Minería de Datos: Aspectos Avanzados

Equipo de Kaggle: Visual Transformers

David Herrea Poyatos
Adán Cruz Vargas
Andrés Herrera Espino
Zewei Wang Chen

25 de agosto de 2022

Índice

1. Introducción

En este documento se trata el problema de clasificación de usos del suelo, como una competición en la plataforma Kaggle para la asignatura de Minería de Datos: Aspectos Avanzados del máster DATCOM de la UGR.

Los datos constan de 29 clases, siendo cada una un tipo de terreno. Estas clases están formadas por imágenes RGB de tamaño 224x224 píxeles sacadas desde el satélite Sentinel 2.

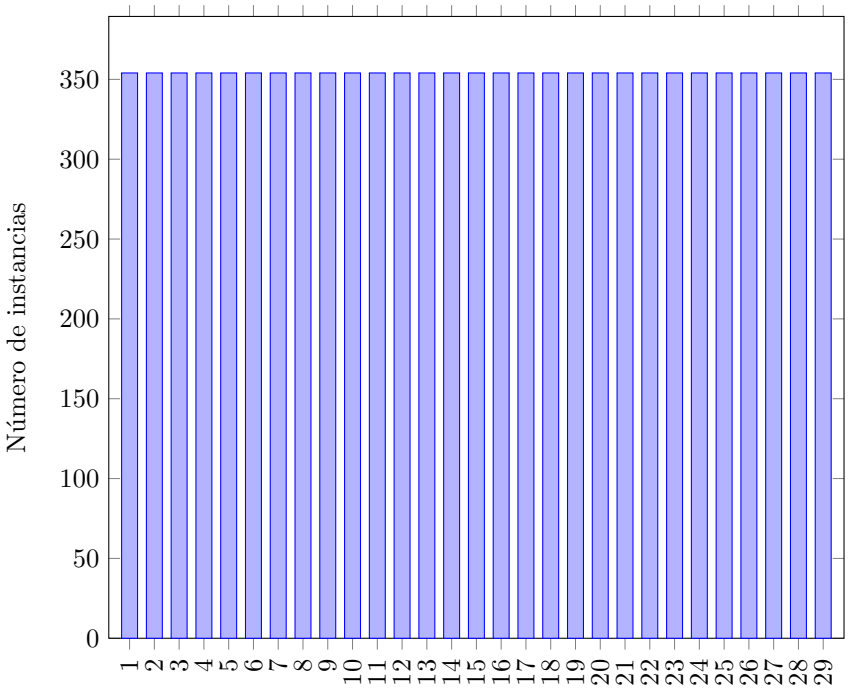


Figura 1: Distribución de clases

Se puede apreciar que este problemas es balanceado, teniendo 354 imágenes para cada clase.

El objetivo del problema es conseguir clasificar correctamente cada tipo de suelo a partir de las imágenes. La dificultad radica en la similitud que hay entre varias clases, en parte debido a la distancia desde la que se toman las fotografías y su baja resolución.

En la Figura 2 se aprecian pares de clases similares, que pueden llegar a ser un problema a la hora del entrenamiento de los modelos por la dificultad al diferenciar las clases.

Se plantea utilizar arquitecturas CNN y ViT definidas en la Sección 3, centrándose en el preprocesamiento de los datos planteadas en la Sección 2 debido a su importancia para obtener buenos resultados. Posteriormente se realizan

