Análisis Estructural con Deep Learning para Identificación de Fallas Geológicas

Structural Analysis with Deep Learning for Identification of Geological Faults

Andrade C.[1], Bonifaz M.[2], Flores K.[3], Mejía C.[4], Parión C.[5]

[1-5] *Universidad Central del Ecuador, Facultad de Ingeniería en Geología, Minas, Petróleos y Ambiental, Quito Ecuador*

[1] email: caandradem@uce.edu.ec
[2] email: mcbonifaz@uce.edu.ec
[3] email: kafloresa@uce.edu.ec
[4] email: cimejia@uce.edu.ec
[5] email: cmparion@uce.edu.ec

RESUMEN

La ejecución de algoritmos se puede implementar en distintas ramas de la geología como la geología estructural, donde nos permite realizar la identificación de fallas geológicas locales y regionales de forma rápida, sencilla y confiable en imágenes satelitales, para su detección se implementó la red convolucional YoloV3 que utiliza deep learning y CNN, obteniendo así un modelo para el análisis estructural de la zona de estudio. El proyecto tuvo como finalidad la aplicación del aprendizaje automático de la detección de objetos en las ciencias de la Tierra mediante la predicción de fallas en imágenes satelitales que contribuirá en estudios regionales para la realización de una modelización geológico - estructurales y definir su configuración tectónica.

Palabras clave: algoritmos, red convolucional, predicción, detección, geología, fallas

ABSTRACT

The execution of algorithms can be implemented in different branches of geology such as structural geology, where it allows us to identify local and regional geological faults quickly, easily and reliably in satellite images, for their detection the Yolo system was used in the convolutional network, thus obtaining a model with a validation of 70% for the structural analysis of the study area. The purpose of the project was the application of automatic learning of object detection in Earth sciences by predicting faults in satellite images that contribute to regional studies to carry out geological-structural modeling and define their tectonic configuration.

Keywords: algorithms, convolutional network, prediction, detection, geology, faults

1. Introducción

El aprendizaje automático es considerado una disciplina de las ciencias computacionales que permite a una máquina tener la capacidad de aprender mediante la aplicación de algoritmos en programación, centrado en el modelamiento predictivo que implica un riguroso análisis de datos y son ampliamente utilizadas en negocios mecanismo de ayuda a la toma de decisiones (Espino C. 2017) una de las técnicas que ha desarrollado este aprendizaje se denominada Deep Learning el cual se ha convertido en un método habitual para llevar a cabo la detección de objetos que forma parte del reconocimiento de objetos, que no solo identifica el objeto, sino que lo localiza en una imagen[2], lo que implica el uso de una red neuronal convolucional (CNN) que detecte una cantidad limitada o específica de objetos.

Este artículo propone la aplicación de técnicas de Deep Learning en el tratamiento de imágenes satelitales para la identificación de fallas mediante la elaboración de un dataset extenso que contenga imágenes satelitales obtenidas de Google Earth. El diseño e implementación de un modelo basado en Deep Learning (CNN) permitirá determinar la factibilidad del uso de las redes neuronales en un análisis geológico – estructural, obteniendo así una localización exacta de las fallas dentro del área a estudiar, facilidad de trabajo a diferentes escalas, una mayor cobertura, además de una diferenciación de los diversos tipos de estructuras, y la optimización del tiempo en su identificación superando así los métodos tradicionales de un análisis estructural.

2. Materiales y métodos

La metodología a desarrollar en el proyecto se presenta a continuación mediante un flujograma.

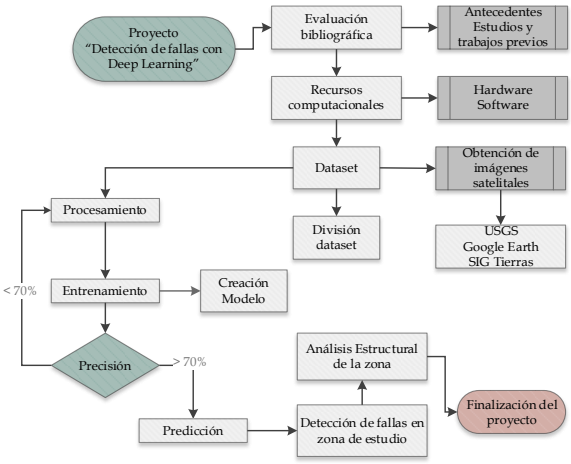


Figura 1. Flujograma de la metodología

2.1. Ubicación

La zona de estudio se encuentra ubicado en la provincia de Imbabura, norte del Ecuador. Geomorfológicamente la provincia se encuentra en la Cordillera Occidental, limita con el Valle Interandino al este y la región Litoral al oeste.

