Reconocimiento de imágenes

- **Análisis del problema**

Es muy fácil para nosotros reconocer objetos en nuestra vida diaria, sin importar que cambien de tamaño, se encuentren desplazados, rotados o incluso cuando estén parcialmente obstruidos o sólo se tenga a la vista una parte. Por otro lado, para la computadora este proceso es complicado, y a lo largo de los años se han buscado diversos métodos para atacar el problema.

Básicamente cuando se aborda un tema de reconocimiento de imágenes comúnmente se siguen los siguientes pasos:

- Preprocesamiento
  Se aplican diversas técnicas para mejorar la imagen, quitando el ruido y realzando las características para los siguientes pasos.
- Segmentación
  Aísla los objetos de interés de la imagen.
- Extracción de rasgos
  Se le asignan valores numéricos a los rasgos de la imagen que nos interesan.
- Clasificación
  De acuerdo a los valores de los rasgos el programa asigna la imagen a un clase, u objeto de interés.

Las redes neuronales convolucionales (CNN o Convent) son una clase de redes neuronales profundas muy utilizadas para el reconocimiento de imágenes.
Este tipo de redes constan de múltiples capas, siguiendo el proceso antes descrito, pero relativamente más rápido en comparación con otros algoritmos de clasificación de imágenes.

En cada capa el algoritmo toma un cuadrado pequeño y comienza a aplicarlo sobre la imagen de tal manera que se pueden detectar características, a medida que profundizamos en las capas de la red neuronal esta puede combinar los hallazgos y aprender continuamente conceptos más complejos.

Es por lo anterior que se decidió utilizar este tipo de redes.

- **Forma de trabajo**

Cada integrante del equipo se encargó de entrenar un animal:

Integrante	Animal
Christian	Jirafa
Uriel	Avestruz
Daniel	Cuyo
David	Hipopótamo
Karla	Pingüino

Así como de crear su respectiva prueba.
Para las clases que conectan el programa de decidió una participación colectiva, ya que estas clases tendrán interacción con las redes hechas por cada uno.

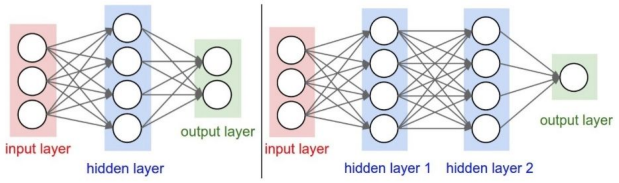
- **Arquitectura de la red neuronal**

Como muchas cosas en computación, si no es que todo, para resolver un problema complejo se basan en un modelo natural y buscan adaptar este lo mejor posible a un proceso que pueda efectuar la computadora, así que para el caso de reconocimiento de imágenes se basan en la manera en que el humano y los animales aíslan toda la información para obtener en unos instantes el resultado de lo que se observa.

Lo que hace la corteza cerebral es tener una jerarquía de neuronas, por ejemplo, el área visual primaria se ocupa de las características visuales de bajo nivel (información básica de colores y contornos), y alimenta con esta información a neuronas que están en rangos inferiores, donde cada uno de estos niveles se ocupa de aspectos más específicos o abstractas de la información, es así como surge las redes neuronales convolucionales. Por lo que la arquitectura general de una CNN consiste en una capa de entrada, capas ocultas y una capa de salida.

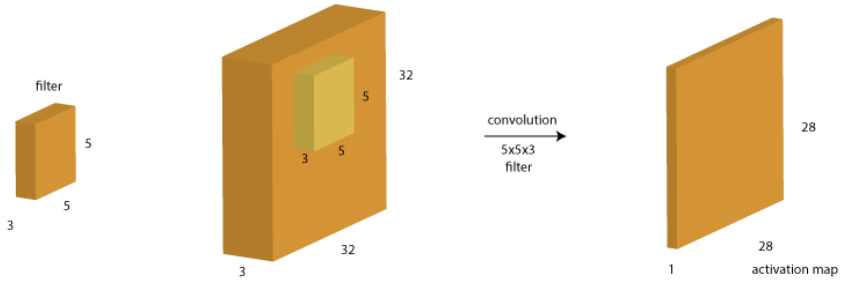
Como es mucha información la que se procesa en una imagen, sobretodo si la imagen es grande, las CNN trabajan modelando de forma consecutiva pequeñas piezas de información, en donde cada capa de la red evaluará información cada vez más específica para que las capas finales intententen hacer coincidir una imagen de

entrada con todas los patrones y mostrar una predicción final como una suma ponderada de todos ellos.