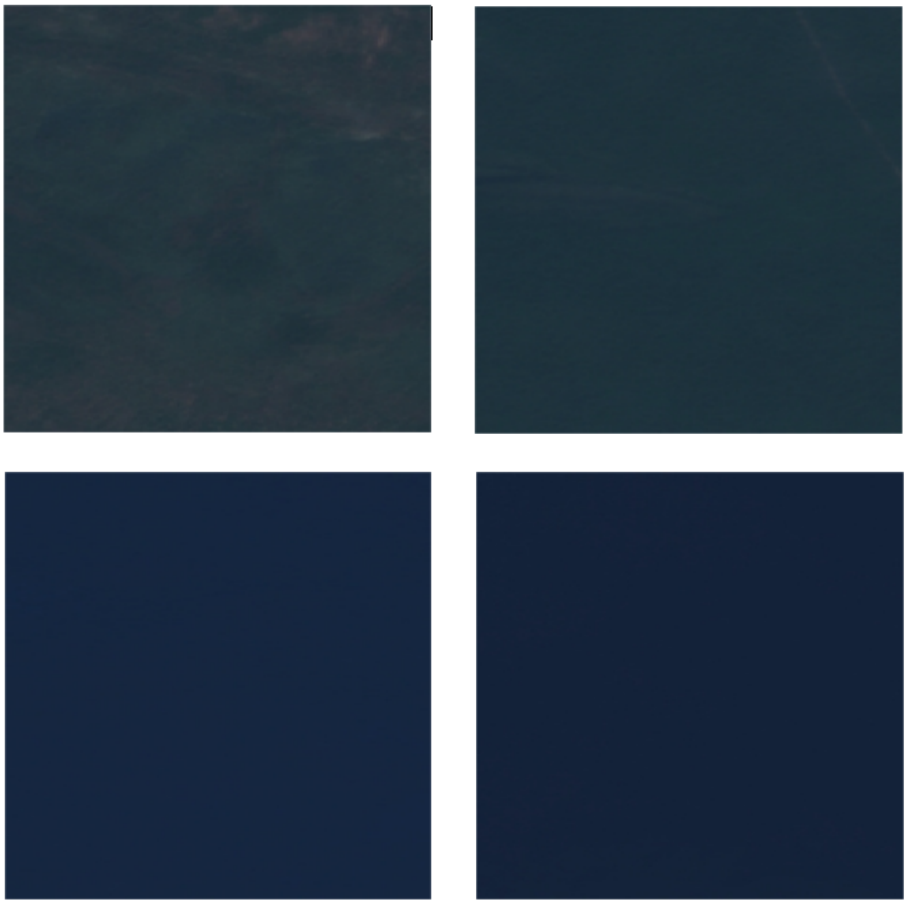


Figura 2: Imágenes parecidas de clases distintas. En la parte superior se encuentra imágenes de la clase ForestsClDeNe y ForestDeDeNe. En la parte inferior las clases WaterBodyMari y WaterBodyCont.

algunas optimizaciones descritas en la Sección 4, terminando con ensembles de los mejores modelos para mejorar los resultados de los modelos base.

2. Preprocesamiento

Dado que se trabajará con arquitecturas que fueron entrenadas con la base de datos ImageNet [2] que tiene mas de un millón de imágenes, y nuestra base de datos es de aproximadamente 10000 imágenes es importante evitar el overfitting, para esto se emplea una secuencia de 5 técnicas diferentes para aumentar el tamaño de nuestra base de datos. La primera etapa del aumento de datos consiste en usar la corrección gamma, esta permite codificar y decodificar valores de luminancia, esta descrita por la ecuación 1, donde I_{in} es la imagen original, I_{out} es la imagen de salida, $gain$ es un factor de ganancia que en nuestro caso siempre se mantiene en 1, y γ es un valor que se elige aleatoriamente en un rango de [0,8, 1,1]

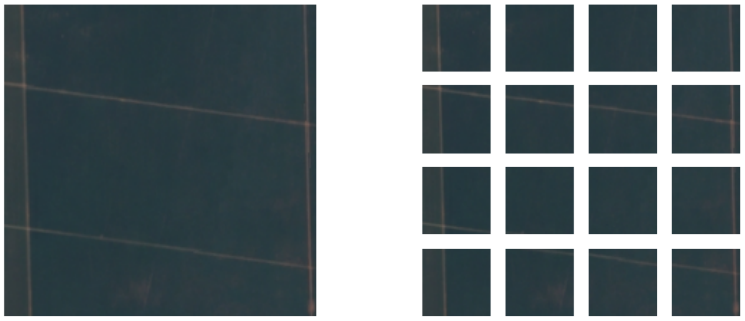
$$I_{out} = 255 * gain \frac{I_{in}}{255}^{\gamma} \tag{1}$$