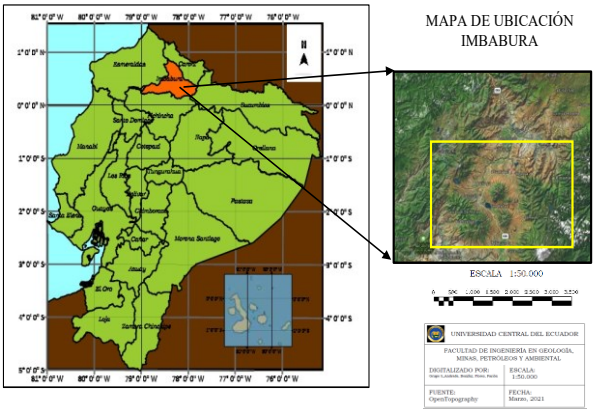


Figura 2. Mapa de Ubicación de la zona de estudio

2.2. Geología Estructural

La zona de estudio se encuentra en su mayor parte en la Cordillera Occidental el cual es

producto de una compleja evolución geológica que involucra en su mayoría, eventos de acreción y post-acreción. Las principales fallas que atraviesan esta zona están relacionadas con el sistema de fallas dominantemente en sentido NE-SO.

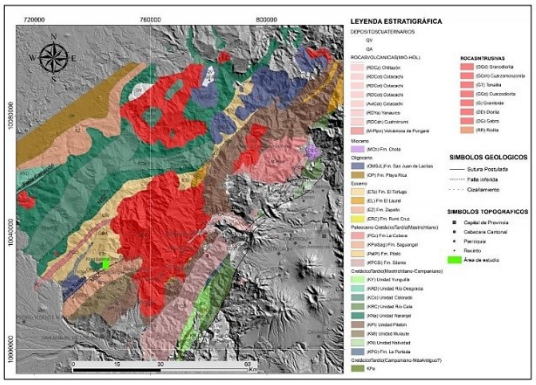


Figura3. Mapa Geológico de Cordillera Occidental entre 0°-1°N (Modificado de Boland et al.,)

El área comprendida entre 0°-1°N, se encuentra conformada por una serie de fallas en sentido NE-SO. Se encuentra, además, un conjunto de fallas conjugadas E-O caracterizadas como zonas altamente tectonizadas que cortan de manera transversal al sistema regional principal de fallamiento (Boland et al., 2000).

La dirección de deformación dominante en el área tiene un rumbo NE, que afectó fuertemente a las unidades Ambuquí y Macuchi. Las fallas E-W generan alto tectonismo en la zona, todas estas fallas pertenecen al gran sistema de fallas Calacalí-Pujilí-Pallatanga (CPPF), forma parte de la zona de sutura expuesta en la porción central de la Cordillera Occidental (Hughes y Pilatasig 2002).

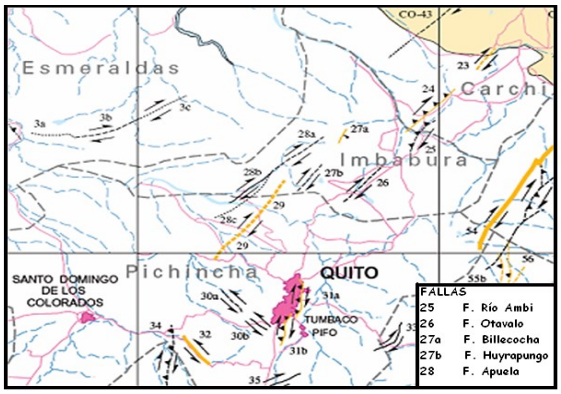


Figura4. Mapa de Fallas activas del Ecuador 2013.

2.3. Recursos

2.3.1. Hardware

Para la realización de este proyecto se utilizó una laptop de marca Dell Inc, modelo Inspiron 14-3467 con un sistema operativo de Microsoft Windows 10 Pro. Esta laptop cuenta con un procesador Intel Core i5-7200 CPU @2.50 GHz, 2701 Mhz de 63 bits y una tarjeta de video de Intel HD Graphics 620. Posee las siguientes unidades almacenamiento; RAM de 8GB y un Disco C de 930GB.

2.3.2. Software

Dentro de la realización de este proyecto se utilizaron varios programas que se describen a continuación

Google Earth: obtención de imágenes satelitales, este programa nos permitió la descarga de estas imágenes para obtener nuestro dataset.

LabelImg: Para la detección de objetos, es una herramienta de anotación de imágenes gráficas

y cuadros delimitadores de objetos de etiquetas en imágenes, se encuentra escrito en Python y usa Qt para su interfaz gráfica. Las anotaciones se guardan como archivos XML en formato PASCAL VOC, el formato utilizado por ImageNet. Además, también es compatible con el formato YOLO. (Tzutalin, 2015)

2.3.3. Lenguaje de programación

Python

Este lenguaje de programación es uno de los más utilizados en proyectos de Machine Learning debido a su fácil uso y versatilidad haciendo que sea un lenguaje amigable y de amplio crecimiento.

