UNIVERSIDAD EUROPEA MIGUEL DE CERVANTES

ESCUELA POLITÉCNICA SUPERIOR

TITULACIÓN:
MÁSTER UNIVERSITARIO EN GESTIÓN Y ANÁLISIS DE GRANDES VOLÚMENES DE DATOS

TRABAJO FIN DE MÁSTER

DETECCIÓN DE HUMO Y FUEGO EN IMÁGENES

AUTOR

CELIA QUERO TORRES

TUTOR

IGNACIO GÓMEZ PÉREZ

VALLADOLID, JULIO

Índice

1.Objetivos del trabajo

En este proyecto llevaremos a cabo la detección de fuego y humo a través de un dataset compuesto por fotografías mediante redes neuronales implementadas en Python.

Los objetivos que se han planteado en este proyecto son:

1. Estudiar el estado actual de la detección automática de incendios forestales. Entender cómo funcionan las redes neuronales, qué técnicas se utilizan para tratar el dataset y si su aplicación es satisfactoria o no.

2. Realizar una profunda exploración en el preprocesado de imágenes mediante filtros y máscaras. Plantear las bases de un algoritmo que sea capaz de detectar incendios forestales: basado en los algoritmos estudiados y en los conocimientos de procesado de imagen, proponiendo un algoritmo que sea capaz de detectar incendios en la naturaleza.

3. Testear el algoritmo realizando los ajustes pertinentes para obtener buenos resultados en detección de incendios forestales. Analizar los resultados obtenidos aplicando distintas métricas.

4. Proponer mejoras sobre los algoritmos planteados para que el sistema de detección pueda ser mejorado por terceros.

2.Introducción a las redes neuronales.

2.1 La neurona biológica

Las redes neuronales intentan imitar el funcionamiento del cerebro humano, haciendo uso de las neuronas como las mínimas unidades de trabajo.

Para entender la estructura del cerebro es necesario presentar la anatomía de una neurona:

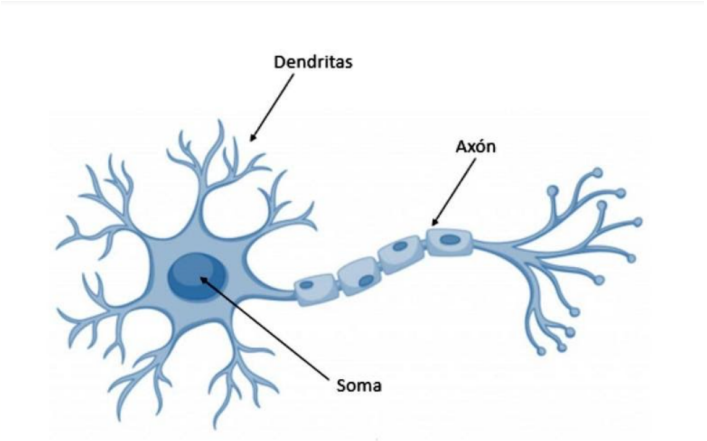


Figura 1. Partes de la neurona biológica

El papel que desempeñan los neurotransmisores es fundamental en el aprendizaje y almacenamiento de conocimientos.

Estos son los encargados de amplificar o disminuir la cantidad de potencial que se transmite de una neurona a otra, y es por este motivo por el que existen sinapsis inhibidoras y sinapsis excitadoras.

Queda reflejado que el flujo de información (determinado por los neurotransmisores) accede a la neurona mediante las dendritas, para acabar en el soma, donde es procesada y enviada a otra neurona mediante las dendritas de la misma.

2.2 La neurona artificial y el aprendizaje hebbiano

El primer algoritmo que presentaba una red neuronal simple se llamó Perceptrón, creado por Frank Rosenblatt en 1958, basado en el trabajo realizado previamente por Santiago Ramón y Cajal y Charles Scott Sherrintong (pioneros en el estudio del funcionamiento del cerebro humano).

En la siguiente imagen podemos observar la estructura del perceptrón:

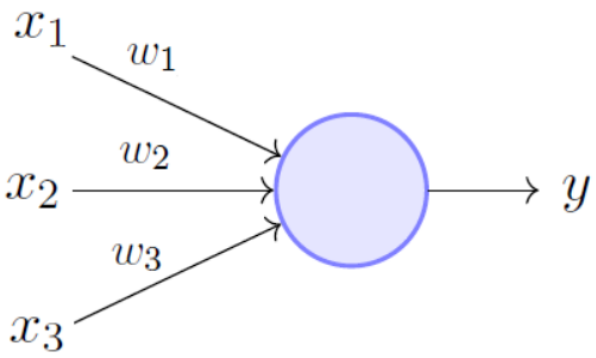


Figura 2. Representación de la estructura del perceptrón.

Glosario de los elementos que se presentan en la Figura 2:

- x_i representa la neurona i-esima.

- w_i representa el valor de inhibición o excitación entre la neurona x_i y la neurona que vemos en la imagen.

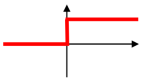
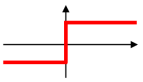
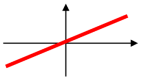
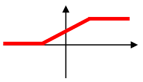
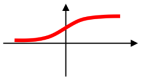
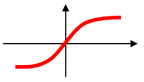
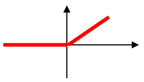
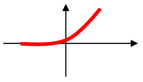
- y representa el resultado generado por la neurona i (equivalente a la salida del axón que se produce en la neurona biológica).

- La circunferencia azul representa el núcleo de la neurona, donde se calcula la suma de cada resultado que llega por parte de cada neurona multiplicado por su respectivo valor de inhibición/excitación. Lo denotaremos como $Net_i = \sum_j y_j w_{ji}$.

A dicha suma se le aplica la función de salida o transferencia. Nótese que carece de

sentido hacer uso de funciones lineales, debido a que en este caso caeríamos en la linealidad del cómputo, pudiendo reducir todo el procedimiento a la implementación en una sola neurona.

Es decir, para que la red neuronal pueda realizar un cómputo adecuado es necesario establecer una relación no lineal entre la entrada de cada neurona y su salida. En la Figura 3 presentamos las funciones más usadas en redes neuronales:

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)

Figura 3. Principales funciones de activación usadas en redes neuronales.

2.3 El aprendizaje Hebbiano

El proceso de aprendizaje consiste en variar los valores de los pesos w_i siguiendo unas pautas establecidas.

En el caso de que $w_i = 0$, se considera que la conexión no aporta ningún tipo de información.

Por otra parte, diremos que la red neuronal ha aprendido a la perfección cuando se han determinado los valores ideales para los w_i, de tal forma que por cada entrada que se le presenta nos proporciona el resultado esperado.

Se considera que el aprendizaje ha finalizado cuando se tiene que $\frac{dw_{ij}}{dt} = 0$.

Podemos esquematizar el proceso de aprendizaje como sigue:

1. Se proporcionan unos datos de entrada a la red neuronal.

2. Se obtiene la salida de la red neuronal a partir de dicha información proporcionada. A la salida esperada se le denomina etiquetas. Este aprendizaje, requiere de un dataset etiquetado.

3. Se compara la salida obtenida con la esperada.

4. Si existe un error significa que hay que variar los pesos para modificar la red neuronal para aproximarla más al objetivo deseado.

5. El proceso se repite hasta que se considera aceptable la diferencia entre las salidas que se obtienen y las esperadas.

"En el aprendizaje supervisado por corrección de error se ajustan los pesos en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error producido en la salida."(Sancho, 2021a).

