Pontificia Universidad Católica del Perú

Facultad de Ciencias e Ingeniería

Deep Learning (1INF52)

Informe de Proyecto

Cántaro Márquez, Patricia Natividad	20210907
Nicho Manrique, Saymon Estefano	20211866
Zegarra Barrenechea, Carlos Eduardo	20216177

20 de febrero de 2025

Índice

1. Introducción

Los incendios forestales representan una amenaza para el medio ambiente y la salud pública. En la actualidad, su impacto ha aumentado por el cambio climático y el crecimiento de la actividad humana en zonas boscosas [1]. Estos eventos pueden partir de un origen natural o antrópico (provocados, por ejemplo, por negligencia humana) y generan consecuencias graves como daños en infraestructuras, pérdida de biodiversidad y efectos económicos significativos.

La creciente frecuencia de incendios en bosques y otros biomas ha impulsado el desarrollo de sistemas automáticos de vigilancia diseñados para detectar fuego y humo de manera temprana, reduciendo el tiempo de respuesta de los equipos de extinción y minimizando la propagación de los incendios.

El aprendizaje profundo (*deep learning*) ha demostrado ser altamente eficaz en el análisis de imágenes, especialmente para la detección de incendios y humo, superando limitaciones presentes en métodos tradicionales. En este trabajo, se propondrá el **desarrollo de un modelo** basado en estas tecnologías con el objetivo de integrarlo en un dron para la **detección en tiempo real de incendios forestales de forma más rápida y eficiente**.

2. Planteamiento del Problema

A pesar de los avances en el uso de aprendizaje profundo para la detección de incendios, muchas de las arquitecturas existentes no son lo suficientemente ligeras para ser desplegadas en un dron, lo que limita su aplicabilidad en entornos de monitoreo aéreo en tiempo real. Aunque existen modelos optimizados para dispositivos con restricciones computacionales, muchos de ellos no han sido actualizados con las versiones más recientes de arquitecturas de detección de objetos, lo que podría afectar su precisión y eficiencia.

Este trabajo busca abordar esta problemática explorando modelos que logren un equilibrio entre precisión y eficiencia computacional, permitiendo su implementación efectiva en drones para la detección temprana de incendios forestales.

3. Estado del Arte

3.1. Métodos Clásicos y Sensores

Existen diversos enfoques convencionales de detección de incendios:

1. Sistemas basados en sensores (ópticos, de humo, de gas, de temperatura, etc) que pueden percibir señales asociadas al fuego, pero cuyo alcance y capacidad de respuesta pueden resultar limitados.
2. Técnicas de visión clásica basadas en la segmentación de color característico del fuego (naranja, amarillo, rojo) o el humo (blanco, gris, negro) que han sido útiles en sistemas de videovigilancia tempranos, pero que suelen verse afectados por altas tasas de falsas alarmas (reflejos, luces parásitas, nubes con tonos similares, etc).

3.2. Métodos con ML y DL

Con la expansión de la videovigilancia y la aparición de la visión por computadora, se han intentado proponer métodos más sofisticados que no solo integran lo mencionado anteriormente, sino que también agregan forma, textura, dinámica y variación de luz para identificar el comportamiento propio del fuego.

3.2.1. Clasificación con CNNs pre entrenadas

Diversas arquitecturas de redes neuronales convolucionales (CNN), como pueden ser Inception, VGGNet, Xception, DenseNet o ResNet, han demostrado resultados prometedores en la clasificación de imágenes de fuego y humo. Sin embargo, uno de los retos es la disponibilidad de grandes volúmenes de datos. Por ello, se han considerado diversas técnicas que permitan sobrellevar este problema.

1. **Transfer Learning:** Consiste en reutilizar una red preentrenada en un conjunto de datos grande y ajustarla a un conjunto de datos más pequeño y específico.
2. **Fine Tuning:** Consiste en ajustar los pesos de una red preentrenada en un conjunto de datos similar al de interés, pero con una tarea diferente.
3. **Data Augmentation:** Consiste en generar nuevas imágenes a partir de las existentes, aplicando transformaciones como rotaciones, traslaciones, zooms, cambios de brillo y contraste, etc.

3.2.2. Detección de objetos con YOLO

A pesar de que las CNNs clásicas son efectivas en la clasificación de imágenes, no son tan eficientes en la detección de objetos. Por ello, se han propuesto arquitecturas como YOLO (You Only Look Once) que permiten identificar y localizar elementos en imágenes en tiempo real y con alta precisión.

3.2.3. Clasificación con Transformadores de Visión (ViTs)

Los Vision Transformers (ViTs) han surgido como una alternativa prometedora a las CNNs para la clasificación de imágenes. A diferencia de las CNNs, que utilizan convoluciones para extraer características, los ViTs emplean mecanismos de autoatención para capturar relaciones globales en la imagen. Esta capacidad les permite modelar dependencias a largo alcance y adaptarse mejor a variaciones en la textura y el contexto del fuego y el humo. Aunque requieren grandes volúmenes de datos para un entrenamiento efectivo, el preentrenamiento en datasets masivos y fine-tuning ha permitido su aplicación en la detección de incendios con resultados competitivos.

