Clasificador de imágenes

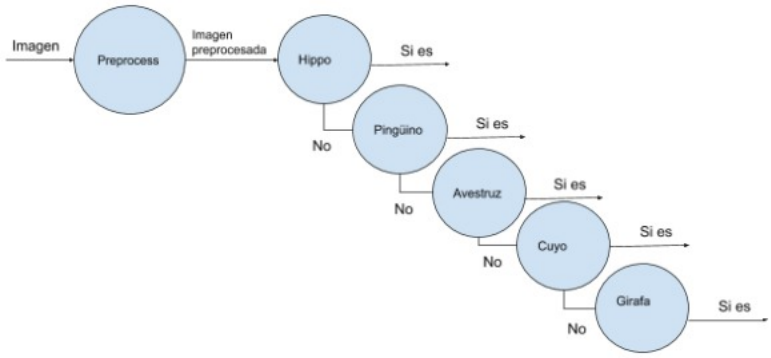
Alejandro Hernández Cano [*]
Fernando Daniel Castillo Barrón [**]
Tinoco Miguel Laura Itzel [***]
Saavedra Escalona Braulio Rubén [****]

1. Análisis del problema

Se requiere programar un clasificador de fotografías de animales. El programa debe recibir como entrada la fotografia de un animal y como salida el programa imprimirá si se trata de un cuyo, hipopótmo, jirafa, pingüino o avestruz.

Se requiere que el programa use TensorFlow como libreria de Machine Learning y Keras para el modelo de predicción de datos. Deben usarse 5 modelos diferentes (uno para cada animal), que se encarguen de hacer una predicción específica según su tarea y dar una respuesta positiva o negativa (es o no es el animal), pasando al siguiente en caso de obtener una respuesta negativa, como se explica en el siguiente diagrama.

Figura 1: Arquitectura general del programa



[*]ale.hdz333@ciencias.unam.mx
[**]danielcastillo59814@ciencias.unam.mx
[***]itzel_tinoco@ciencias.unam.mx
[****]braulio@ciencias.unam.mx

1.1. Alternativas de solución

Para el lenguaje de programación de python, existen diferentes librerias de Machine Learning y Deep Learning que pueden utilizarse para solucionar el problema planteado:

- Scikit-learn: librería de python para Machine Learning y Análisis de Datos. Está basada en NumPy, SciPy y Matplotlib. Permite realizar aprendizaje supervisado y no supervisado. Usado para resolver problemas tanto de clasificación y como de regresión.
- PyTorch: permite el cálculo numérico eficiente en CPU y GPUs. Usa cálculos matriciales y de derivadas masivos y paralelizables para aprendizaje profundo (deep learning).

1.2. Mejor alternativa

TensorFlow version 2.0 presenta mejoras frente a las demas, y es por ello que se utilizará como mejor alternativa:

- La documentación es bastante clara para seguir los pasos de instalación, pruebas y opciones avanzadas para su uso en Python.
- Keras permite diseñar diferentes tipos de modelos para redes neuronales, lo que lo hace flexible y nos ayudará a elegir el modelo adecuado a nuestro problema
- Pensando en trabajo a futuro, cuenta con implementaciones en diferentes plataformas (android, swift, javascript), permitiendo reutilizar la estructura del proyecto para diferentes distribuciones

2. Reparto de trabajo

El reparto del trabajo inicialmente fue llevado a cabo como se sugirió en el documento de planteamiento de problema. Lamentablemente, uno de los integrantes del equipo por causas de fuerza mayor no pudo contribuir en este proyecto. Al final los modelos fueron realizados por:

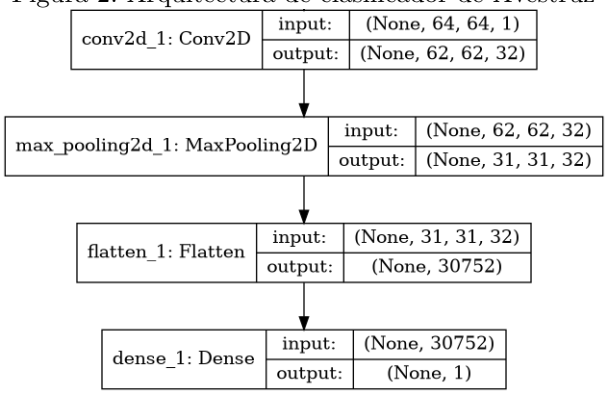
- Avestruz: Alejandro.
- Hipopotamo y cuyo: Daniel.
- Jirafa: Itzel.
- Ping red. ..uino: Braulio.

Cada uno se encargó de la documentación y preubas unitarias de cada clase. Una vez terminados los modelos de clasificación de cada animal, Alejandro se encargó de realizar la clase que integra cada modelo y el programa principal para interactuar con el usuario. Posteriormente Itzel y Braulio realizaron las pruebas unitarias para esta clase.