A este tipo de aprendizaje se le denomina Aprendizaje Hebbiano, y se implementa con la siguiente Regla Delta:

$$\Delta w_{ji} = \alpha \cdot y_j(d_i - y_i)$$

donde:

- d_i es el valor de salida esperado de la neurona número i.

- y_i es el valor de salida de la neurona i.

- α es el factor de aprendizaje. Indica la escala que usaremos para variar los pesos.

2.4 El Perceptrón.

El caso del perceptrón es la red neuronal más sencilla, pues sólo consta de una única neurona. Por lo que sólo hay una salida de resultado.

Dicha neurona admite un número ilimitado de entradas.

La fórmula matemática con la que se calcula la salida es la siguiente:

$$y = f\left[\sum_{i=1}^{N} x_i w_i - \theta\right]$$

El parámetro θ se trata de una variable que nos permite añadir un grado de libertad. Para entenderlo con mayor exactitud expondremos un caso particular:

Tomemos como función f la conocida como Función Escalón de Heaviside:

$$\forall x \in \mathbb{R}: \quad u(x) = H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$$

Figura 4. Definición de la función Escalón de Heaviside.

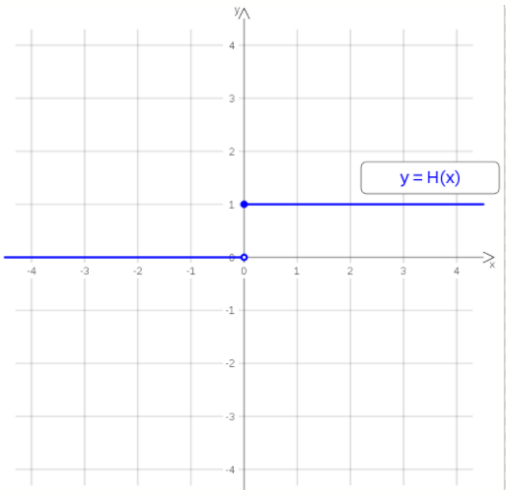


Figura 5. Representación de la función Escalón de Heaviside.

Supongamos que $y = 0$. Esto nos lleva la siguiente ecuación de la recta:

$$x_2 = -\frac{w_i}{w_2}x_1 + \frac{\theta}{w_2}.$$

Por lo que si variamos el coeficiente $\frac{w_i}{w_2}$podremos regular la pendiente de la recta y si alteramos el parámetro $\frac{\theta}{w_2}$modificaremos el punto de corte de la recta con el eje y.De esta forma tenemos la siguiente situación:

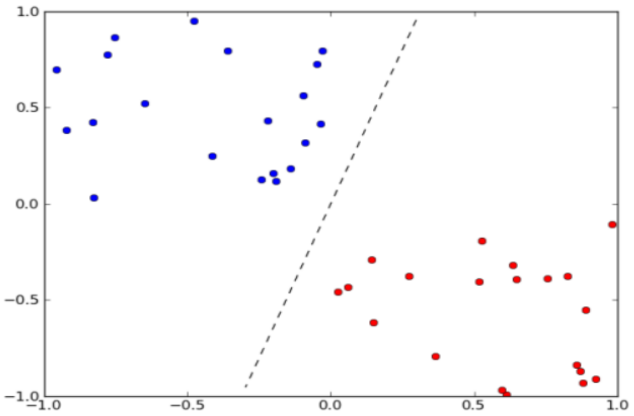


Figura 6. Representación de la separación realizada por el perceptrón en un dataset.

2.5 Redes multicapa y el algoritmo Back Propagation.

Debido a que el perceptrón sólo permite realizar separaciones lineales, es necesario crear redes neuronales con mayor número de neuronas, más semejantes a la que se muestra en la Figura 7. Estas redes conseguirán clasificar muestras que no cumplen la condición de separabilidad lineal.

En este caso observamos una red neuronal más elaborada. Nótese que cuantas más neuronas contenga la capa oculta, más compleja debe ser la frontera de separación entre las clases.

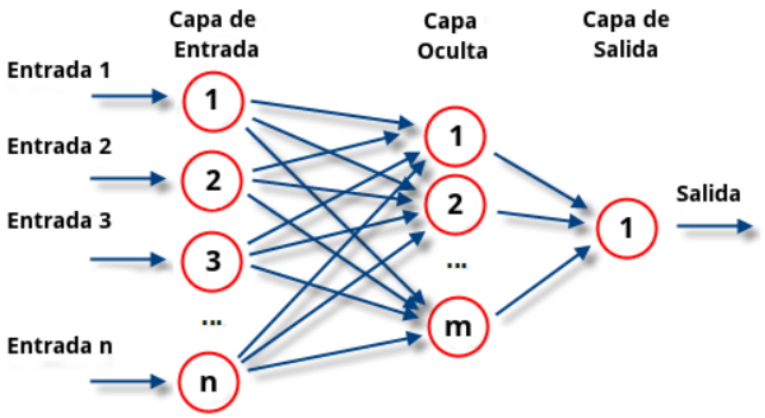


Figura 7. Representación de la arquitectura de una red neuronal.

En la mayoría de los casos se añaden varias capas ocultas para poder realizar separaciones más complejas.

En 1961, Rosenblatt desarrolló el perceptrón diseñando un modelo neuronal usando únicamente dos capas de neuronas, sin embargo no fue hasta el año 1986 en el que Rumelhart, Hinton y Williams idearon un método para que una red neuronal "aprendiera" dichas relaciones entre las muestras proporcionadas y sus correspondientes clases.

A este algoritmo se le denominó Back Propagation, es decir, el algoritmo de propagación hacia atrás o retropropagación de gradiente.

La idea general es la misma que la del perceptrón: comparar las salidas obtenidas con las salidas esperadas(etiquetas) y ajustar los pesos de tal manera que la siguiente vez que se presente el mismo patrón de entrada, la red produzca un resultado más cercano al deseado.

A este método se le denomina Regla Delta Generalizada, para ello se necesitarán funciones de activación f que sean no lineales, monótonas crecientes y derivables.

Partamos de la Regla Delta empleada en el perceptrón:

$$\Delta w_{ij} = \alpha \cdot y_j (d_i - y_i)$$

Definimos como δ_i el error que se produce en la neurona u_i:

$$\delta_i = d_i - y_i$$

En el caso de la Regla Delta Generalizada, cuando u_i sea una unidad de salida definiremos δ_i mediante la siguiente expresión:

$$\delta_i = (d_i - y_i) f'(Net_i)$$

En el caso de que esto no sea así, el error se produce en las neuronas que reciben como entrada la salida de u_i. Y en ese caso,cuando la neurona pertenece a una capa intermedia, seguiremos la siguiente definición:

$$\delta_i = f'(Net_i) \sum_k w_{ik} \delta_k$$

La idea principal del algoritmo Back Propagation, tal y como indica su nombre, consiste en la obtención de errores empezando por la capa de salida y retrocediendo hasta llegar a la capa de entrada. Una vez calculado el error δ_i podremos modificar el valor de los pesos w_{ij} teniendo en cuenta que:

$$\Delta w_{ij} = \alpha \cdot y_j \cdot \delta_i$$

Este algoritmo de propagación hacia atrás consta de los siguientes pasos:

1- Inicializar los pesos con valores aleatorios pequeños.

2- Tomar una muestra de entrada, llamada x, y escogida aleatoriamente.

3- Propagar la señal hacia adelante a través de toda la red neuronal.