3.3. Hallazgos recientes destacados

3.3.1. Yang (2019), Mao et al. (2018)

Desarrollaron un pipeline para clasificación de incendios forestales con CNNs pre entrenadas de 3 modelos destacables, VGG16, InceptionV3 y Xception. Exploraron el fine tuning y usaron optimización bayesianda con LwF. Esto demostró una mejor abrupta en los nuevos datosy demostró el gran potencial existente en las técnicas de deep learning y transfer learning para la detección de incendios y clasificación de imágenes.

3.3.2. Chetoui & Akhloufi (2024)

Los autores propusieron emplear los modelos YOLOv8 y YOLOv7 para la detección de incendios forestales. Para ello, usaron un dataset de 11,000 imágenes de incendios. Además, aplicaron técnicas de fine tuning y data augmentation para mejorar el rendimiento de los modelos. Condujeron sus experimentos en un NVidia V100SXM2 (16GB) y en un CPU Intel Gold 6148 Skylake (2.4 GHz). Los resultados mostraron que YOLOv8 superó a YOLOv7 en términos de precisión y recall. Incluso en situaciones con bajo contraste del humo en las imágenes de validación, logró obtener una confianza de aproximadamente 0.8 [2].

3.3.3. Mehta & Rastegari (2022)

Los autores propusieron MobileViT, un modelo ligero basado en Vision Transformers (ViTs) optimizado para tareas de visión en dispositivos con recursos limitados, como teléfonos y drones. A diferencia de ViTs tradicionales, que requieren una gran cantidad de parámetros y capacidad computacional, MobileViT combina convoluciones con autoatención global para capturar tanto características locales como globales con menos cómputo. Sus experimentos en ImageNet-1k y MS-COCO demostraron que MobileViT supera a CNNs ligeras como MobileNetV3 y a modelos ViT compactos como DeIT, logrando un mejor equilibrio entre precisión y eficiencia. Este enfoque sugiere que los transformers pueden ser una opción viable para la detección de incendios en tiempo real en drones sin comprometer rendimiento ni consumo energético [3].

3.3.4. Palaparthi & Nangi (2023)

Los autores desarrollaron FireSight, un modelo de clasificación de incendios basado en imágenes aéreas capturadas por drones. Para mejorar la precisión y eficiencia en dispositivos con recursos limitados, combinaron Vision Transformers (ViTs) con CNNs en un modelo ensemblado. Su enfoque incluyó técnicas de fine-tuning, data augmentation y reducción de parámetros mediante la poda de capas del ViT. Compararon su método con arquitecturas previas como Xception y ResNet, logrando un 82.28 % de precisión en la detección de incendios en el conjunto de datos FLAME. Sus experimentos demostraron que el ensemblado de ViTs y CNNs captura mejor las características de fuego y humo, lo que hace que este enfoque sea prometedor para la detección temprana de incendios con drones [4].

4. Marco Teórico

El propósito de este marco teórico es introducir los conceptos clave que permitan abordar la problemática de la detección de incendios forestales a partir del análisis de imágenes con técnicas de *deep learning*. Se explorarán las capacidades de las redes neuronales, específicamente las redes neuronales convolucionales (CNNs), como herramientas a usar para la detección y mitigación de estos eventos.

4.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNNs) son un tipo de arquitectura de aprendizaje profundo altamente efectiva para el procesamiento de datos que se pueden representar

como una cuadrícula, como lo son las imágenes. Su diseño les permite aprender de forma automática jerarquías espaciales y patrones relevantes, lo cual las convierte en una herramienta ideal para la claificación de imágenes y la detección de objetos.

Su arquitectura generalmente consta de tres tipos de capas principales: convolucionales, de agrupación (*pooling*) y completamente conectadas. Las capas convolucionales aplican filtros a la imagen de entrada para extraer características relevantes, mientras que las de agrupación reducen la dimensionalidad de las características extraídas. Finalmente, las capas completamente conectadas se encargan de la clasificación final, al asegurar que cada neurona de salida esté conectada a todas las neuronas de la capa anterior.

El uso de CNNs en la detección de humo de incendios forestales se ha vuelto clave gracias a su capacidad para procesar grandes volúmenes de datos visuales, como imágenes sateliteles y transmisiones de cámaras de vigilancia. Esto permite diferenciar el humo de elementos como nubes o niebla, y de forma más eficiente a diferencia de métodos tradicionales.

4.2. Xception (Extreme Inception)

Xception es una arquitectura basada en Inception que sustituye las convoluciones estándar por *depthwise separable convolutions*, lo que reduce la cantidad de parámetros sin afectar el rendimiento. Separa la extracción de características en dos etapas: primero filtra por canal y luego mezcla la información entre canales. Esto permite un aprendizaje más eficiente y ha mostrado mejor rendimiento que InceptionV3 en conjuntos de datos grandes, aunque su implementación puede ser más costosa computacionalmente en algunos casos [5].