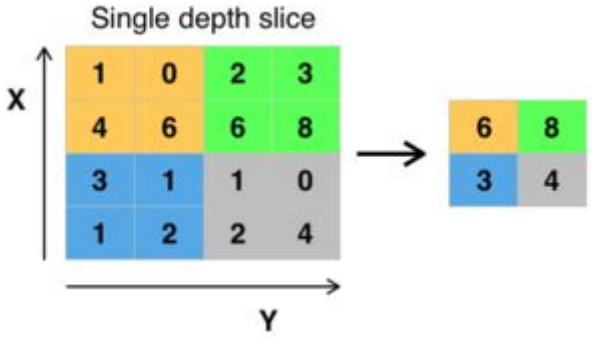


Capas utilizadas:

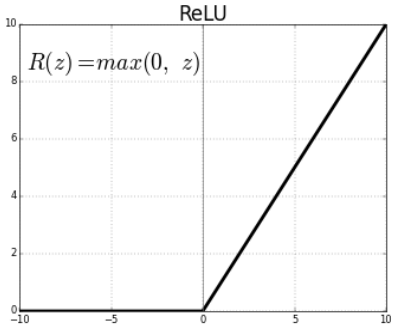
- Convolucional
  Aplica un filtro a una imagen que recibe como entrada, esto nos devuelve un mapa de dimensiones menores pero con las características de la imagen original .



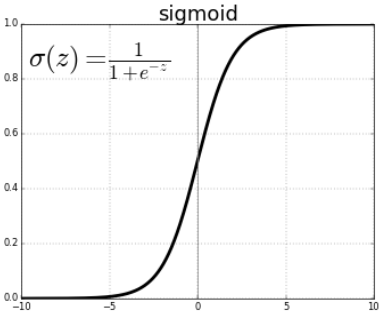
- Pooling (de Reducción)
  Se coloca generalmente después de una capa convolucional, y su principal función es reducir el alto y ancho de la imagen para evitar una sobrecarga de cálculos en capas próximas.
  Se divide a la imagen de entrada en un conjunto de rectángulos y usualmente para formar el nuevo mapa se queda con el máximo valor (max-pooling), aunque también es posible calcular el promedio de valores del rectángulo y utilizar dicho valor.



- Activación (SoftMax)
  Lo que hace esta capa es detectar una gama de patrones y predecir con un cierto grado de precisión la etiqueta para una imagen determinada, por ello, esta capa es la comunicación entre las capas antes descritas, y la capa final.
  Normalmente función de activación de ReLu se usa en capas ocultas, mientras que la capa final generalmente consiste en una función de activación de SoftMax.

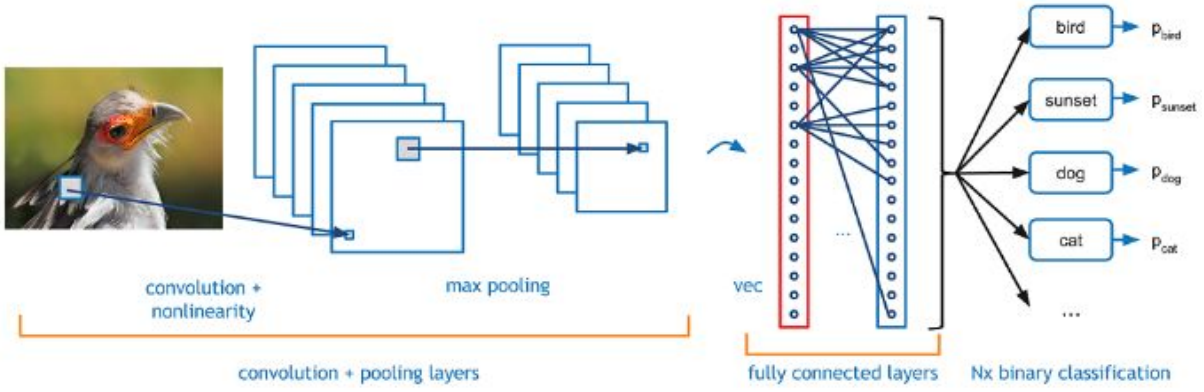


ReLu: Permiten el paso de todos los valores positivos sin cambiarlos y asigna todos los valores negativos a 0.



SoftMax: A menudo se encuentra en la capa final, actúa básicamente como un normalizador y produce un vector discreto de distribución de probabilidad, esta será la probabilidad de que la imagen de entrada pertenezca a una cierta clase.

Así nuestra red por animal tendrá una estructura como la siguiente



Después de plantear la red y su funcionamiento, se procede a entrenarla, proporcionando diversas imágenes de los animales de interés y de otras cosas. Por último se guarda cada red para juntarlas en una clase principal posteriormente.