La idea de usar la corrección gamma es debido a que variar el grado de iluminación en las imágenes permite simular condiciones que no están presentes en todas las imágenes, por lo que nuestra red neuronal se beneficiaría de ver el mismo terreno pero con condiciones de luz similares a las que tendría imágenes de otras superficies. En la segunda etapa de preprocesamiento la imagen es rotada en un rango de $[-45, 45]$ grados, permitiendo que la red neuronal trabaje con terrenos que en otras condiciones pudieron estar rotados, está imagen es después usada en una tercera y cuarta etapa donde se le aplica un flip horizontal y uno vertical, respectivamente, para el caso de los flips, la probabilidad de que se aplique o no es de 0.5. Finalmente, se recorta la imagen de 224×224 a 112×112. El resultado de aplicar este pipeline a una imagen de la base de datos se puede observar en la Figura 3

Figura 3: Pipeline utilizado durante el preprocesamiento.

Uno de los principales problemas a la hora de entrenar modelos es la falta de datos, se propone generar imágenes a partir de las originales para incrementar el tamaño del conjunto de datos original. A continuación se describen las 2 técnicas empleadas:

- **División mediante parches:** Se propone dividir la imagen original en parches, obteniendo el mayor número de imágenes posibles del tamaño deseado. El objetivo es incrementar el tamaño del conjunto de entrenamiento y utilizar la mayor área posible de cada imagen.



(a) (b)

Figura 4: Imagen original (a) de tamaño 224x224 frente a los parches (b) de tamaño 56x56 extraídos de la imagen.

(a) (b) (c)

Figura 5: Image original (a) frente a dos imágenes (b) y (c) generadas mediante el pipeline de preprocesamiento aleatorio.

Esta técnica se puede aplicar tras el pipeline de preprocesamiento mostrado en la sección anterior.

- **Repetición de imágenes:** Se propone repetir varias veces las imágenes del conjunto de entrenamiento, aplicando posteriormente el pipeline de preprocesamiento. Al aplicar transformaciones aleatorias en el preprocesamiento, a partir de una misma imagen se obtienen varias diferentes.

  En la Figura 5 se puede observar que a partir de una imagen original aplicándole transformaciones aleatorias se obtienen imágenes diferentes. Esto permite que se repitan imágenes en el conjunto de entrenamiento, obteniendo una mayor cantidad de imágenes distintas tras aplicar el preprocesamiento.

3. Modelos empleados

En esta sección se incluye una breve descripción de las arquitecturas de los modelos empleados.

3.1. Convolutional Neural Networks

La CNN[8] es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos. Para ello, la CNN contiene una serie de capas ocultas especializadas y con una jerarquía: esto quiere decir que las primeras capas pueden detectar lineas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

Las modelos de clasificación usados se basan en arquitecturas VGG, ResNet, EfficientNet y DenseNet. Esta sección describe las características más relevantes de cada arquitectura. Estos modelos se encuentran disponibles en la librería de TensorFlow[1] o Pytorch[2].

3.1.1. VGG

Las redes VGG son una serie de algoritmos de redes neuronales convolucionales propuestos por el Grupo de Geometría Visual (Visal Geometry Group o VGG) de la Universidad de Oxford, incluyendo VGG-11, VGG-11-LRN, VGG13, VGG-16 y VGG-19. Las redes VGG aseguró el primer lugar en la en la competición del ImageNet Challenge en 2014. Los autores de VGG demuestran que aumentar la profundidad de las redes neuronales puede mejorar el rendimiento final de la red hasta cierto punto.

3.1.2. ResNet

Las Redes Neuronales Residuales o *ResNet*[5] son redes neuronales que utilizan saltos de conexiones o atajos para saltarse algunas capas. Los modelos típicos de ResNet están implementados con doble o triple salto de capas.