4.3. DenseNet

DenseNet optimiza el flujo de información conectando cada capa con todas las anteriores, fomentando la reutilización de características y mejorando la propagación del gradiente. Gracias a esta estructura, logra reducir el número de parámetros en comparación con otras arquitecturas profundas sin perder precisión. Es útil para modelos muy profundos, pero su alto número de conexiones puede aumentar el consumo de memoria durante el entrenamiento.

4.4. ResNet

ResNet introduce *skip connections* o conexiones residuales que permiten que los gradientes atraviesen varias capas sin degradarse, resolviendo el problema del desvanecimiento del gradiente en redes profundas. En lugar de aprender transformaciones completas, aprende diferencias entre la entrada y la salida esperada, lo que facilita el entrenamiento y permite construir redes con cientos de capas. Aunque es altamente eficiente, las versiones más profundas pueden requerir un ajuste cuidadoso de hiperparámetros para evitar sobrecostos computacionales [5].

4.5. YOLO

You Only Look Once es un modelo avanzaado de detección de objetos diseñado para identificar y localizar elemntos en imágenes en tiempo real con alta precisión. Su funcionamiento se basa en redes neuronales convolucionales (CNNs) para analizar imágenes con

un solo procesamiento, dividiéndolas en cuadrículas y prediciendo la posición, dimensiones y clase de los objetos detectados [6].

YOLOv8, la versión más reciente, introduce un enfoque sin anclas (*anchor-free*), eliminando las posiciones predefinidas de las cajas delimitadoras usadas en versiones anteriores. Esto le ayuda a simplificar el entrenamiento y mejorar la precisión en la detección de objetos con formas y tamaños variables. Además, incorpora CSPNet (*Cross-Stage Partial Networks*) como backbone, una arquitectura que optimiza el flujo de información y reutiliza características de capas anteriores para reducir las redundancias y mejorar la eficiencia computacional.

Estas características le permiten a YOLOv8 servir como herramienta altamente eficaz para tareas en tiempo real, y para actividades de vigilancia y monitoreo ambiental como es el caso de la detección de incendios forestales [7].

4.6. Transfer Learning

El aprendizaje por transferencia es una técnica de aprendizaje automático que consiste en reutilizar conocimientos aprendidos en un dominio para mejorar el rendimiento en otro dominio relacionado. En el contexto de las redes neuronales, esto implica tomar una red preentrenada en un conjunto de datos grande y ajustarla a un conjunto de datos más pequeño y específico.

5. Baseline

Nuestro modelo se basa en el trabajo realizado por Palaparthi y Nangi en el proyecto FireSight de Stanford, el cual propone un enfoque de clasificación binaria para la detección de incendios en imágenes aéreas capturadas por drones. En su estudio, los autores comparan el desempeño de redes convolucionales profundas (CNN) con modelos basados en Transformers, implementando técnicas de aumento de datos y estrategias de ensamble para mejorar la precisión del modelo.

Como baseline, tomamos la arquitectura de FireSight, la cual alcanza una precisión del 82.28 % mediante un ensamble de modelos CNN (DenseNet y ResNet) y Transformers (ViT). El modelo de Stanford evalúa diferentes estrategias de combinación de características y probabilidad para mejorar la clasificación entre imágenes con y sin fuego, y además explora la compresión de modelos para una posible implementación en dispositivos con recursos limitados, como UAVs.

Dado que las CNNs utilizadas en FireSight no lograron igualar el rendimiento de los Transformers en la detección de incendios, nuestro trabajo se centra en mejorar los resultados de los modelos CNN mediante un ensamble optimizado de Xception, DenseNet y ResNet.

Además, una vez obtenido el mejor modelo de ensamble, aplicaremos knowledge distillation con MobileNetV3, buscando reducir el tamaño del modelo sin perder precisión, para su posible implementación en sistemas de detección en tiempo real.

6. Propuesta de Modelo

6.1. Dataset

Se usará el FLAME dataset [8] ya que este ha sido diseñado para estudios de detección y segmentación de incendios, conteniendo imágenes y videos capturados mediante drones en bosques del norte de Arizona. Para este proyecto, se usarán solo las imágenes y se aprovechará que estas ya han sido distribuidas y etiquetadas para ser usadas: 39,375 imágenes para entrenamiento/validación y 8,617 imágenes para testeo.

6.2. Arquitectura Propuesta

Con el dataset de FLAME, se usará un ensamble de 3 modelos, los cuales serán: Xception, DenseNet y ResNet; y posteriormente se destilarán en un modelo ligero como lo es MobileNetV3. Además, se usarán ténicas de pruning antes del deployment para reducir el tamaño del modelo y mejorar la eficiencia en la inferencia.