4-Para calcular los errores que se producen en la capa de salida, emplearemos las siguientes fórmulas:

$\delta_i = f'(Net_i)(d_i - y_i)$ (si la neurona pertenece a la capa de salida)

$\delta_i = f'(Net_i) \sum_k w_{ik}\delta_k$ (si la neurona no pertenece a la capa de salida, es decir, pertenece a una capa intermedia)

5- Calcular los errores pertenecientes a la capa anterior hasta llegar a la capa de entrada.

6- Actualizar los pesos, empleando:

$$\Delta w_{ji} = \alpha \cdot y_j \cdot \delta_i$$

7- Repetir el procedimiento desde el paso 2 hasta lograr que el error sea menor a un umbral previamente establecido o hasta que se haya alcanzado un número fijado de

iteraciones(epochs).

En el caso de una dimensión, en la que sólo disponemos de una neurona en la capa de entrada, el Algoritmo del Gradiente Descendente nos permite encontrar el mínimo de la función f dentro de la superficie de error.

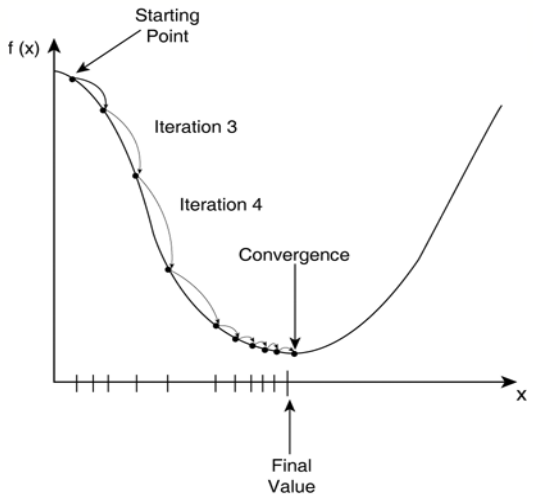


Figura 8. Representación de la superficie de error.

Cada avance que se produce viene dado por el gradiente negativo, que poco a poco nos acerca a la solución, es decir, el mínimo de la función.

Cuando la red es más elaborada y consta de mayor número de neuronas, la superficie crece en dimensión y en complejidad, pudiendo existir multitud de máximos y mínimos relativos que hacen imposible la convergencia del algoritmo.

Cada punto de la superficie de error corresponde a un conjunto de valores de los pesos de la red. Mediante el algoritmo se pretende descender por la superficie de error, por lo que quizás el algoritmo finalice en un mínimo local y no global.

El factor de aprendizaje α es un parámetro regulador del tamaño del paso, por lo que cuanto mayor sea este valor más grandes serán los pasos de avance para la localización del mínimo. Sin embargo esto puede provocar que el algoritmo nunca

converja, debido a las oscilaciones provocadas por un valor de α demasiado elevado.

3.Análisis de la situación.

La detección automática de incendios forestales ha sido un tema de interés desde que se dispone de herramientas tecnológicas que permiten llevarla a cabo, cada vez con más precisión. Además es una cuestión que repercute tanto en el ámbito urbano como en el rural, ya que mediante una detección precoz podremos salvar vidas humanas y de animales, bienes materiales y parajes naturales.

Países como España, Grecia o Italia son ejemplos de clima mediterráneo. En estas regiones la pluviosidad es escasa y las temperaturas son suaves en invierno y cálidas en verano. La vegetación resultante del estrés hídrico es abundante en matorrales y con árboles no muy altos pero robustos como la encina o el pino. Debido a la escasez de lluvia, el clima mediterráneo es muy susceptible de padecer incendios forestales, por ejemplo, en España se queman de media 113.847 hectáreas de masa forestal al año debido a incendios forestales.

La fumarada en entornos forestales acostumbra a verse desde largas distancias (a más de 100 m). Este humo tiene un comportamiento algo diferente que el humo cercano, su movimiento es lento, más denso, etc. Por esto existen algunos algoritmos dedicados específicamente al caso. En detección de humos forestales las falsas alarmas son producidas especialmente por nubes y sombras.

El objetivo es detectar incendios forestales mediante cámaras situadas sobre el terreno y técnicas de procesado de imagen. Actualmente existen dispositivos que permiten vigilar más de 15 mil hectáreas con un único sistema. Esta detección se puede hacer en base a la identificación del fuego o del humo, ambos indicativos de los incendios forestales. Ambos son dos elementos con características distintas: mientras que el fuego tiene colores vivos y se mantiene siempre en un mismo sitio pero es muy variante; el humo tiene unos colores muy apagados y tiende a moverse expandiéndose por el aire; así la detección debe ser independiente para ambos elementos.

En ocasiones, esta detección también se lleva a cabo mediante sensores de partículas de humo, y aunque esta alternativa nos puede servir para una confirmación del

posible incendio, no nos proporciona datos sobre el origen o localización del mismo.

Por otra parte, es fundamental el reconocimiento precoz del mismo.

En la Figura 9 mostramos una gráfica que refleja la relación exponencial existente entre el área quemada(medida en Ha) en función del tiempo transcurrido .

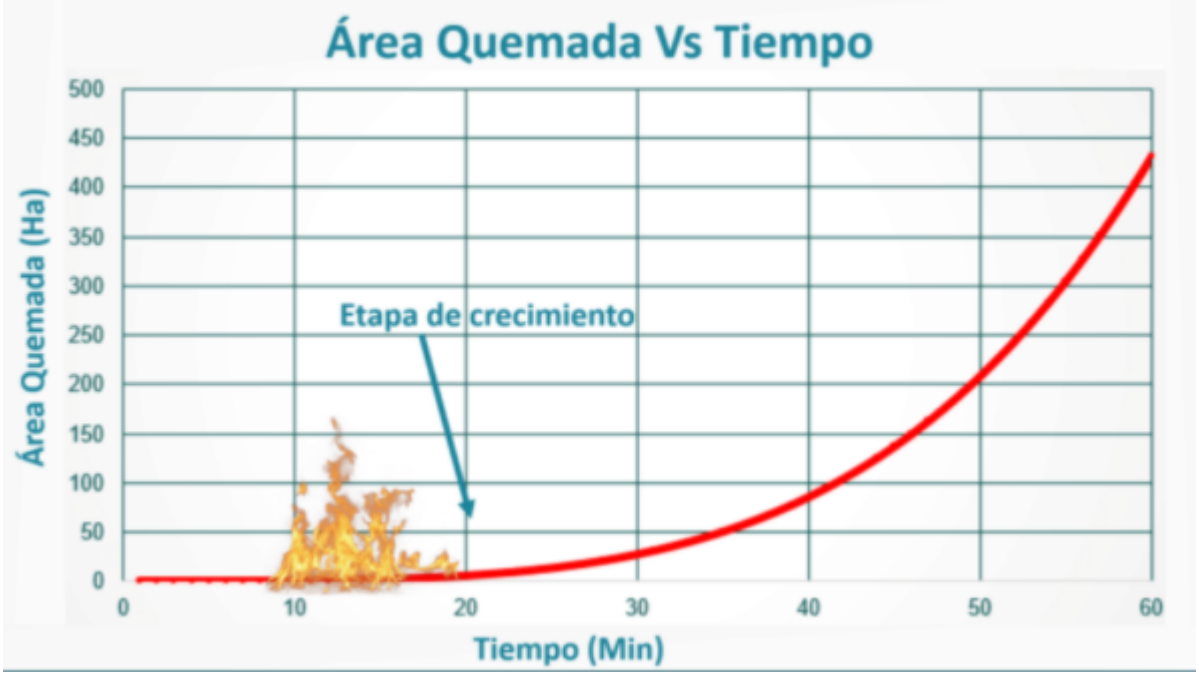


Figura 9. Gráfico en el que se refleja el área quemada en función del tiempo transcurrido.