Su mayor impacto se debe a que el paradigma ResNet, propuesto por Microsoft, permitió por primera vez entrenar redes muy profundas controlando con éxito el problema del desvanecimiento de los gradientes, y ganó la competición IMAGENET en 2015.

Existe una implementación de ResNet con más pesos que produce una ligera mejora respecto a la arquitectura clásica. En los experimentos realizados se ha optado por el uso de la arquitectura clásica.

[1] `https://www.tensorflow.org/`
[2] `https://pytorch.org/`

3.1.3. EfficientNet

EfficientNet[14] es una arquitectura de red neural convolutiva, que utiliza un método de escalado que escala uniformemente todas las dimensiones de profundidad/ancho/resolución. A diferencia de la práctica convencional que escala arbitrariamente estos factores, el método de escalamiento de EfficientNet escala uniformemente el ancho, la profundidad y la resolución de la red con un conjunto de coeficientes, lo que lleva a obtener mejores resultados de una manera más eficiente que el resto de los modelos.

3.1.4. DenseNet

DenseNet[6] es un tipo de red neural convolutiva que utiliza conexiones densas entre capas, a través de Bloques Densos, donde conectamos todas las capas (con tamaños de mapas de características coincidentes) directamente entre sí. Para preservar la naturaleza de la alimentación, cada capa obtiene entradas adicionales de todas las capas precedentes y pasa sus propios mapas de características a todas las capas subsiguientes.

3.2. EfficientNet + Histogramas

Relacionado a la arquitectura, se busca más información que pueda usarse para la clasificación y no solo usar la imagen preprocesada, para esto se emplean los histogramas de color de la imagen, aunque no se obtiene una buena precisión clasificando solo con los histogramas, pues como ya se mencionó, algunas imágenes de clases diferentes son muy similares a otras, si es posible obtener una buena precisión haciendo que una red de histogramas y una de imágenes trabajen juntas. Para esto se usa la arquitectura EfficientNet-B0, que ya tiene una alta precisión en este dataset, y su capa de salida se cambia para que tenga 100 neuronas, paralelo a esto, los histogramas de color se pasan por una red compuesta de 3 capas de neuronas densas con 128, 64 y 32 neuronas, la salida de la red EfficientNet y la red de histogramas se concatenan, generando un descriptor de 132 elementos que se manda a una capa con 29 neuronas que se usan para clasificar. Esta arquitectura puede observarse en la Figura 6, las dimensiones del batch de imágenes son $b \times c \times h \times w$, donde b es el tamaño del batch, c el número de canales de la imagen, h y w el largo y ancho de la imagen respectivamente.

Para entrenar la red que usa EfficientNet-B0 se parte de una red preentrenada en ImageNet, mientras que la red de histogramas se inicializa con valores aleatorios entre 0 y 1, dando mejores resultados que la inicialización Xavier descrita en [4], en ambas arquitecturas se entrenan todos los parámetros usando un tamaño de batch de 128. Las etiquetas de las diferentes clases se transforman a one hot label y se usa entropía cruzada para calcular el error. El método que se usa como ley de aprendizaje es Adam [7], también se explora Gradiente Descendente Estocástico pero se obtiene una menor precisión.
En un principio se entrena dejando fijo el learning rate pero al final se adopta