3. Sobre los modelos

A continuación describiremos la arquitectura de nuestro clasificador de avestruces. Podemos ver un diagrama de este modelo en la figura 2. Daremos más detalles sobre esta red neuronal a continuación. Nuestra red neuronal recibe imágenes de tamaño 64×64 en escala de grices. Ahora daremos detalles sobre las capas que conforma nuestra red:

Figura 2: Arquitectura de clasificador de Avestruz

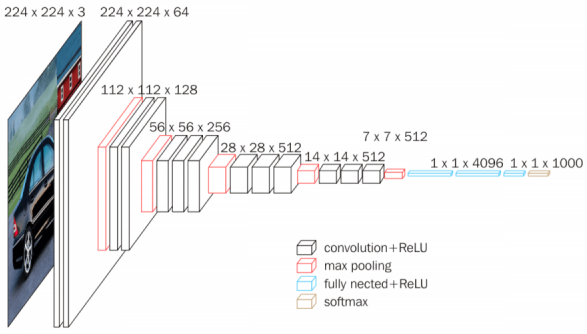


1. **Capa Convolucional 2D**.
   - Entrada: Tensor de tamaño $64 \times 64 \times 1$. Nuestra imagen en escala de grices.
   - Filtros: 32 filtros.
   - Tamaño del kernel: 3×3.
   - Función de activación: Función ReLU [1].
2. **MaxPooling** [2]
   - Entrada: Tensor de tamaño $62 \times 62 \times 32$. La salida de la capa anterior.
   - Paso: 2×2.
3. **Flatten**
   - Entrada: Tensor de tamaño $31 \times 31 \times 32$.
4. **Capa densamente conexa**
   - Entrada: Tensor de tamaño 30752.
   - Función de activación: Función sigmoide [3].
   - Unidades: 1.

Y para el modelo utilizamos el algoritmo de optimización Adam con función de pérdida a optimizar de entropía cruzada binaria. Describiremos ahora los motivos para cada una de las especificaciones de nuestro modelo y daremos una pequeña descripción del proceso para llegar a esta configuración y de otras alternativas que se probaron antes de llegar a la configuración actual.

En los últimos años se ha probado la efectividad de redes neuronales convolucionales para clasifiación de imágenes, por lo tanto se decidió utilizar una red neuronal de este estilo para resolver nuestro problema. En particular, se estudió y analizó la arquitectura de la red neuronal desarrollada por la universidad de Oxford, la red VGG [4], que obtuvo una presición mayor al 97 % en el famoso conjunto de datos ImageNet [5]. En la figura 3 podemos observar la arquitectura de esta red neuronal.

Figura 3: Arquitectura de VGG



Dado que esta red fue propuesta en un problema de clasificación que quería identificar images de más de 20000 clases, decidimos simplificar este modelo y la primera propuesta que desarrollamos fue de tres bloques convolucionales, separados cada uno por un max pooling, simulando un VGG pequeño. Otra de las observaciones iniciales de esta red fue que la última función de activación, la función softmax, nos daba resultados muy difíciles de interpretar y para la red de optimizar, probablemente dado que la clasificación de nuestro problema era binario. Después de varios experimentos se observó que ReLU daba los mejores resultados.

Después se experimentó cambiando la cantidad de bloques convolucionales, cambiando el tamaño del filtro y agregando una capa flatten antes de la ultima capa. No se observó grandes cambios en la presición en el conjunto de prueba después de estos experimentos, sin embargo se observó una gran diferencia en el tiempo de entrenamiento que requería la red neuronal para el entrenamiento, por lo tanto se optó por una de las arquitecturas mas sencillas que se probó, dando como configuración final la arquitectura descrita anteriormente.

Utilizamos la primera capa con filtro de tamaño 3×3 ya que este es el filtro más pequeño que se puede utilizar para no perder la noción de arriba/abajo, izquiera/derecha dentro de una imagen. Después de esto llegamos a una capa max pooling para decrementar la dimensionalidad dentro de nuestra red, pero sin perder la información que esta contiene. Útil tanto para agilizar el tiempo de entrenamiento como para reducir la complejidad de nuestra red para finalmente poder llegar a la solución deseada.

Después se utiliza una capa Flatten para poder pasar nuestra red a un tensor de una sola dimensión ya que para este tamaño le podemos aplicar una última capa densamente conexa para poder determinar nuestra predicción. Nuestra arquitectura es simple, lo cual garantiza un rápido entrenamiento y siguiere una mayor velocidad en la convergencia de nuestro modelo buscado, pues en resultados experimentales se observó que la función pérdida de nuestra red decrementaba insignificantemente a partir del epoch 5.