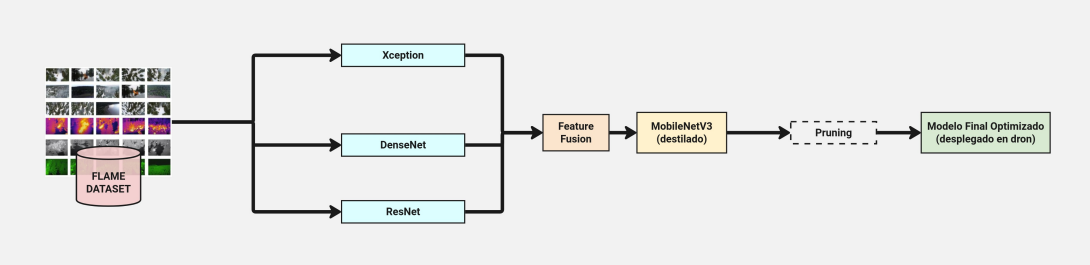


Figura 1: Descripción del modelo

Se utilizarán **CNNs entrenadas sobre el dataset FLAME**, específicamente:

- **Xception**: Utiliza *Depthwise Separable Convolutions (DSC)* para optimizar la extracción de características, capturando relaciones complejas en la imagen. Se considera más eficiente en comparación con arquitecturas tradicionales.
- **DenseNet**: Usa *bloques densos* que facilitan la reutilización de características, beneficiando la detección de elementos sutiles como *ríos o humo de bajo contraste*.
- **ResNet**: Implementa *conexiones residuales* para evitar la degradación del gradiente en redes profundas, lo que mejora la convergencia en entrenamientos largos.

6.2.1. Entrenamiento previo al ensamble

Cada modelo se entrenará **de manera individual** para la clasificación de incendios, empleando *cross-entropy* como función de pérdida y aplicando *early stopping* para evitar el sobreajuste.

6.2.2. Fusión de Modelos (Ensamble)

Una vez completados los entrenamientos individuales, se procederá con la fusión de modelos. Se consideran dos estrategias:

1. **Fusión a nivel de salidas finales**:

- Se combinan las *probabilidades* generadas por los tres modelos en un vector mayor.
- Se ajusta el *peso* de cada modelo y se valida con *cross-validation*.

2. **Fusión en capas intermedias**:
   - Se extraen *vectores de características* desde capas intermedias de cada modelo y se concatenan en un vector mayor.
   - Se entrena un *clasificador adicional* sobre este vector fusionado para generar la salida final.

6.2.3. Knowledge Distillation

Dado que el ensamble es **demasiado grande** para dispositivos con recursos limitados, se aplicará *Knowledge Distillation*. Se entrenará un **MobileNetV3** como versión comprimida del ensamble, preservando la mayor cantidad de información relevante.

6.2.4. Optimización Adicional: Pruning

Para reducir aún más el tamaño del modelo, se aplicará *pruning*, eliminando **pesos irrelevantes o de magnitud baja**. Esto permite reducir la carga computacional sin afectar significativamente la precisión.

6.3. Justificación

- **Uso de Xception, DenseNet y ResNet en paralelo:**
  - **Xception**: Usa convoluciones separables, reduciendo cálculos sin perder precisión.
  - **DenseNet**: Mejora reutilización de características, ideal para detectar detalles pequeños.
  - **ResNet**: Usa conexiones residuales, facilitando el entrenamiento en redes profundas.
- **Destilación en MobileNetV3:**
  - Modelo ligero y eficiente para *drones*.
  - Mantiene precisión al aprender de los modelos grandes.
- **Ventajas sobre el estado del arte:**
  - Más eficiente que ViT y ResNet en inferencia en dispositivos de bajo consumo.
  - Mejor precisión que modelos CNN individuales.
- **Despliegue en *drones*:**
  - Menor consumo de energía que ViT y CNNs pesados.
  - Inferencia en tiempo real en *Jetson Nano, Coral TPU, Raspberry Pi*.

6.4. Pipeline

El flujo de producción está diseñado para ejecutar una secuencia de cinco pasos que garantizan reproducibilidad y eficiencia. Estos pasos son:

1. **Entrenamiento de DenseNet:** El modelo DenseNet se entrena utilizando hiperparámetros fijos y ajustados.
2. **Entrenamiento de ResNet:** De manera similar, el modelo ResNet se entrena con sus mejores hiperparámetros.
3. **Entrenamiento de Xception:** El modelo Xception se entrena bajo el mismo enfoque.
4. **Ensamblado de modelos:** Las predicciones de los tres modelos entrenados se promedian para construir un modelo en ensamblado.
5. **Destilación del modelo:** El modelo ensamblado actúa como maestro para destilar el conocimiento en un modelo estudiante más ligero.

El script de bash (ubicado en `scripts/run_pipeline.sh`) orquesta el flujo de trabajo.