Este fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba. Primariamente se concibe para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC. El 16 de octubre del 2000 se lanza Python 2.0 que contenía nuevas características como completa recolección de basura y completo soporte a Unicode. (Sega et al, 2018)

Editor de programación Jupyter Notebook

Jupyter Notebook es una interfaz web de código abierto que permite la inclusión de texto, video, audio, imágenes, así como la ejecución de código a través del navegador en múltiples lenguajes. Esta ejecución se realiza mediante la comunicación con un núcleo (Kernel) de cálculo. Aunque en principio, el equipo de desarrollo de Júpiter Notebook incluye por defecto únicamente el núcleo de cálculo Python, el carácter abierto del proyecto ha permitido aumentar el número de núcleos disponibles, incluyendo, por ejemplo Octave, Julia, R, Haskell, Ruby, C/C++, Fortran, Java, SageMath, Scala, Colab, Matlab y Mathematica. Esta interfaz, agnóstica del lenguaje (de ahí su nombre al unir 3 de los lenguajes de programación de código abierto más utilizados en el ámbito científico: Ju-lia, Python y R), puede suponer por tanto una estandarización para mostrar el contenido de cursos científicos, sin encontrarse limitado a la adopción de un ´único lenguaje. (Cabrera et al, 2014)

Google Colab

Google Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube. Colab te permite escribir y ejecutar código de Python en un navegador, con las siguientes particularidades: sin configuración requerida, acceso gratuito a GPU, facilidad para compartir.

Para utilizarlo se debe acceder a nuestra cuenta de Google y, o bien entrar directamente al enlace de Google Colab o ir a nuestro Google Drive.

2.3.3.1. Principales librerías

Keras: es una Api para entrenar redes neuronales de alto nivel con Deep Learning,

desarrollada bajo lenguaje de programación Python, se puede ejecutar sobre TensorFlow, CNTK o Theano. Keras permite implementar modelos complejos de Deep Learning solo se debe definir las capas que constituirán la red neuronal. En la detección de objetos, Keras trabaja con modelos que toman las imágenes como entrada y da probabilidades de clasificación como salida. (Keras Documentation, 2018)

TensorFlow: es una librería de python, desarrollada por Google, para realizar cálculos numéricos mediante diagramas de flujo de datos. Esto puede chocar un poco al principio, porque en vez de codificar un programa, codificaremos un grafo. Los nodos de este grafo serán operaciones matemáticas y las aristas representan los tensores (matrices de datos multidimensionales). Con esta computación basada en grafos, TensorFlow puede usarse para deep learning y otras aplicaciones de cálculo científico. (Quan et al, 2017)

Imgaug: es una librería para el aumento de imágenes en experimentos de aprendizaje automático. Admite una amplia gama de técnicas de aumento, permite combinar y ejecutar fácilmente en orden aleatorio o en múltiples núcleos de CPU, tiene una interfaz estocástica simple pero poderosa y no solo puede ampliar imágenes, sino también puntos de referencia, cuadros delimitadores, calor mapas y mapas de segmentación. (Christopher et al., 2006)

OpenCv: es una librería que tiene más de 2500 algoritmos, que incluye algoritmos de machine learning y de visión artificial para usar, estos algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, extraer modelos 3D, encontrar imágenes similares, reconocer escenarios, etc. (Cabrera et al, 2014)

2.4. Dataset

El dataset se basa en imágenes satelitales, las cuales fueron obtenidas de Google Earth, el cual, es de libre acceso, las imágenes fueron descargadas con una resolución de 1920x1080 pixeles y que pesan menos de 1 MB, las imágenes fueron guardadas en una carpeta, a la cual se le asignó el nombre ¨Proyecto¨, y dentro de esta la carpeta denominada ¨Images¨, en la cual se colocó el siguiente código para cada imagen ISGE_PS_G4_1 (Imagen Satelital Google Earth_Proyecto Software_Grupo4_ Número de la Imagen), teniendo un total de 400 imágenes.