Esta gráfica demuestra como el momento crucial transcurre en los primeros 20-30min desde el inicio del incendio, siendo este un punto crítico en la etapa de crecimiento del incendio en cuanto al área quemada.

Algunos de los problemas a afrontar son las diferentes características de la cámara instalada, el manejo de los diferentes puntos de vista obtenidos o evitar las distintas situaciones lumínicas que puedan inducir a error.

En el año 2019 se produjo un gran avance en este campo, IBM lanzó en España un modelo de Inteligencia Artificial, desarrollada por Compta e IBM Watson, que permite detectar y combatir los incendios forestales de una manera más ágil. El algoritmo se conoce como Bee2FireDetection, y es capaz de calcular la probabilidad de que se produzca un incendio

antes de que ocurra haciendo uso de espectrómetros, cámaras digitales de video vigilancia y termografía (una técnica que permite analizar temperaturas a distancia) y por supuesto de técnicas de Deep Learning para procesar toda esta información y detectar incendios forestales a distancias de hasta 15 kilómetros durante 24 horas al día los 365 días del año.

4.Obtención, procesado y almacenamiento de datos.

4.1 Fire Images Dataset

Este dataset fue creado por el grupo de trabajo constituido por Ahmed Gamaleldin, Ahmed Atef, Heba Saker, Ahmed Shaheen durante el reto "NASA Space Apps" en el año 2018.

Esta recopilación de imágenes tiene como objetivo entrenar un modelo capaz de distinguir aquellas fotografías que contienen fuego de otras que no lo tengan.

Dado que este problema trata sobre la clasificación de imágenes en dos tipos o clases, el dataset se encuentra dividido en dos carpetas que contienen fotografías ordenadas aleatoriamente:

- fireimages: Contiene 755 imágenes de incendios forestales, donde algunas contienen gran cantidad de humo.
- non-fireimages: Contiene 244 imágenes captadas en la naturaleza donde encontramos distintos elementos; como cascadas, lagos, puestas de sol, ríos, distintos tipos de bosques, etc.

El nombre que recibe cada imagen está compuesto por el etiquetado que deseamos obtener("fire" o "non_fire"), el número de imágen y el formato en el que se encuentran ("png"). El peso del dataset es de 390 MB.

Podemos encontrar el enlace de descarga para la obtención del dataset de dominio público en (*FIRE Dataset*, 2020).

4.2 Almacenamiento y carga del dataset.

Una vez descargado del dataset de (*FIRE Dataset*, 2020), lo cargamos en google drive e implementamos el siguiente código para poder usarlo.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[ ] import os
    os.chdir("/content/drive/My Drive/TFM(BIG DATA)")
    !pwd