7. Preprocesamiento de datos

7.1. Exploración de datos

Se realizaron ciertas tareas de exploración de datos sobre el FLAME dataset, para lo cual el enfoque fueron las imágenes de entrenamiento y prueba de las categorías **Fire** y **No_Fire**. A continuación, se muestra un resumen de los principales hallazgos:

1. **Distribución de imágenes:**
   a) **Entrenamiento:**
      - **No_Fire:** 14 357 imágenes ($\approx 36{,}45\,\%$)
      - **Fire:** 25 027 imágenes ($\approx 63{,}55\,\%$)

   b) **Prueba:**
      - **No_Fire:** 5 137 imágenes ($\approx 59{,}61\,\%$)
      - **Fire:** 3 480 imágenes ($\approx 40{,}39\,\%$)

   Se observa un desbalance en el conjunto de entrenamiento, donde la categoría **Fire** predomina.
2. **Formato y dimensiones:** Todas las imágenes se encuentran en formato JPEG y poseen dimensiones uniformes.
3. **Análisis de color:** Se realizó un estudio de los histogramas de color para identificar patrones en las distribuciones de intensidades:
   - Las imágenes de la categoría **Fire** muestran picos de intensidad en el canal rojo, lo cual es coherente con la presencia de fuego y las altas temperaturas que generan tonos más cálidos.

- En contraste, las imágenes de la categoría **No_Fire** exhiben distribuciones de color más uniformes o predominio de tonos verdes y azules, lo que sugiere escenas más naturales o ambientes sin fuego.

Debajo se muestran algunos histogramas de dichas categorías:

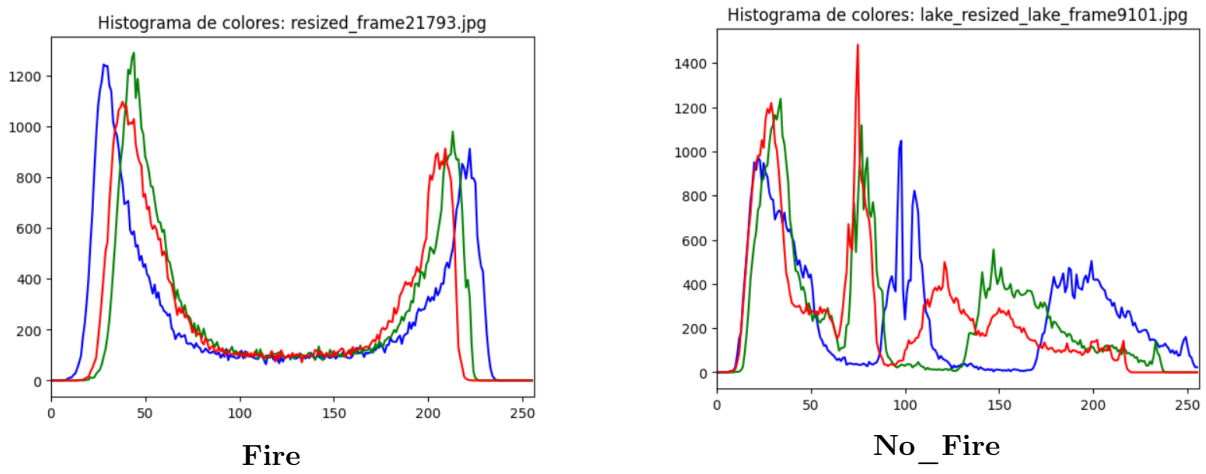


Figura 2: Histogramas de color para las categorías **Fire** y **No_Fire**.

7.2. Redimensionamiento y Aumento de datos

Dado que emplearemos modelos preentrenados, se optó por redimensionar todas las imágenes a 224x224 píxeles. Para este propósito, se implementó una función que emplea el método de remuestreo LANCZOS (óptimo para preservar la calidad). Luego, se guardaron las imágenes en una estructura de directorios paralela a la original.

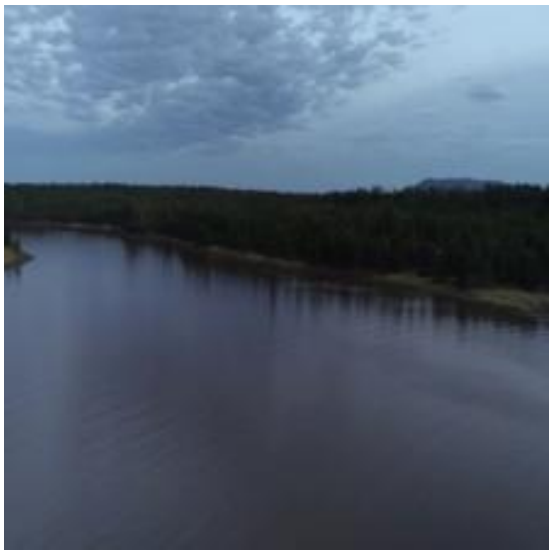
Además, durante la etapa de exploración, se observó un desbalance significativo entre las dos categorías de los datos de entrenamiento. En particular, la clase **Fire** contaba con un mayor número de imágenes que la clase **No_Fire**.