Tabla 2. Características de las imágenes satelitales

Tamaño	≤ 1MB
Dimensiones	1920x1880
Resolución Horizontal	96 ppp
Resolución Vertical	96 ppp
Profundidad en bits	24

2.5. Procesamiento

Luego de tener el dataset con las imágenes satelitales ya guardadas en una carpeta, se prosigue a crear un archivo XML donde se anotará el objeto que se identifique en la imagen, sus posiciones x, y su alto y ancho. El XML será de este tipo:

<?xml version="1.0"?>
<annotation>
<folder>Imágenes Satelitáles</folder>
<filename>ISGE_PS_G4_1.jpg</filename>
<path>C:\Users\LENOVO CORE I5\Desktop\Imágenes Satelitáles\ISGE_PS_G4_1.jpg</path>
<source>
<database>Unknown</database>
</source>
<size>
<width>1920</width>
<height>1080</height>
<depth>3</depth>
</size>
<segmented>0</segmented>
<object>
<name>Falla</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>310</xmin>
<ymin>307</ymin>
<xmax>1382</xmax>
<ymax>790</ymax>
</bndbox>
</object>
</annotation>

Figura3. Modelo de archivo XML

Para la realización de los archivos XML se utilizó el programa de edición LabelImg, en este editor se selecciona una carpeta la cual es la fuente de la imagen, y siguiente una carpeta donde se guardarán los archivos XML, en el editor se crea una caja (bounding-box), sobre cada objeto que queremos detectar en la imagen en este caso la falla geológica y escribir su nombre (clase), el cual fue guardado como ¨Falla¨, cuando finalizamos damos guardar y proseguimos con la siguiente imagen de nuestra carpeta fuente.

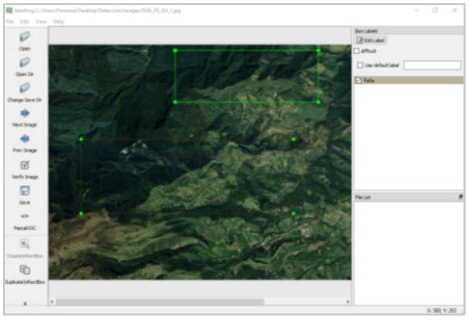


Figura 4. Edición en Labelimg

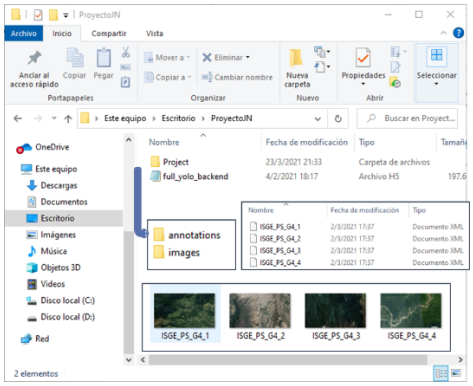


Figura5. Arquitectura de almacenamiento de datos

2.6. Creación del modelo

La CNN está compuesta por una capa de entrada, una capa de salida y muchas capas intermedias ocultas, estas van desde las primeras capas simples hasta llegar a capas profundas especializadas, las capas realizan operaciones que alteran los datos con el objetivo de aprender características específicas de dichos datos. Las 3 capas más frecuentes son: convolución, activación o ReLU, y pooling.

La convolución somete las imágenes de entrada a un conjunto de filtros convolucionales, cada uno de los cuales activa ciertas características de las imágenes. Para empezar con las convoluciones se toman grupos de pixeles cercanos de la imagen de entrada y se va operando matemáticamente (producto escalar) contra una matriz llamada kernel. Al tener una imagen a color el kernel es de 3x3x3, es decir: un filtro con 3 kernels de 3×3; luego esos 3 filtros se suman (y se le suma una unidad bias) y conforman 1 salida. La función de activación utilizada para esta red neuronal es ReLu por Rectifier Linear Unit y consiste en f(x)=max (0, x), esta se denomina activación, ya que solo las características activadas pasan a la siguiente capa. Las capas de pooling permiten reducir el peso de la representación para así agilizar el proceso de aprendizaje de la red neuronal, si el valor de pixeles es alto significa que la capa convolucional habrá encontrado un patrón, es así que el pooling asegura que los patrones detectados en la capa convolucional se mantengan en la siguiente capa de la red. Estas operaciones se repiten en decenas o cientos de

capas, de modo que cada capa aprende a identificar diferentes características.

Después de extraer las características en varias capas, la arquitectura de la CNN pasa a la detección. Aquí se utilizará las posiciones x, y, alto y ancho. Para ello se valdrá de Anclas, las cuales son ventanas de distintos tamaños, que servirán para hacer la detección.

La última capa de la red predice probabilidades de clases y coordenadas para las cajas de identificación; para este paso se normaliza la altura y ancho de la caja de identificación con respecto a los parámetros de la imagen, de tal manera que sus valores se mantengan entre 0 y 1.