    /content/drive/My Drive/TFM(BIG DATA)
```

Figura 10. Código necesario para cargar archivos desde google drive.

Seguidamente especificamos las rutas donde se encuentran alojados los archivos, comprobamos que se han cargado correctamente. Efectivamente el dataset está compuesto por 755 imágenes donde aparecen incendios y 244 fotografías sin fuego.

El peso del dataset es de 390 MG. Las imágenes inicialmente presentan distintas dimensiones, incluso hay algunas que se han realizado horizontalmente y otras verticalmente. Todas las fotografías son a color.

5.Análisis exploratorio.

Para realizar el análisis exploratorio tomaremos una imagen presente en el dataset original, presente en el fichero de imágenes con fuego.

Figura 11. Imagen original tomada del dataset.

Comenzamos imprimiendo en pantalla la imagen, y en un 3-upla mostramos las dimensiones de la misma y el número de canales que presenta, en este caso 3 (RGB) debido a que es una fotografía a color.

Definimos los distintos kernels para aplicar a la imagen seleccionada. De esta manera podremos calcular algunos de los estadísticos principales, como la media y mediana. El tamaño que toma el kernel es de 3x3 píxeles, puesto que las imágenes presentan un tamaño medio. Hemos de tener en cuenta el tamaño de las fotografías que componen el dataset para elegir un tamaño adecuado.

Cuanto mayor sea el kernel mayor información tomaremos del entorno que rodea a dicho píxel, pero el tiempo de procesamiento se verá incrementado.

Sin embargo, un tamaño del kernel de 2x2 píxeles puede ser insuficiente puesto que no toma la suficiente información del entorno de dicho píxel.

En particular, definimos cuatro filtros: media, mediana, mínimo y Sobel (potenciador de bordes).

Figura 12. Imagen resultante tras aplicar el filtro media

Figura 13. Imagen resultante al aplicar el filtro mediana

Figura 14. Imagen resultante al aplicar el filtro de mínimo

Figura 15. Imagen resultante al aplicar el filtro Sobel.

Observamos que con el filtro Sobel conseguimos marcar las líneas de contorno, sin embargo no se aprecia si esto será o no positivo puesto que los parajes naturales presentan numerosas formas irregulares y semejantes al fuego y humo provocados por el incendio.

Por ello, llevamos la imagen a tonalidad de grises para estudiar con mayor precisión los

filtros “realzadores” de bordes: Sobel, Roberts y Prewitt. De esta forma apreciamos con mayor facilidad los bordes detectados en la fotografía evadiendo las tonalidades presentadas en la fotografía.

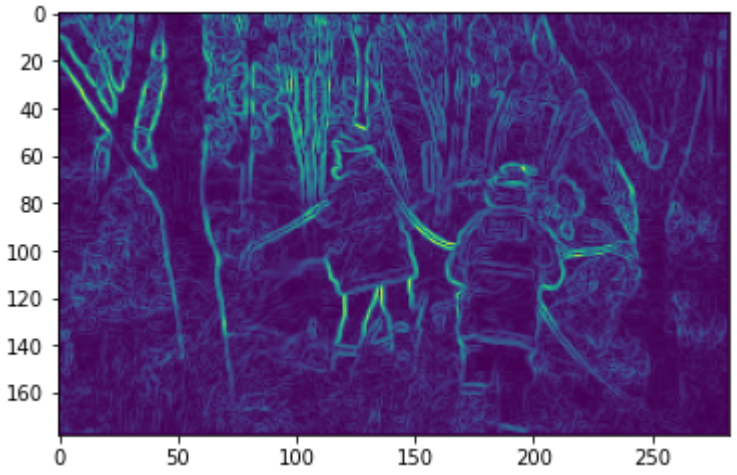


Figura 16. Imagen, en tonalidad de grises, después de aplicar el filtro Sobel.

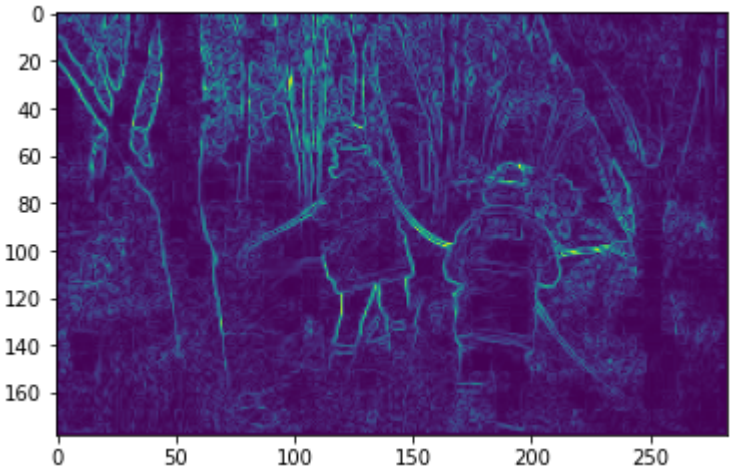


Figura 17. Imagen, en tonalidad de grises, después de aplicar el filtro Roberts.

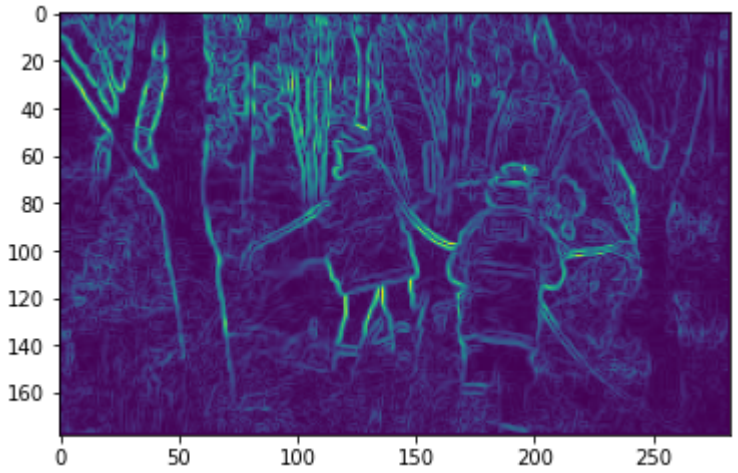


Figura 18. Imagen, en tonalidad de grises, después de aplicar el filtro Prewitt.

Observamos como ninguno de los anteriores filtros facilita la identificación de fuego o humo en dicha fotografía. En particular, apenas se distinguen los bordes que delimitan el humo y fuego de los que delimitan los matorrales.

Por lo que concluimos que la aplicación previa al dataset de estos filtros no favorecen la detección precoz de un incendio.

Continuaremos con la definición de una máscara caracterizada por realzar aquellas componentes conexas que se encuentren por encima de la media de los todos valores de los píxeles que componen la imagen.

Recordemos que las máscaras son filtros geométricos de una imagen. Por ejemplo, si queremos seleccionar una región de una imagen, podemos hacerlo multiplicando la matriz de la imagen original por una matriz de igual tamaño que contenga unos en la región que queremos conservar y ceros en el resto.

En la Figura 19 podemos comprobar el resultado obtenido una vez aplicada la máscara a la Figura 11.

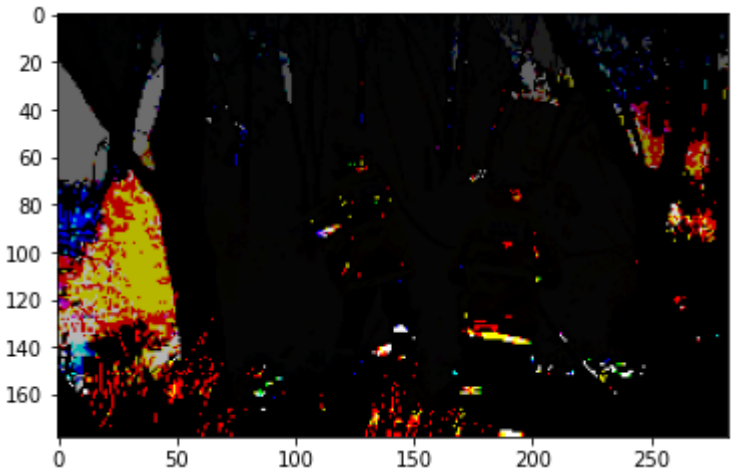


Figura 19. Imagen resultante al aplicar la máscara realzadora de componentes conexas superiores a la media.

Observamos que este tipo de pre-procesamiento de imágenes podría favorecer el aprendizaje del modelo, ya que destaca los píxeles implicados en el incendio.

Probemos con un posible contraejemplo para asegurar que no existen casos en los que se realcen zonas susceptibles de ser confundidas con un incendio.

Realizaremos las pruebas del posible contraejemplo con la fotografía de la Figura 20.

Figura 20. Fotografía original de un paisaje con bruma y hojas de color cobrizo.

Esta imagen se encuentra en el dataset Fire Dataset, y forma parte del conjunto de fotografías que no presentan incendios. Se trata de un paisaje no muy habitual, en el que la bruma y los colores presentes en la región inferior derecha de la figura pueden generar cierta confusión.

Aplicamos la máscara realzadora de las componentes conexas superiores a la media, para generar la Figura 21.

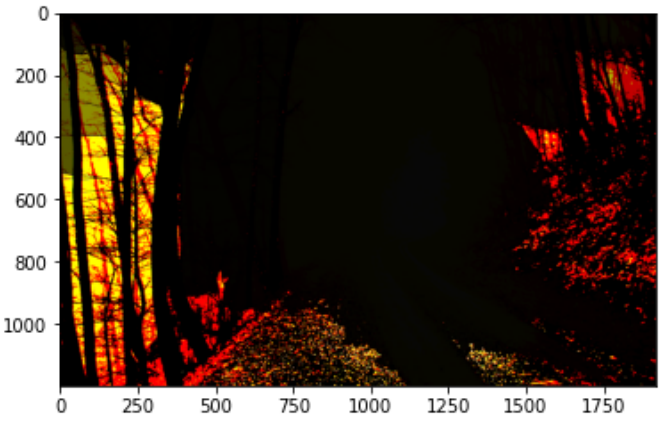