A fin de abordar este problema y mejorar la capacidad de generalización del modelo, se aplicó una técnica de *data augmentation* utilizando la clase `ImageDataGenerator` de Keras. Esta técnica genera nuevas imágenes a partir de las originales aplicando transformaciones aleatorias, lo que permite aumentar el número de ejemplos y enriquecer el dataset con variaciones realistas. Las transformaciones utilizadas fueron:

- **Rotación:** Se rotó la imagen en un rango de $\pm 30°$, ayudando al modelo a reconocer objetos independientemente de su orientación.
- **Desplazamiento:** Se realizaron cambios aleatorios en la posición horizontal y vertical (hasta el 20 % del tamaño), lo que permitió que el modelo sea menos sensible a la ubicación del objeto dentro de la imagen.
- **Cizallamiento (shear):** Se aplicó una deformación de la imagen que simula cambios en la perspectiva, contribuyendo a la robustez del modelo frente a distorsiones.
- **Zoom:** Se efectuó un zoom in/out en un rango del 20 %, generando variaciones en la escala del objeto.

- **Volteo horizontal:** Se invirtió la imagen de forma horizontal, lo cual duplica las variaciones disponibles y es especialmente útil cuando la orientación no afecta la clasificación.
- **Relleno:** Se utilizó el modo de relleno `'nearest'` para completar los espacios vacíos que puedan generarse durante las transformaciones.

Para aplicarlo, se calculó el número de imágenes adicionales necesarias de tal forma que la cantidad de imágenes en la clase **No_Fire** sea equivalente a la de **Fire**. Luego, el generador aplicó de forma iterativa estas transformaciones a cada imagen de la categoría **No_Fire** y se guardaron las nuevas imágenes en un directorio específico para datos aumentados. Debajo se muestra un ejemplo de una imagen aumentada.



Original

Aumentada

Figura 3: Ejemplo de imagen original y su versión aumentada en la categoría **No_Fire**.

8. Entrenamiento

En esta sección, describimos la primera fase de nuestro experimento con la arquitectura DenseNet. El objetivo de este entrenamiento inicial es obtener un modelo base y analizar aspectos clave como la estabilidad de la pérdida, el sobreajuste y los desbalances en las clases. Los resultados de esta fase servirán como base para mejorar el modelo y el entrenamiento de los demás en las siguientes iteraciones.

8.1. Hiperparámetros

8.1.1. Tamaño del Batch (Batch Size)

- **Inicial:** 16
- **Optimización:** Se redujo a 8 en la última versión del modelo para mejorar la **generalización**, reduciendo la probabilidad de sobreajuste.

8.1.2. Tasa de Aprendizaje (Learning Rate)

- **Inicial:** 0.001
- **Optimización:** Se redujo a 0.0001 para hacer los ajustes de los pesos más suaves y evitar grandes oscilaciones en la pérdida.

8.1.3. Optimizador

- **Inicial:** Adam
- **Optimización:** Se mantuvo Adam porque ofrece una actualización eficiente de los pesos con momentums adaptativos, pero se ajustaron sus hiperparámetros internos.

8.1.4. Fine-Tuning)

- **Modelo 2:** Se descongelaron 10 capas de DenseNet, pero esto no mejoró la precisión.
- **Modelo 3:** Se redujo a 5 capas, logrando un entrenamiento más estable sin tanto sobreajuste.

8.1.5. Loss Function

- **Inicial:** Cross-Entropy estándar.
- **Optimización:** Se agregó **pesado de clases (class weighting)** para reducir el sesgo del modelo hacia imágenes de "Fuegoz mejorar la detección de "No Fuego".

8.1.6. Balanceo de Clases

- **Problema:** El dataset tenía muchas más imágenes de "Fuego"que de "No Fuego".
- **Solución:** Se aplicó pesado de clases en la función de pérdida, obligando al modelo a prestar más atención a la clase minoritaria.

8.1.7. Epochs

- **Inicial:** 30 épocas.
- **Optimización:** Se ha mantenido, pero con ajustes en la tasa de aprendizaje y el batch size para evitar sobreajuste prematuro.

8.1.8. Data Augmentation

- **Inicial:** Transformaciones básicas como escalado y normalización.
- **Optimización:** Se añadieron rotaciones, flips horizontales y cambios de brillo y contraste para mejorar la robustez del modelo.

8.1.9. Early Stopping

- Habilitado para detener el entrenamiento si la pérdida de validación deja de mejorar después de un cierto número de épocas, evitando un uso innecesario de recursos.

8.1.10. Número de Neuronas en Capas Densas

- **Denso Final:** Se mantuvo en 512 neuronas con activación ReLU antes de la capa de salida.

Regularización L2

- Añadida en las capas densas para evitar sobreajuste al penalizar valores de pesos muy altos.

8.1.11. Dropout

- **Inicial:** 0.3
- **Optimización:** Aumentado a 0.5 en la última versión para mejorar la robustez del modelo reduciendo la dependencia de neuronas individuales.