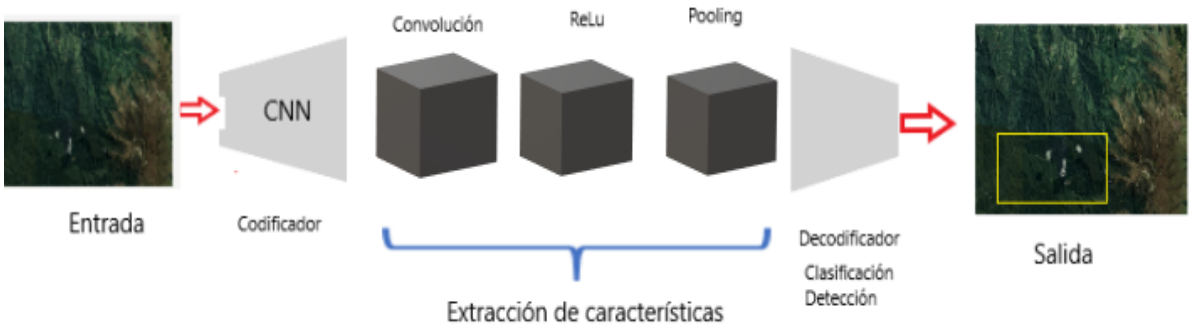


Figura6. Representación del modelo de detección

"Para refinar el modelo y que detecte los objetos se utilizó dos funciones con las cuales se descarta áreas vacías. Las funciones son:

IOU: Intersection Over Union, que nos da un porcentaje de acierto del área de predicción contra la "caja" real de predicción.

Non Maximum suppression: permite quedarse de entre nuestras 5 anclas, con la que mejor se ajusta al resultado. Al tener áreas diferentes propuestas que se superponen, de entre todas, nos quedamos con la mejor y eliminamos al resto." (Bagnato, 2020)

El entrenamiento busca tener un solo cuadro de identificación por objeto, lo cual se consigue a partir de las probabilidades predichas para cada cuadro, manteniendo el de mayor alta probabilidad.

Para la predicción se indica los parámetros y el número de épocas a utilizar, el último valor debe ir variando. Además, se debe ir guardando tanto las predicciones, como el error que está cometiendo. De esta manera se podrá visualizar cómo ha entrenado la red. A medida que avanza se observa cómo ha mejorado el error de la red en cada iteración.

2.6.1. Tabla de hiperparámetros

Los hiperparámetros son las variables que rigen el proceso de entrenamiento, parte de la configuración de una red neuronal profunda es decidir cuántas capas ocultas de nodos usar entre la capa de entrada y la de salida, y cuántos nodos debe usar cada capa.

Tabla3. Hiperparámetros del proyecto

HIPERPARÁMETRO	VALOR
CONV2D	Convolución en dos dimensiones
Número de filtros	32
Tamaño del filtro	3x3
Función de activación	ReLU
Pooling	2x2
Función	Sigmoidal

2.7. **Entrenamiento**

El entrenamiento de nuestra red se realizó mediante la implementación de dos modelos preentrenados para detección de objetos como son Yolov2 y Yolov3, cada uno considerado en un diferente entorno de programación.

Modelo de Jupyter notebook

El primer modelo se lo desarrollo en jupyter notebook mediante la creación de un ambiente de trabajo idóneo para su ejecución implementando las principales librerías como son tensorflow y keras, luego se procedió a direccionar la carpeta de los archivos del proyecto para aplicar la normalización y estandarización de las mismas.

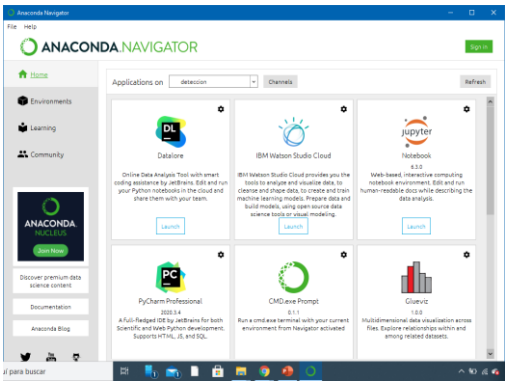


Figura7. Entornos del Software Anaconda Python

En la figura 8 se pueden apreciar los parámetros necesarios para realizar el entrenamiento de la red el cual consistió en aplicación de 10, 20 hasta 50 épocas con el objetivo de obtener un modelo de alta precisión para la detección.