Finalmente, se decidió optar por el algoritmo de optimización Adam [6] pues este ha sido probado

por ser efectivo en muchos trabajos anteriores y, como es descrito por sus creadores, combina las ventajas de dos de los algoritmos de optimización de stochastic gradient descent más utilizados: AdaGrad y RMSProp. La selección de función de pérdida de entropía cruzada binaria es inherente al tipo de problema que nos estamos enfrentando y ha sido probada efectiva en trabajos anteriores, por lo tanto se optó por esta función.

Al momento de explorar posibilidades de modelo probamos este modelo para todas las clases y se determinó que, en general, esta arquitectura tenía muy buen desempeño para todas las clases, por lo tanto fue utilizado para cada clasificador de animal. El tiempo de entrenamiento de esta red es extraordinariamente rápido ($<$ 1 min para conjuntos de datos medianos o pequeños).

4. Sobre las pruebas

4.1. Avestruz

- Total de imágenes para el entrenamiento: 3025 imágenes (365 avestruz, 2660 otro anima)
- Porcentaje de acierto según las pruebas: 76 %

4.2. Hipopótamo

- Total de imágenes para el entrenamiento: 4005 imágenes (2001 hipopótamos, 2004 no hipopótamos)
- Porcentaje de acierto según las pruebas: 85 %

4.3. Jirafa

- Total de imagenes para el entrenamiento : 3020 imagenes (920 jirafas, 2100 no jirafas)
- Porcentaje de acierto al evaluar el modelo : 87 %

4.4. Pingüino

- Total de imagenes para el entrenamiento : 3020 imagenes (397 pingüinos, 2100 no pingüinos)
- Porcentaje de acierto al evaluar el modelo :85 %

4.5. Pruebas generales

- Son 6 test:
- 1:Falsos Negativos: Se tienen 100 imágenes, en las cuales no aparece ninguno de los 5 animales, se busca ver cuántas veces el clasificador cuenta alguno de estos cinco animales donde no los hay.
- 2:Reconoce Hipopotamo: Se dan 100 imágenes donde aparecen hipópotamos. Se busca la cantidad de veces que el clasificador los detecta

- 3:Reconoce Pingüino: Se dan 100 imágenes donde aparecen pingüinos. Se busca la cantidad de veces que el clasificador los detecta
- 4:Reconoce Avestruz: Se dan 100 imágenes donde aparecen avestruces. Se busca la cantidad de veces que el clasificador los detecta
- 5:Reconoce Cuyo: Se dan 100 imágenes donde aparecen cuyos. Se busca la cantidad de veces que el clasificador los detecta
- 6:Reconoce Jirafa: Se dan 100 imágenes donde aparecen jirafas. Se busca la cantidad de veces que el clasificador los detecta

5. Sobre la integración de los clasificadores

La itegración de nuestras clases en una sola clase principal fue muy directa. Debido a que cada clasificador heredaba el comportamiento de una clase más general, AnimalClassifier, pudimos realizar nuestro clasificador general de manera sencilla, especificamos los clasificadores que queríamos utilizar, los instanciamos y cargamos los pesos de cada uno de ellos. Para poder realizar una predicción, utilizamos un flujo muy parecido al especificado en la figura 1.

Utilizar el código principal es muy sencillo. Este tiene dos modalidades, una para predecir imagenes guardadas en el sistema de archivos local y la segunda es especificando una URL de alguna imagen que se desee probar. Cada una de estas opciones se especifican utilizando la bandera `--mode`, teniendo como opciones disponibles `local` o `link`. El uso del código principal es: `main.py [--mode link,local] [imagenes ...]`.

Por ejemplo, si queremos predecir imagenes de nuestra computadora podemos hacer `main.py --mode local path/to/imgs/img1 path/to/imgs/img2`. Después de esto se cargarán los modelos y se desplegará una ventana con cada una de las imagenes dadas y el título de esa ventana será la predicción que tiene nuestro modelo. Se puede observar un ejemplo de ejecución de nuestro clasificador en la figura 4.

Figura 4: Ejemplo del uso del programa principal

Referencias

[1] Hahnloser, R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; Seung, H. S. (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". Nature. 405: 947–951.

[2] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE). pp. 3642–3649.

[3] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco (eds.). From Natural to Artificial Neural Computation. Lecture Notes in Computer Science. 930. pp. 195–201.

[4] Karen Simonyan, Andrew Zisserman (2015). Very Deep Convolutionak Networks for large-scale Image Recognition.

[5] Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Fei-Fei, Li (2009), ImageNet: A Large-Scale Hierarchical Image Database", 2009 conference on Computer Vision and Pattern Recognition

[6] Diederik P. Kingma, Jimmy Ba (2014). Adam: A Method for Stochastic Optimization. 3rd International Conference for Learning Representations, San Diego.