8.1.12. Análisis de Mejoras del Modelo

Modelo 1 → Modelo 2

- El ajuste fino (descongelar 10 capas) no mejoró la precisión.
- La pérdida de validación disminuyó, pero el sobreajuste aumentó.
- El desbalance de clases siguió siendo un problema grave (Recall de "No Fuego- 0.02).
- El modelo tenía un sesgo fuerte hacia las imágenes de "Fuego".

8.1.13. Modelo 2 → Modelo 3

- Se redujo el ajuste fino de 10 capas a 5 capas → Mayor estabilidad.
- La pérdida de validación se volvió más estable.
- El Recall de "No Fuego"mejoró de 0.02 a 0.06 (aunque sigue siendo bajo).
- El modelo aún tiene dificultades para detectar "No Fuego".

8.1.14. Modelo 3 → Modelo 4 (Modelo Actual)

- Se redujo el tamaño del batch de 16 a 8 → Mejora la generalización.
- Se redujo la tasa de aprendizaje de 0.001 a 0.0001 → Previene el sobreajuste.
- Se agregó balanceo de pesos de clase → Obliga al modelo a enfocarse más en "No Fuego".
- Se agregó ReduceLROnPlateau → Reduce la tasa de aprendizaje dinámicamente cuando el entrenamiento se ralentiza.

8.1.15. Modelo 4 → Modelo 5

- Se introdujo **regularización L2** con un valor de **0.01** para reducir el sobreajuste penalizando pesos excesivamente altos.

- Se agregó **data augmentation con rotaciones de hasta 20 grados** para mejorar la capacidad de generalización del modelo.
- La precisión global no cambió significativamente, pero el **F1 Score mejoró** levemente y la estabilidad del modelo aumentó.
- A pesar de estos cambios, el modelo aún tenía un **recall bajo en la clase "No Fuego"**.

8.1.16. Modelo 5 → Modelo 6

- Se aumentó el número de **capas descongeladas a 10**, permitiendo un mayor ajuste fino de la red.
- Se añadió **Dropout de 0.3** en las capas densas para reducir el sobreajuste y mejorar la robustez del modelo.
- Se ajustó el **L2 a un valor más pequeño (1e-13)** para permitir mayor flexibilidad en la optimización sin sacrificar generalización.
- Con estos cambios, el modelo alcanzó su **mejor precisión hasta ahora (61.40 %)** y el **mejor F1 Score (0.2152)**.
- La **precisión y recall de "No Fuego"** también mejoraron significativamente, lo que indica un mejor balance en la clasificación.

8.2. Comparación de Modelos

Métrica	**Modelo 1** (Congelado)	**Modelo 2** (10 Capas)	**Modelo 3** (5 Capas)	**Modelo 4** (Balanceado)	**Modelo 5** (L2=0.01)	**Modelo 6** (10 Capas, L2=1e-13)
Precisión	57.27 %	57.27 %	58.76 %	58.48 %	58.48 %	61.40 %
F1 Score	0.0386	0.0386	0.1012	0.0996	0.1033	0.2152
R^2 Score	-0.7187	-0.7187	-0.5241	-0.4149	-0.4998	-0.4499
AUC-ROC	0.7820	0.7820	0.8306	0.8293	0.8304	0.8324
Precisión Fuego	0.59	0.59	0.60	0.60	0.60	0.62
Recall Fuego	0.95	0.95	0.95	0.94	0.94	0.94
Precisión No Fuego	0.21	0.21	0.42	0.40	0.40	0.60
Recall No Fuego	0.02	0.02	0.06	0.06	0.06	0.13

Cuadro 1: Comparación de métricas de evaluación entre diferentes modelos entrenados.

8.3. Entrenamiento del Modelo DenseNet

Para entrenar el modelo, se utilizó **DenseNet121** con pesos preentrenados de *ImageNet*, aplicando ajuste fino en sus capas superiores para mejorar su capacidad de extracción de características. La optimización de hiperparámetros se realizó utilizando **Keras Tuner**, permitiendo encontrar la mejor combinación de valores para maximizar el rendimiento del modelo en la tarea de clasificación.

8.3.1. Optimización con Keras Tuner

Para encontrar la mejor configuración del modelo, se empleó **Keras Tuner** con *RandomSearch*, explorando distintas combinaciones de hiperparámetros:

- **Tasa de dropout**: 0.35
- **Factor de regularización L2**: 1×10^{-3}
- **Número de capas descongeladas**: 20
- **Tasa de aprendizaje**: $1{,}047 \times 10^{-3}$

Se realizaron 10 experimentos (*trials*), almacenando el mejor modelo de cada uno. El mejor modelo se determinó en función de la **exactitud en el conjunto de validación**.

8.3.2. Resultados del Mejor Modelo

El modelo con los hiperparámetros óptimos obtuvo los siguientes resultados en el conjunto de testeo:

- **Exactitud (Accuracy)**: 86.50 %
- **F1-Score**: 83.24 %
- **Coeficiente de Determinación ($\mathbf{R^2}$)**: 0.5798
- **Área bajo la curva ROC (AUC-ROC)**: 94.60 %

El modelo final se guardó como `final_best_model.keras` y se usaron para la fase de ensamble despliegue.