Se procede a importar el modelo full-yolo-bancked.h5 que se encuentra dentro de los archivos proporcionados por el modelo preentrenado.

```
[ ]  yolo.train(train_imgs         = train_imgs,
                valid_imgs         = valid_imgs,
                train_times        = 6,
                valid_times        = 1,
                nb_epochs          = 20,
                learning_rate      = 1e-4,
                batch_size         = 8,
                warmup_epochs      = 2,
                object_scale       = 5,
                no_object_scale    = 1,
                coord_scale        = 1,
                class_scale        = 1,
                saved_weights_name = mejores_pesos,
                debug              = True)
```

Figura8. Entornos del Software Anaconda Python

Se obtuvo un archivo con los mejores pesos del entrenamiento, el cual fue evaluado mediante la aplicación de comandos a una nueva imagen.

```
mejores_pesos = "red_Falla.h5"

image_path = "C:/Users/Personal/Desktop/Proyecto/Imágenes test/ISGE_PS_G4_81.jpg

mi_yolo = YOLO(input_size          = tamanio,
               labels              = labels,
               max_box_per_image   = 5,
               anchors             = anchors)

mi_yolo.load_weights(mejores_pesos)

image = cv2.imread(image_path)
boxes = mi_yolo.predict(image)
image = draw_boxes(image, boxes, labels)

print('Detectados', len(boxes))

cv2.imwrite(image_path[:-4] + '_detected' + image_path[-4:], image)
```

Figura9. Evaluación del modelo, entorno d jupyter notebook

Modelo Google Colab

En el modelo implementado en este entorno consistió en conectarnos con nuestro One Drive

que posee las carpetas con las 400 imágenes y los 400 archivos xml generados, para una mayor facilidad de manejo de datos, posteriormente se instala el modelo de Yolo de ImageAI que es un detector y clasificador que posee todas las librerías necesarias para el modelo de detección, mediante una nueva celda ejecutamos un comando para traer los pesos predeterminados del modelo de Yolo generando el archivo pretrained-yolov3.h5

Se coloca la instrucción para la utilización del modelo, luego se establece el directorio de trabajo, el modelo y las épocas para la realización del entrenamiento.

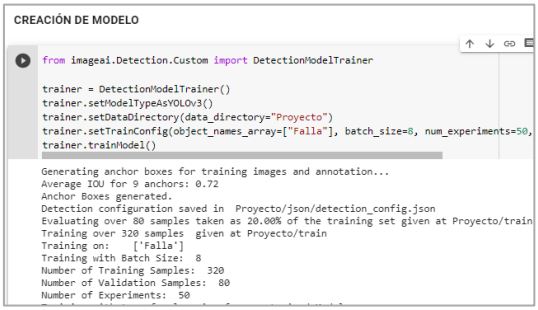


Figura10. Celda con instrucciones para entrenamiento en el entorno de Google colab.

Al ejecutar la celda se pueden apreciar los diferentes datos que genera el modelo como el número de muestras para validación y para entrenamiento, para este modelo se realizó pruebas con 20, 50 y 100 épocas para determinar el de mayor precisión.

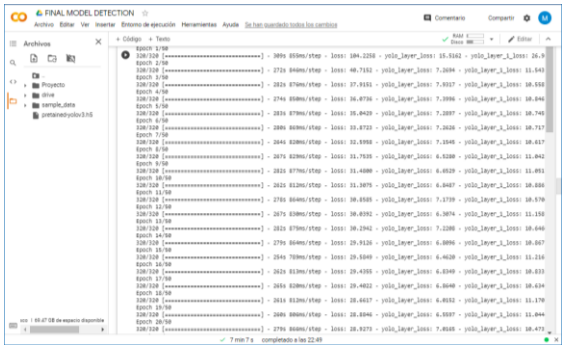


Figura11. Entrenamiento con 50 épocas

Al finalizar el entrenamiento que dura alrededor de tres horas en un entorno GPU, procedemos a realizar la validación del modelo mediante la obtención de sus métricas.

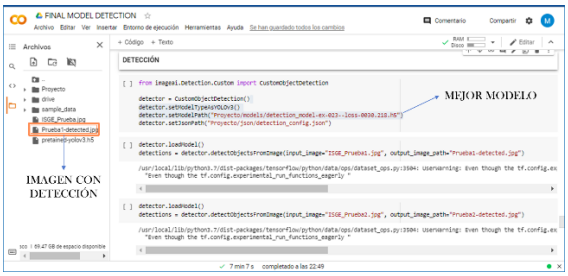


Figura 12. Entorno Google Colab con celdas para detección

2.7.1. Pruebas

Para el modelo de Jupyter notebook se realizaron tres pruebas de 20, 30 y 50 épocas, cada una de ellas con diferentes parámetros y con 160 imágenes de entrenamiento y 40 imágenes de validación.

En el modelo de Google Colab se realizaron 10 pruebas para su entrenamiento los cuales se almacenaron en la carpeta Colab notebook de nuestro One Drive, la división de datos fue 320 de entrenamiento y 80 de validación, cada una de estas pruebas consistió en incrementar el número de épocas y comprobar la validación del modelo mediante las graficas de perdida y precisión.