Figura 21. Fotografía resultante tras aplicar a la Figura 20 la máscara realzadora de componentes conexas superiores a la media.

Queda testado que esta máscara puede generar confusiones en el modelo, ya que realza con el mismo grado y tonalidades ciertas zonas de imágenes que no presentan incendios.

A continuación, probaremos a realizar correcciones de los parámetros gain y gamma, estrechamente relacionados con la luminosidad de la fotografía, para obtener los valores que nos permitan resaltar el fuego frente a los demás elementos que puedan aparecer en la imagen.

Dado que hemos de probar con múltiples combinaciones y valores de los mismos, crearemos dos sliders, en los que podremos dar valores a ambos parámetros con saltos de 0.1 unidades.

Figura 22. Sliders creados para la corrección de los posibles valores de los parámetros gain y gamma.

Primeramente aplicaremos dichas modificaciones a la Figura 11, imagen con incendio.

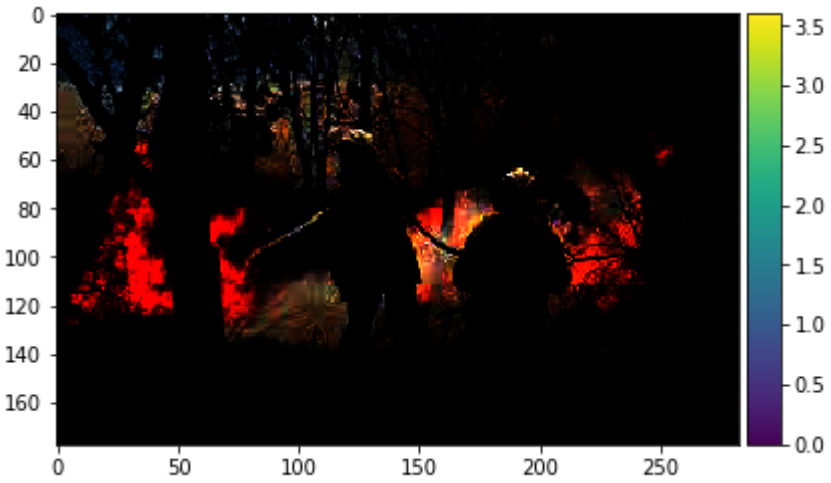


Figura 23. Imagen resultante al modificar valores de los parámetros gain y gamma en la Figura 11.

Nótese que mediante esta corrección obtenemos unos resultados favorables para aplicar en el tratamiento previo del dataset.

Al igual que hicimos con el diseño de la anterior máscara, probaremos si existe algún posible contraejemplo que pueda inducir a error a nuestro modelo.

Para esta comprobación usaremos la misma imagen utilizada anteriormente, la Figura 20.

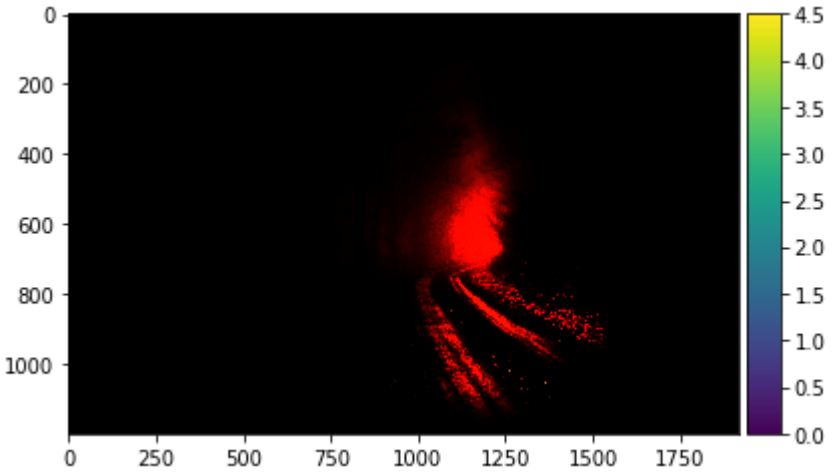


Figura 24. Resultado obtenido una vez realizada la misma corrección de los parámetros gain y gamma en la Figura 20.

Nótese cómo este método vuelve a propiciar errores en el dataset. En la Figura 24 se aprecia la misma gama de colores que la obtenida en la Figura 23, presentando sólamente una de ellas un incendio.

Por lo tanto este tratamiento de imágenes no es el óptimo para el dataset Fire.

Finalmente, construimos la máscara Sharpen. Esta será la composición de varias máscaras definidas previamente. Partiremos de nuevo de la imagen de referencia hasta ahora, la Figura 11.

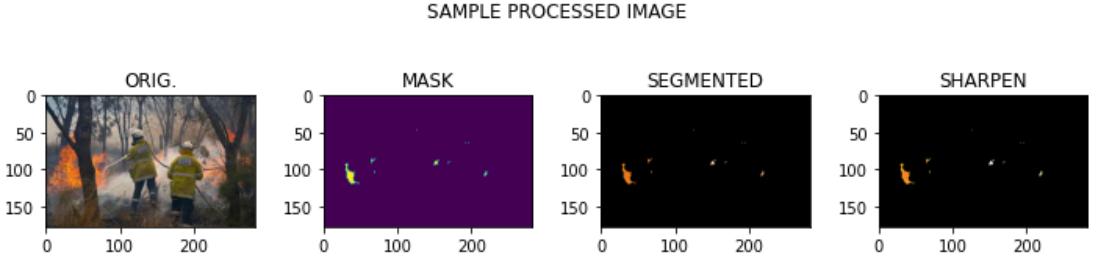


Figura 25. Conjunto de máscaras aplicadas a la imagen original hasta llegar a la máscara Sharpen.

Nótese como con la máscara Sharpen obtenemos los resultados deseados.

Sin embargo, falta probar estos resultados en la Figura 20. Con ella observaremos si esta máscara es propicia a generar resultados similares en imágenes que capturan algún fuego.

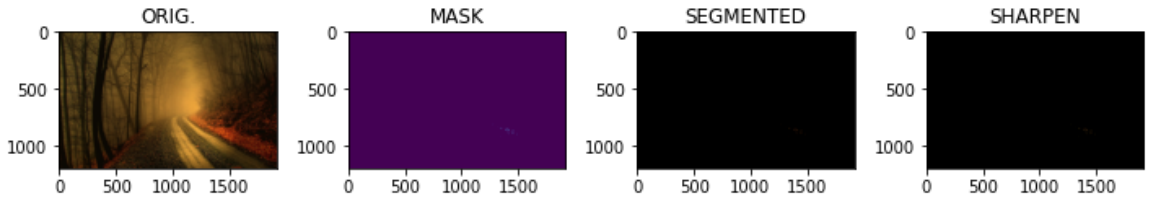


Figura 26. Resultados obtenidos aplicando a la Figura 20 un conjunto de filtros hasta llegar a la máscara Sharpen.

Observamos como esta máscara no realza, al menos en la Figura 20, los píxeles que presentan tonalidades similares a las de un incendio. Por lo que todo apunta a que se trata de un procedimiento que resulta ser un buen candidato para aplicar al pre-procesamiento del dataset.

Para asegurarnos de la fiabilidad de esta máscara he realizado un profundo análisis del dataset para seleccionar y extraer aquellas fotografías que, aparentemente, son candidatas a generar una posible confusión en la clasificación de las mismas.

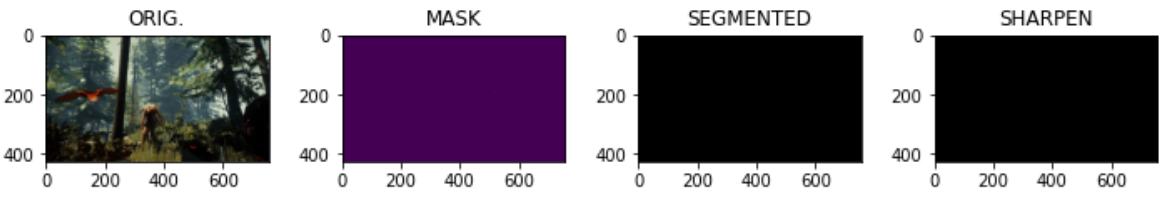


Figura 26. Máscaras aplicadas a la imagen original hasta llegar a la máscara Sharpen.

Una vez realizado este análisis exploratorio basado, principalmente, en la aplicación de distintos filtros y máscaras diseñadas para analizar las múltiples características que presenta el dataset, optamos por aplicar la máscara Sharpen a todas las imágenes(tanto a las de entrenamiento como las de testeo).

Gracias a la máscara Sharpen conseguimos potenciar la silueta, colores y brillos tan característicos que presenta el fuego, ennegreciendo el entorno que lo rodea.

Además, comprobamos que distingue casos especialmente inciertos como por ejemplo colores brillantes y anaranjados que presentan algunos plumajes de aves, hojas con tonalidades cobrizas, ambientes con niebla, etc.

6.Diseño e implementación de los modelos o técnicas necesarias.

Para conseguir un manejo más sencillo del dataset comenzaremos creando un dataframe que constará de 999 filas, es decir, tantas filas como imágenes tenemos el dataset. Serán tres las columnas presentes, la primera contiene el nombre de la fotografía, la segunda recogerá su id, es decir la codificación 0 para aquellas en las que aparezca un incendio y 1 para las restantes, y la tercera columna guardará la etiqueta, sin codificar, coincidente con el nombre de la carpeta a la que pertenecen.

	file	id	label
0	fire_images/fire.100.png	0	fire_images
1	fire_images/fire.10.png	0	fire_images
2	fire_images/fire.1.png	0	fire_images
3	fire_images/fire.101.png	0	fire_images
4	fire_images/fire.102.png	0	fire_images

Figura 27. Vista de las primeras filas del data frame creado.

Tal y como hemos explicado anteriormente, aplicaremos la máscara Sharpen a las 999 imágenes que componen el dataset. Este procedimiento es complejo, por lo que se demora unos cuantos minutos.