8.4. Entrenamiento del Modelo ResNet

Para entrenar el modelo, se utilizó **ResNet50** con pesos preentrenados de *ImageNet*, permitiendo el ajuste fino de ciertas capas para mejorar su capacidad de generalización. La optimización de hiperparámetros se realizó utilizando **Keras Tuner**.

8.4.1. Optimización con Keras Tuner

Se empleó **Keras Tuner** con *RandomSearch*, evaluando distintas combinaciones de hiperparámetros. Los mejores valores obtenidos fueron:

- **Tasa de dropout**: 0.40
- **Factor de regularización L2**: 5×10^{-4}
- **Número de capas descongeladas**: 15
- **Tasa de aprendizaje**: $8{,}23 \times 10^{-4}$

Se ejecutaron 10 pruebas (*trials*), almacenando el mejor modelo de cada una. El criterio para seleccionar el mejor modelo fue la **exactitud en el conjunto de validación**.

8.4.2. Evaluación del Mejor Modelo

Después de seleccionar el mejor modelo basado en validación, se realizó una evaluación final en el conjunto de prueba. Los resultados obtenidos fueron:

- **Exactitud (Accuracy)**: 60.20 %
- **F1-Score**: 60.10 %
- **Coeficiente de Determinación (R^2)**: 0.4621
- **Área bajo la curva ROC (AUC-ROC)**: 70.75 %

El modelo final se guardó como `final_best_model.keras` y será utilizado en la fase de despliegue.

8.5. Entrenamiento del Modelo Xception

Para entrenar el modelo, se utilizó **Xception**, una arquitectura basada en *deep separable convolutions*, con pesos preentrenados de *ImageNet*. Se realizó un ajuste fino en las capas superiores para mejorar la capacidad de aprendizaje de características específicas. La optimización de hiperparámetros se realizó utilizando **Keras Tuner**.

8.5.1. Optimización con Keras Tuner

Se empleó **Keras Tuner** con *RandomSearch*, explorando múltiples combinaciones de hiperparámetros para mejorar el desempeño del modelo. Los valores óptimos encontrados fueron:

- **Tasa de dropout**: 0.30
- **Factor de regularización L2**: 5×10^{-4}
- **Número de capas descongeladas**: 10
- **Tasa de aprendizaje**: $7{,}15 \times 10^{-4}$

Se ejecutaron 10 pruebas (*trials*), almacenando el mejor modelo de cada una. El criterio para seleccionar el mejor modelo fue la **exactitud en el conjunto de validación**.

8.5.2. Evaluación del Mejor Modelo

Tras seleccionar el mejor modelo basado en validación, se realizó una evaluación final en el conjunto de prueba. Los resultados obtenidos fueron:

- **Exactitud (Accuracy)**: 78.00 %
- **F1-Score**: 67.11 %
- **Coeficiente de Determinación (R^2)**: 0.2840
- **Área bajo la curva ROC (AUC-ROC)**: 86.29 %

El modelo final se guardó como `final_best_model.keras` y será utilizado en la fase de despliegue.

Referencias

[1] Pablo Bot, Mauro Castelli, and Aleš Popovič. A systematic review of applications of machine learning techniques for wildfire management decision support. *International Journal of Disaster Risk Reduction*, 71:102989, 2022.

[2] Mohamed Chetoui and Moulay A. Akhloufi. Fire and smoke detection using fine-tuned yolov8 and yolov7 deep models. `https://www.mdpi.com/2571-6255/7/4/135`, 2024.

[3] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations (ICLR)*, 2022.

[4] Amrita Palaparthi and Sharmila Reddy Nangi. Firesight – wildfire detection through uav aerial image classification. In *Conference on Computer Vision Applications for Wildfire Monitoring*, 2023.

[5] Veerappampalayam Easwaramoorthy Sathishkumar, Jaehyuk Cho, Malliga Subramanian, and Obuli Sai Naren. Forest fire and smoke detection using deep learning-based learning without forgetting. *Fire Ecology*, 19(1):9, February 2023.

[6] Muhammad Yaseen. What is yolov8: An in-depth exploration of the internal features of the next-generation object detector. `https://arxiv.org/abs/2408.15857`, 2024.

[7] Norkobil Saydirasulovich Saydirasulov, Mukhriddin Mukhiddinov, Oybek Djuraev, Akmalbek Abdusalomov, and Young-Im Cho. An improved wildfire smoke detection based on yolov8 and uav images, 2023.

[8] Jeffrey D. Graham, Matthew B. Russell, David Rammer, and Joseph O'Brien. Flame dataset: Aerial imagery for pile burn detection using drones (UAVs). `https://ieee-dataport.org/open-access/flame-dataset-aerial-imagery-pile-burn-detection-using-drones-uavs`, 2024. Accessed: 2024-01-30.