Es así que para cada uno de los modelos se varió los parámetros de predicción IoU, No maximun supression y thershold, que son los más importantes en la precisión mAP.

2.8. Predicción

Para realizar la predicción se indica los parámetros y el número de épocas. Al tener la

red entrenada, se usa la función pre-entrenamiento. Además, se debe ir guardando tanto las predicciones, como el error que está cometiendo. De esta manera se podrá visualizar cómo ha entrenado la red. A medida que avanza se observa cómo ha mejorado el error de la red en cada iteración.

En el modelo pre-entrenado de Colab con 50 épocas se utilizó los siguientes valores llegando obtener un porcentaje de validación del 75%, resultando así un modelo favorable para la detección.

```
Using IoU:  0.5
Using Object Threshold:  0.3
Using Non-Maximum Suppression:  0.5
```

```
Model File:  Proyecto/models/detection_model-ex-025--loss-0028.335.h5

Evaluation samples:  80
Using IoU:  0.5
Using Object Threshold:  0.3
Using Non-Maximum Suppression:  0.5
Falla: 0.1011
mAP: 0.1011
```

```
Using IoU:  0.2
Using Object Threshold:  0.1
Using Non-Maximum Suppression:  0.2
```

```
Model File:  Proyecto/models/detection_model-ex-014--loss-0029.900.h5

Evaluation samples:  80
Using IoU:  0.2
Using Object Threshold:  0.1
Using Non-Maximum Suppression:  0.2
Falla: 0.6439
mAP: 0.6439
```

```
Using IoU:  0.1
Using Object Threshold:  0.1
Using Non-Maximum Suppression:  0.1
```

```
Model File:  Proyecto/models/detection_model-ex-023--loss-0030.218.h5

Evaluation samples:  52
Using IoU:  0.1
Using Object Threshold:  0.1
Using Non-Maximum Suppression:  0.1
Falla: 0.7524
mAP: 0.7524
```

Figura13. Parámetros utilizados en la predicción para definir precisión

2.9. Detección

El modelo de detección devuelve no solo detecciones sino probabilidades de que dicha detección sea acertada. Fijando diferentes umbrales puede construirse una curva de precisión y exhaustividad, calculando los valores de cada uno para un umbral dado.

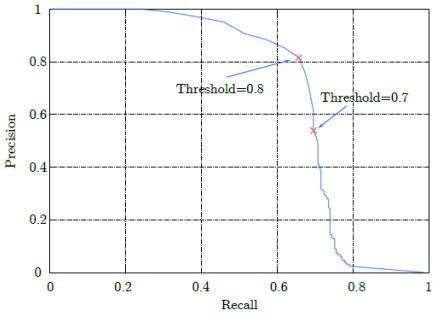
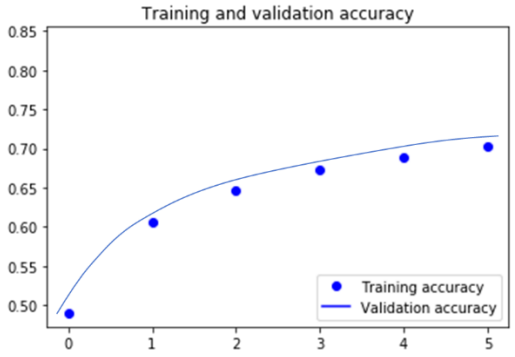
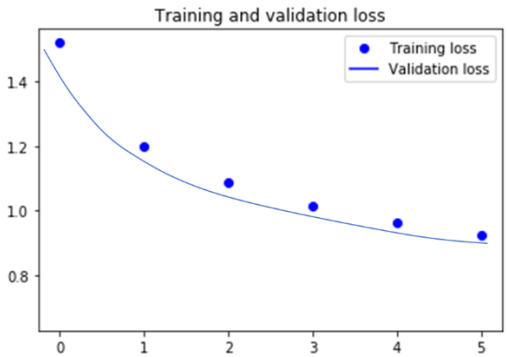


Figura14. Gráfica modelo de Precisión de umbrales

Con el modelo de detección se procedió a realizar las gráficas de precisión y perdida como se muestran a continuación.





Figuras15 y 16. Gráficas de pérdida y precisión.

3. Análisis estructural de la zona de estudio

A partir de la detección de fallas realizada a la zona de estudio se pudo establecer de forma regional la existencia de diferentes sistemas de fallas que mediante este modelamiento fueron detectadas por agrupamientos.

Figura17. Detección de fallas en la zona de estudio

Con base a lo descrito y la implementación de un modelo de Deep Learning podemos identificar estructuras y lineamientos con una orientación preferencial NW, en la zona este se apreciaron estructuras en sentido NE-SW, además se concluye que estas estructuras se encuentran asociadas a los complejos volcánicos de la zona y a los diversos drenajes que parten de los mismos.