El tiempo aproximado de la creación de todo el programa fue de 5 días.
Como para cada animal se utilizan básicamente las mismas capas y el mismo modelo, decidimos describir el proceso una única vez.

ERRORES

- De conocimiento
  En general nos parece que el código no está mal, pero un problema muy grande es que sentimos que entendemos muy por encima las funciones que se utilizan, haciendo el proceso de construcción difícil.
- Obtener imágenes para entrenar la red neuronal
  A pesar de contar con programas que descargan imágenes de Internet notamos que había un límite bajo para descargarla y los programas que permitían descarga masiva metían imágenes que no tenían nada que ver con el animal buscado.
- Con Github
  - Conflictos al hacer Merge de dos ramas
  - Subir el enviroment
    Github marcaba error al querer subir los módulos necesarios para la red neuronal por ser muy pesados (Tensorflow).
- Con el código
  - Actualizar comandos de Keras que producían advertencias
  - Organizar las carpetas correctamente para que el programa encontrará las imágenes correspondientes.
  - Entrenar la red: fue muy complicado entrenar las redes ya que en ocasiones decía que la probabilidad era un cierto número y al probarla eso no correspondía.
  - Problemas de compatibilidad con Python y Python 3

- **Pruebas**

Como hemos visto en clase un tipo de pruebas es la afirmación de un comportamiento esperado de cada función. Para nuestro caso en concreto, cada red regresará un valor numérico que indica la probabilidad de que la imagen pertenezca a una cierta clase, por lo que cada capa sólo contiene dos funciones dignas de prueba: propagación hacia adelante y hacia atrás. En otras palabras lo que se prueba es que cada red tenga como resultado una alta probabilidad para predecir si

la imagen pertenece o no a alguna clase de interés o si se tiene que seguir a otra red, esto a su vez implica que cada capa ya sea de convolución, reducción o de activación funcionen como deberían.

Como cada modelo de animal funcionan y se entrenaron de la misma forma basta con hacer una prueba padre, en la cual por medio de un ciclo le proporcione imágenes correspondientes de un animal y nos devuelva el porcentaje de error, de tal manera que que para probar a un animal en específico bastará con llamar a la prueba padre y especificarle (dirección de las imágenes y modelo) que animal en concreto se probará.

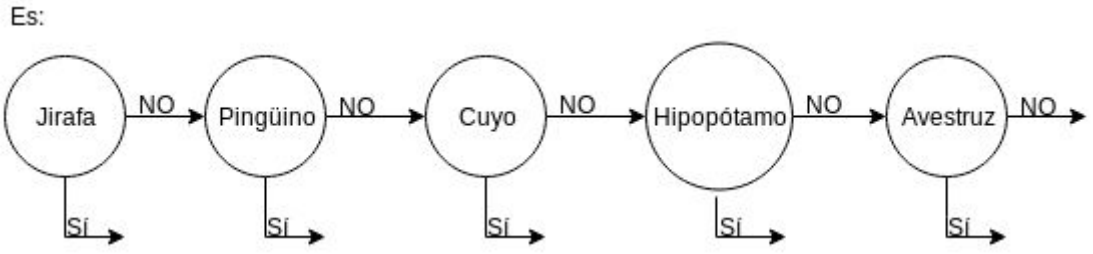
También es necesario probar algunas funciones del archivo Classifier, para asegurar que se guarda y que es posible acceder al modelo una vez que la red ya ha sido entrenada.

Para la clase principal se debe probar que siguen una ruta viable, es decir que si el modelo arroja que no son el animal del vértice en el que están, el programa debe moverse al vecino correspondiente, así como validar que antes de dar el resultado final se pase por la verificación de la última red, que es responsable de decidir si la imagen efectivamente es alguno de los 5 animales, de lo contrario, en cualquier caso, diría que la imagen pertenece a un animal de los 5 aunque este no lo sea.

- **Unión de clases**

Se pensaron dos formas de implementar todos los modelos, de tal manera que al ingresar una imagen en el programa este sea capaz, haciendo uso de todos los modelos previamente entrenados, de predecir si la imagen contiene alguno de los 5 animales (jirafa, pingüino, cuyo, hipopótamo y avestruz) o no.