Una vez aplicado este pre-procesamiento del dataset, lo dividimos en un 80% destinado al entrenamiento y el 20% restante a testeo.

Para el diseño de la red neuronal aplicaremos Transfer Learning o Aprendizaje por Transferencia.

Esta técnica es una vía muy eficiente para conseguir una red neuronal destinada a

resolver un problema concreto a partir de una red ya entrenada para resolver un problema parecido.

De esta forma, aprovechamos los pesos de la red ya aprendidos durante el entrenamiento de la misma durante la resolución de problemas con cierta similitud al que presentamos. Generalmente se hace uso de redes cuya arquitectura ha sido diseñada por expertos, logrando obtener resultados destacados en la historia de las redes neuronales.

Otra de las ventajas es que el coste computacional para hacer uso de dichas redes es habitualmente muy reducido, unos cuantos minutos, sin necesidad de disponer grandes recursos o dispositivos informáticos para ello.

Algunas de las limitaciones que presenta esta técnica es que generalmente el tamaño de los datos de entrada ya está fijado por la red original, lo que en algunos casos imposibilita la aplicación. Además se ha de tener en cuenta cuán diferente es nuestro problema del problema original para el que la red fue diseñada y entrenada.

El procedimiento consiste en cargar la red original junto con sus parámetros entrenados, pero eliminando la capa de salida de la misma. De esta forma se obtiene para cada imagen una serie de características abstractas y no conocidas, que en la capa final serán usadas para clasificar los datos en las distintas categorías.

A esta primera fase en la que se encuentran este conjunto de capas se le denomina extractor de características. A la capa final, que por lo general suele ser una capa de neuronas completamente conectadas, se le denomina clasificador.

A la salida intermedia de la red anterior se acopla un nuevo clasificador, definido especialmente para el problema que pretendemos resolver.

El clasificador toma los datos de entrada procesados por la red que hemos cargado, donde se obtienen las características de las imágenes, y mediante el procesamiento de dicha información nos proporciona la clasificación de las mismas.

En nuestro caso haremos uso de la red Xception, basada en la red Inception.

La idea novedosa de esta última es que la arquitectura consiste en concatenar los resultados obtenidos tras aplicar diferentes filtros de convolución añadiendo ventanas de pooling a una misma entrada. Esto permite al modelo beneficiarse de la extracción de características a múltiples niveles en un único paso, pudiendo extraer características generales y locales a la vez.

La arquitectura de la red Xception es una extensión de Inception. De hecho, al igual que la anterior, esta arquitectura fue presentada por Google. La principal diferencia que presenta es que introduce el concepto de convolución separable por profundidad. Es decir, en lugar de afrontar la convolución como un único paso en el que se obtiene un único mapa de características resultado de convolucionar un filtro con la entrada, ahora, se obtiene un mapa por cada canal. Es decir, se realiza una convolución independiente de cada canal de entrada.

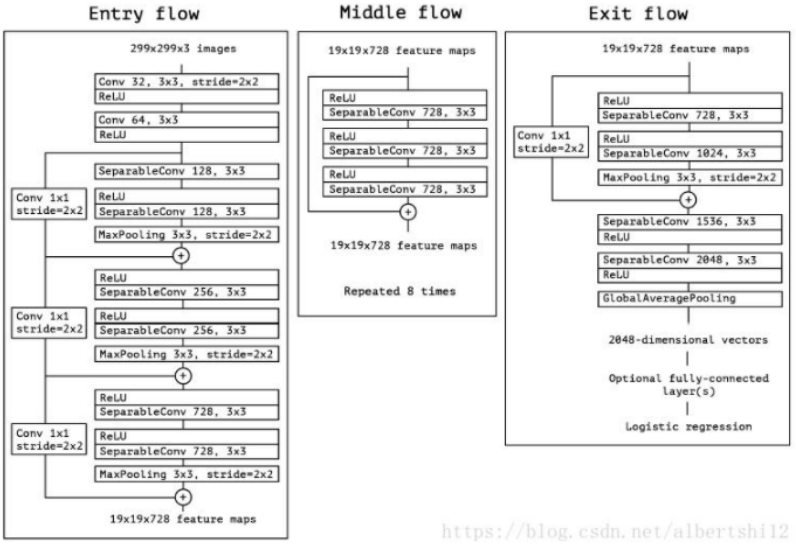


Figura 28. Arquitectura de la red neuronal Xception.

Ambas redes fueron diseñadas para trabajar con un conjunto de datos de ImageNet, el cual contiene 350 millones de imágenes y 17.000 clases.

En particular, la red Xception consta de 22,910,480 parámetros.

La elección de esta red para aplicar Transfer Learning nace de las prestaciones notablemente mejores frente a las que ofrecen otras redes, como por ejemplo, la disminución del número de parámetros y el uso más eficiente de los mismos, precisión, y tiempos de ejecución.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

Figura 29. Comparación de características de varios modelos de redes neuronales.

Una vez cargada y guardadas las salidas obtenidas con dicha red, diseñamos la parte destinada al clasificador, cuya arquitectura será la presentada en la Figura 30.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 200)               409800
_________________________________________________________________
batch_normalization_20 (Batc (None, 200)               800
_________________________________________________________________
dropout_7 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_37 (Dense)             (None, 64)                12864
_________________________________________________________________
batch_normalization_21 (Batc (None, 64)                256
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 65
=================================================================
Total params: 423,785
Trainable params: 423,257
Non-trainable params: 528
_________________________________________________________________
```