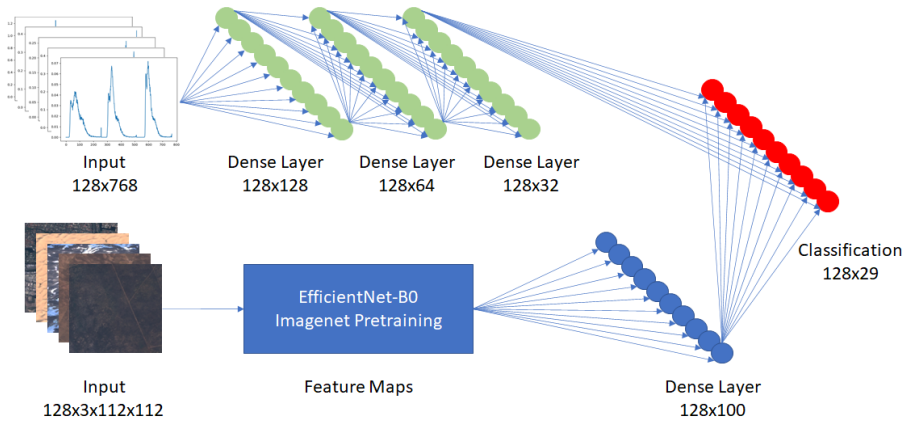


Figura 6: Arquitectura usada para la clasificación, como entradas un batch de imágenes de color y sus respectivos histogramas de color.

una metodología para variar el learning rate que es similar a la usada por [10] pues se obtienen mejores resultados, es importante mencionar que en nuestro caso no se reinicia el learning rate, pero si se parte de un tasa de aprendizaje muy baja que después se incrementa a un máximo de 0.0003, todo esto en las primeras 20 épocas, para posteriormente ir decreciendo hasta prácticamente convertirse en 0 y siguiendo un decaimiento cosenoidal durante 170 épocas, resultando en un entrenamiento de 190 épocas, en la Figura 7 se puede ver el comportamiento que tiene la tasa de aprendizaje durante todas las épocas del entrenamiento.
Para mejorar la precisión en la clasificación se varían los parámetros de la red de histogramas y de la basada en EfficientNet, aumentando de tamaño el vector que se usa como descriptor para la clasificación, también se explora aumentar el número de capas en la red de histogramas pero nada de esto mejora la precisión. Otra idea que es desechada por no incrementar la precisión en la tarea, es generar el histograma de color como una imagen de 3×256 con un solo canal y usar capas convolucionales para generar el descriptor al que también se le varía su tamaño.
Por último, aunque en el entrenamiento se usan imágenes de 112×112 en la evaluación se usa la imagen original de 224×224, sin hacer ningún preprocesamiento.

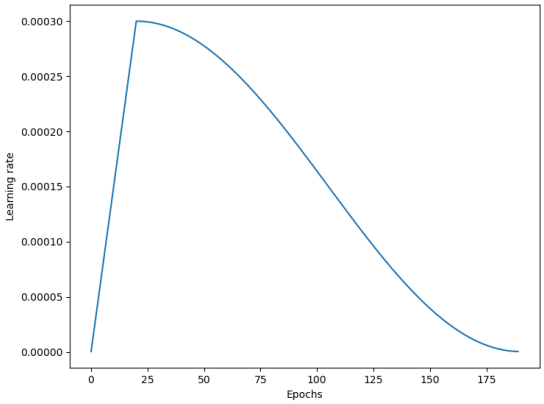


Figura 7: Variaciones en el learning rate siguiendo la metodología Cosine Annealing with warmup.

3.3. Las redes de atención, Transformers

3.3.1. Introducción

Se ha realizado una implementación basada en las redes neuronales de Transformers. Así como también de las redes convolucionales.

3.3.2. Objetivos

No nos centraremos tanto en conseguir fine-tuning adecuado para este dataset, dado una arquitectura de red neuronal, sino más bien en conseguir que el modelo rinda medianamente bien, por encima del 0.7 de precisión en test nos bastará, en el caso de los Transformers. Esto se debe, principalmente a varios motivos. Por un lado, no disponemos de un modelo ya entrenado con los pesos como el caso de las CNNs con imagenet, por otro lado, entrenarlo con más precisión exige bastante coste computacional, más que las Cnns, y por último, ya disponemos de resultados excelentes en el grupo. Por lo que, no es necesario centrarse más en mejorar los modelos con una alta precisión 0.99.

Para el caso de las redes convolucionales se entrenará algunos modelos con el fin de establecer una comparación entre ambas redes y observar para está tarea específica cual es que rinde mejor. Se intenta, por ello establecer unas condiciones equivalentes para ambos casos, en la medida de que sea posible (los Transformers, tardan más entrenar por lo que reduciré las épocas en las ocasiones que sea computacionalmente muy costoso). Por lo que no usaremos el transfer learning de las distintas arquitecturas en el caso de las Cnns.