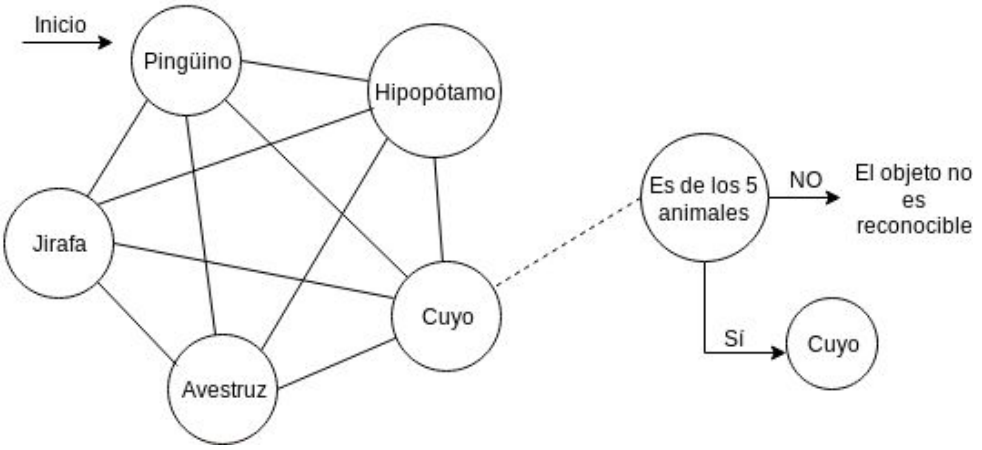
La primera consistía en unir las redes de manera secuencial, de la siguiente manera:



Esta implementación requiere de un entrenamiento extenso de cada modelo, ya que para enseñar a la red neuronal si la imagen contiene una jirafa (primer modelo) es equivalente a enseñarle todo lo que no es una jirafa, siendo necesario mucho

tiempo de procesamiento y cantidad de imágenes para que el modelo sea un modelo decente, y este proceso se tendría que hacer por cada modelo de animal, dando un total de 5, aunado a que se contaban con pocas máquinas para entrenar debido a incompatibilidades que se presentaron con python, decidimos descartar esta implementación.

Lo segundo, que se pensó, y se acordó, es acomodar los modelos en una gráfica completa de 5 vértices:



Dada una imagen de entrada se procede a tomar un vértice de la gráfica, cada uno de ellos está entrenado para decidir si la imagen es del animal correspondiente al vértice o si es del vecino, en caso de ser el vecino se procede a verificar lo mismo con este vértice, de lo contrario permanece en el vértice inicial y repite el proceso con otro vecino, de tal manera que siempre se están comparando 2 a 2, si es por ejemplo, una jirafa o un pingüino.

En la última comparación es claro que la decisión estará entre 2 opciones de las 5 posibles, pero podría pasar que la imagen no corresponda a ninguna de las 2, es por ello que antes de decidir se pasa a una red que se fija si la imagen ingresa corresponde a alguno de los 5 animales, en este caso se dará como resultado el último vértice, de lo contrario regresará queel objeto de la imagen no se puede reconocer.

Esta última red podría pensarse como un paso previo a la gráfica completa, pero se optó por ponerla al final ya que como no es una red perfecta de ponerla al inicio podría pasar que de una respuesta incorrecta y no proceda a verificar si es un animal, claro que esto también puede pasar en el orden en que nosotros dictamos, pero al pasar por más redes se reduce el margen de error.

Con este modelo solucionamos el problema que se tenía al entrenar una red donde se decide si es un objeto o no es ese objeto, pero también se reduce la escalabilidad del proyecto, haciendo poco viable agregar más animales.

- **Retrospectiva**

Creemos que la mayor dificultad a la hora de desarrollar este proyecto es la complejidad de los temas que se esconden en él y el no ver de manera más extensa la teoría necesaria.

No comprender los temas necesarios para la resolución del problema nos hacía sentir que estábamos resolviendo el problema a ciegas, sin entender del todo que estaba pasando. Intentando buscar en la documentación el problema se hizo más grande ya que era muy extensa la información y requería de conocimientos amplios de álgebra y cálculo, eso nos llevó un poco a la frustración y a que algunos integrantes del equipo se perdieran en el camino. Haciendo aún más difícil lograr el cometido asignado.

Uno de los mayores aciertos es en parte debido al problema más grande, perseverancia, el equipo nunca se detuvo, ni se dio por vencido a pesar de que el entrenamiento parecía imposible.

- **Fuentes consultadas**

  - https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/
  - https://medium.com/datos-y-ciencia/construye-tu-primer-clasificador-de-deep-learning-con-tensorflow-ejemplo-de-razas-de-perros-ed218bb4df89
  - https://medium.com/nybles/create-your-first-image-recognition-classifier-using-cnn-keras-and-tensorflow-backend-6eaab98d14dd
  - https://towardsdatascience.com/virtual-environments-104c62d48c54
  - https://machinelearningmastery.com/save-load-keras-deep-learning-models/