4. Enlaces del proyecto

- Modelo Google Colab

https://colab.research.google.com/drive/1j8oDtOn9HGQh417meYUzXujYWFs3etXR#scrollTo=T4OgVEzf1Zlv

- Repositorio de imágenes y archivos

https://drive.google.com/drive/folders/1uKGhR3yu05Y1gkQ4XR_bqroPRXfQheWD?usp=sharing

- Pruebas realizadas

https://drive.google.com/drive/folders/160swkrM1Fdv05nmz_M0mz53lPGmWxqwC?usp=sharing

5. Conclusiones

Los datos etiquetados con precisión son esenciales para el aprendizaje automático exitoso, y la visión por computadora, por lo cual para la etiquetación de nuestras imágenes se utilizó el programa LabelImg para tener un mejor desempeño en la detección de objetos.

El número de épocas se debe ir aumentando progresivamente para reducir el porcentaje de error, pero se debe evitar el sobreentrenamiento del modelo, en este caso se puede llevar a que el modelo memorice las imágenes utilizadas y al presentar nuevas imágenes no las detecte de manera óptima.

La detección de fallas a partir de las redes convolucionales presenta una baja precisión para un análisis estructural a detalle debido a las múltiples consideraciones para establecer una falla como son su desplazamiento, el tipo de falla y su orientación, parámetros que no pueden ser señalados dentro de una red convolucional.

Es importante el identificar como el avance en las ciencias computacionales nos permiten reemplazar los métodos tradicionales en diversas ramas de la geología.

6. RECOMENDACIONES

La generación de modelos de aprendizaje automático depende de su arquitectura por lo que resulta recomendable realizar usar una separación de 80% de datos para entrenar y 20 % de datos para evaluar y validar el modelo.

A medida que se incrementa el número de imágenes, la red convolucional tendrá opción a detectar de forma más precisa las fallas dentro de una determinada área.

Resultaría factible la realización de un modelo con imágenes satelitales que contengan las mismas tonalidades, de igual forma es necesario el entrenamiento con imágenes de fallas representativas.

Además, se sugiere un modelo CNN de mask retainet de coco que permite delimitar la forma del objeto a detectar, que podría incrementar la precisión y el porcentaje de recuperación dentro de una imagen.

7. Referencias

- ARRIOLA I. "Detección de objetos basada en Deep Learning y aplicada a vehículos autónomos"
- Bagnato, J. (2020). Aprende Machine Learning. Available: https://www.aprendemachinelearning.com/deteccion-de-objetos-con-python-yolo-keras-tutorial
- Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006
- Espino C. (2017) *Análisis predictivo: técnicas y modelos utilizados y aplicaciones del mismo - herramientas Open Source que permiten su uso*. Page 12. Universitat Oberta de Catalunya.
- Boland, M.P., Pilatasig, L.F., Ibadango, C.E., McCourt, W.J., Aspden, J.A., Hughes, R.A., Beate, B., (2000). Geology of the West Cordillera between 0°-1°N, Proyecto de Desarrollo Minero y Control Ambiental, Programa de Información Cartográfica y Geológica. Informe No. 10. CODIGEM-BGS, 72p. Quito-Ecuador.
- Hughes, R.A., Pilatasig, L.F., (2002). Cretaceous and Tertiary terrane accretion in the Cordillera Occidental of the Andes of Ecuador.
- Cabrera E., Diaz E., (2014). Manual de uso de Jupyter Notebook para aplicaciones docentes. Universidad Complutense de Madrid.
- Keras, (2018), Keras Documentation. Available: https://keras.io/.
- Daniel Camilleri and Tony Prescott. Analysing the limitations of deep learning for developmental robotics. In Conference on Biomimetic and Biohybrid Systems, pages 86–94. Springer, 2017.
- Quan D., Cao Son T. y Chaudri J., (2017). Classification of Asthma Severity and Medication Using

TensorFlow and Multilevel Databases,» Procedia Computer Science.

- Sega M., Hantal G., B. Fabián y P., (2018). Jedlovszky, «Pytim: A python package for the interfacial analysis of molecular simulations,» Journal of Computational Chemistry,
- Telematica S.A (2019). “Cuando el Deep Learning se encuentra con el GIS”. Machine learning ha sido un componente central del análisis espacial en ARGIS.
- Tzutalin, (2015). LabelImg Git code. Available: https://github.com/tzutalin/labelImg
- Vásquez M. (2016) *Reconocimiento de Objetos usando Deep Learning.* Dep. Ingeniería de Sistemas y Automática Escuela Técnica Superior de Ingeniería Universidad de Sevilla. Pages 5-6