuronal para la clasificacion de imagenes de galaxias respecto al grado de colision entre ellas, los resultados de la prediicion de la red son aceptables por la simplicidad de la red, los problemas se presentan en los limites definidos para el cambio en el grado de colisión entre Early_merger y late_merger, estos problemas se esperaban debido al tamaño del dataset y el uso de pocas capas en la red. como perspectiva queda la optimizacion de la red para aumentar la precicion. El cambio de funcion de activacion para la salida de la red de categorica a una probabilistica.Finalmente un pequeño comentario sobre el aporte del trabajo respecto a la formacion para el proyecto de grado de maestria.

A menudo se asume que las arquitecturas de redes convolucionales profundas garantizan la generalización o invarianza de los objetos a transformaciones como las traslaciones , las rotaciones y las pequeñas deformaciones. Un problema fundamental en el reconocimiento de objetos es el desarrollo de representaciones de imágenes que son invariantes a dichas transformaciones, Este problema persiste en las CNN modernas como VGG16, ResNet50 e InceptionResNetV2, estas pueden cambiar drásticamente su salida cuando una imagen simplemente se traslada en el plano de la imagen en unos pocos píxeles. El presenta rtrabajo es el punto inicial al proyecto de maestria en el que se busca eliminar dichos problemas en las pequeñas transformaciones en la CNN.

Bibliography

[1] D. L Poole, A. K. Mackworth, and R. Goebel. *Computational intelligence: a logical approach*, volume 1. Oxford University Press New York, 1998.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.