Figura 30. Arquitectura diseñada para el clasificador de la red neuronal.

Para evitar parte del sobreajuste detectado aplicamos métodos de regularización como Batch Normalization y Dropout.

Finalmente usamos como recurso TensorBoard para comparar los resultados obtenidos.

Figura 31. Vista del panel resultante de loss y accuracy del modelo obtenido mediante Tensorboard.

En la Figura 31 observamos las líneas que representan los valores de accuracy y loss obtenidos durante cada epoch. Además TensorBoard nos ofrece múltiples opciones, como cambiar el modo de visualización, ofrecer distribuciones sobre los batch normalization o detalles de la arquitectura de la red.

Gracias a estas gráficas hemos detectado overfitting en la red y por ello hemos aplicado técnicas de regularización en la etapa de clasificación, como Dropout o BatchNormalization.

7.Análisis de los resultados obtenidos.

En los resultados representados en las gráficas, obtenemos un valor máximo de accuracy para testeo de 0.9, aunque durante el entrenamiento hemos logrado obtener una precisión superior al 99%.

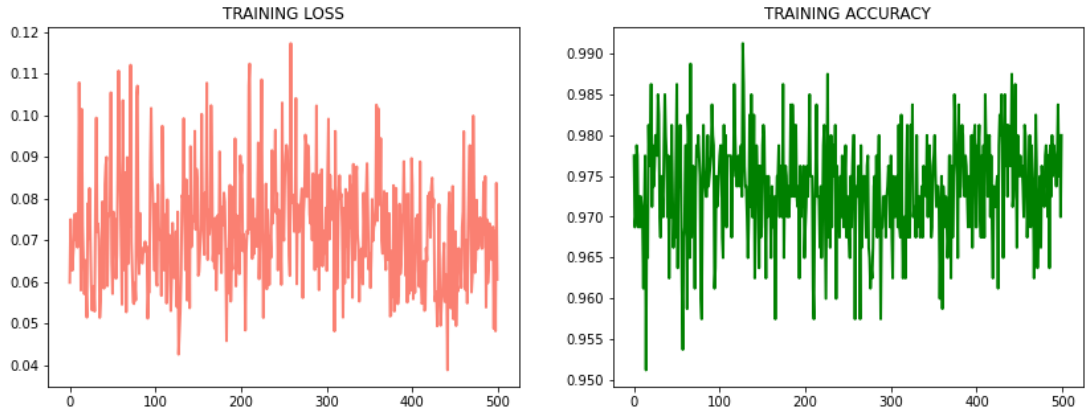


Figura 32. Gráficas de loss y accuracy del dataset de train obtenidas durante el entrenamiento del modelo.

Para evaluar el modelo de una forma más eficiente construimos una tabla con las principales métricas: precisión, recall, f1-score y support.

	precision	recall	f1-score	support
0	0.91	0.95	0.93	145
1	0.85	0.75	0.80	55
accuracy			0.90	200
macro avg	0.88	0.85	0.86	200
weighted avg	0.89	0.90	0.89	200

Figura 33. Cuadro con las principales métricas para evaluación del modelo (precisión, recall, f1-score y support).

Respecto a la precisión del modelo, obtenemos un 91% de accuracy para clasificar aquellas imágenes en las que hay un incendio y un 85% en aquellas en las que no hay fuego. Obtenemos un 90% como precisión final del modelo.

Obtenemos una puntuación de 0.95 en recall . Este cociente es el resultado de dividir el número de imágenes predichas por el modelo como incendios y realmente lo son, entre todas las imágenes que han sido clasificadas por el modelo como fotografías con incendios. De forma análoga obtenemos un valor de 0.75 para las imágenes sin incendios.

En valor de F1-score se utiliza para combinar los resultados obtenidos mediante el recall y precisión.

Por último mostramos que son 145 las imágenes con fuego las que componen el dataset de testeo y 55 las que nos presentan un incendio.

Finalmente diseñamos una matriz de confusión obteniendo los siguientes resultados:

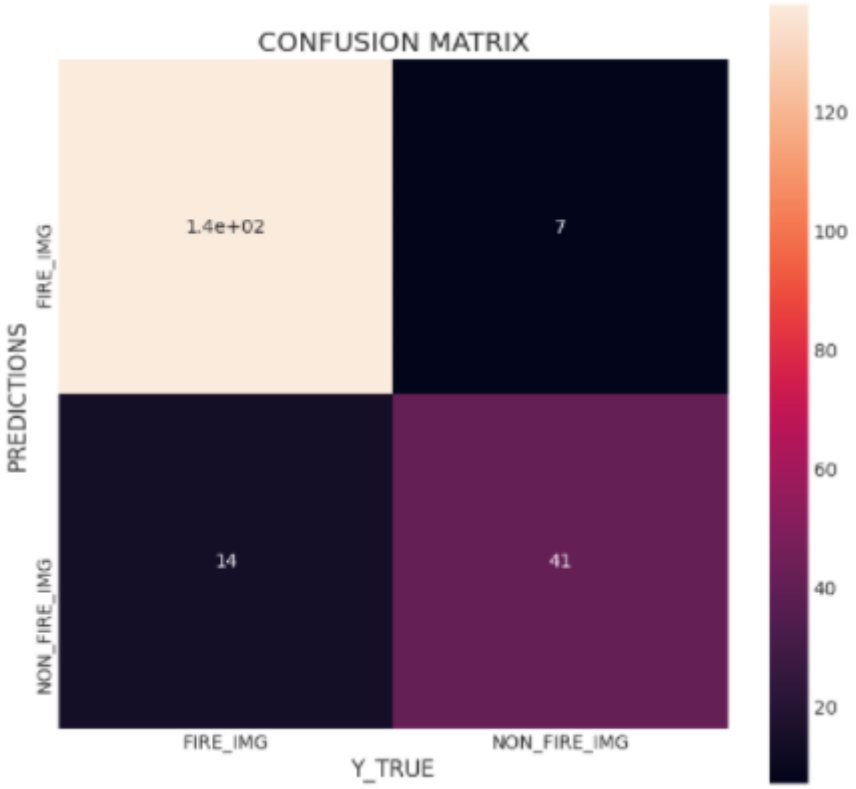


Figura 34. Matriz de confusión obtenida con los resultados de testeo.

Analicemos la diagonal que parte del primer cuadrante hasta el tercer cuadrante (de la parte superior derecha hasta la zona inferior izquierda) en la que se encuentran los falsos positivos y negativos.

Observamos que son 7 las imágenes clasificadas como fotografías con incendios y que realmente no lo son, y 14 las imágenes en las que no se ha detectado fuego y realmente lo presentaban.

8.Conclusiones y planes de mejora.

Con el modelo presentado en este proyecto, diseñado para detección de incendios forestales, logramos obtener un 90% de precisión.
Una de las principales ventajas que presenta este algoritmo es que gracias a la técnica de Transfer Learning que hemos aplicado, haciendo uso de la red Xception entrenada con uno de los mayores conjuntos de ImageNet presenta altas posibilidades de versatilidad. Siendo relativamente sencilla la adecuación del mismo a entornos diferentes donde se puedan provocar incendios; como por ejemplo granjas, edificios públicos o incluso hogares.

Aunque hemos comprobado que la tasa de error es muy pequeña, queda un porcentaje de mejora en cuanto a la precisión del modelo.

Además sería conveniente combinar los resultados de este proyecto con el algoritmo Bee2FireDetection diseñado por IBM y Compta, con el objetivo de cubrir el ámbito de prevención.

De esta forma se tendría una mayor eficacia, y por tanto el número de incendios "descontrolados", causantes de grandes catástrofes, se vería reducido considerablemente.

9.Bibliografía

- FIRE Dataset. (2020, 25 febrero). Kaggle. https://www.kaggle.com/phylake1337/fire-dataset

- Inteligencia artificial contra incendios forestales - innovación en español. Innovaspain. (2019, 23 julio)https://www.innovaspain.com/ibm-inteligencia-artificial-incendios-forestales

- Introduccion Imagen. (2021). https://www.unioviedo.es/compnum/laboratorios_py/Intro_imagen/introduccion_imagen.html#:~:text=Las%20m%C3%A1scaras%20son%20filtros%20geom%C3%A9tricos,y%20ceros%20en%20el%20resto.

- *Kaggle: Your Home for Data Science*. (s. f.). Https://Www.Kaggle.Com/Celiaquero/Fire-Detection-Computer-Vision. Recuperado 15 de junio de 2021, de https://www.kaggle.com/celiaquero/fire-detection-computer-vision

- Manipulación y procesamiento de imágenes usando Numpy y Scipy — Scipy lecture notes. (2021). https://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html.

- Pandey, P. (2020, 15 noviembre). Image Segmentation using Python's scikit-image module. Medium. https://towardsdatascience.com/image-segmentation-using-pythons-scikit-image-module-533a61ecc980

- Sancho, B. S. J. (2021b). Machine Learning y Deep Learning. RAMA.

- Saponara, S. (2020, 10 noviembre). Real-time video fire/smoke detection based on CNN in antifire surveillance systems. Journal of Real-Time Image Processing. https://link.springer.com/article/10.1007/s11554-020-01044-0?error=cookies_not_supported&code=1553189b-b5d8-43a8-96ab-c597de83b317