3.3.3. Fundamentos teóricos de las redes de transformers

La arquitectura de las redes convolucionales y de las redes neuronales de Transformers son muy similares, pero su funcionamiento y la forma que tienen de extraer rasgos en la imagen son diferentes. Mientras que en las redes convolucionales se emplean unos filtros, capas convolucionales, pooling etc. Para obtener unos rasgos de la imagen de forma local para luego aplanarla e insertarlo a una multicapa perceptrón. Los Transformers dividen la imagen en cuadrículas para aplanarlo y más tarde proyectarlo sobre una dimensión superior (embedding) donde se puede obtener los rasgos de las cuadrículas. El resultado se introduce en una capa perceptrón. Dado que los Transformers se usan principalmente en NLP, un símil aquí se podría considerar básicamente que dividen la imagen en parches, le asigna un conjunto de etiquetas numéricas tantas como la dimensión que se establezca, e introducirlo al Transformers encoder, de esta forma se establece la relación entre los distintos parches de las imágenes. Se puede observar en la siguiente imagen:

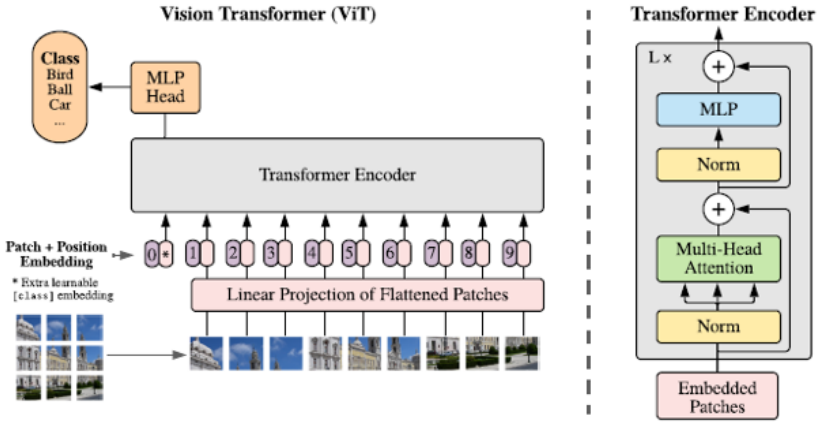


Figura 8: Arquitectura de las redes transformers en visión por computador. Tomada de [3]

Se ha usado dos tipos de redes Transformers. Por un lado, las ViT, descrita en el párrafo anterior, y por otro las Swin Transformers [9] uno mucho más reciente. Son prácticamente similares salvo un detalle y es que la segmentación de las imágenes son variables como se observa en la siguiente imagen. Lo que hace que el mecanismo de atención se centre en los distintos segmentos a lo largo del proceso.

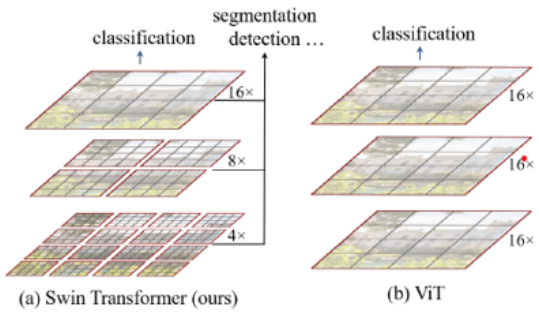


Figura 9: Una segmentación variable en las Swin transformers. Tomada de [9]

3.4. OVA

Un ensamble OVA consiste en tantos modelos como clases tenga el problema. Cada uno de estos modelos es binario, especializado en una clase en concreto. Al predecir la clase de una instancia, se ejecutan todos los modelos binarios con dicha, que predicen una salida real entre 0 y 1 para dicha instancia. Hay distintos métodos para hallar la clase final a partir de las salidas de los modelos binarios, pero la más común y la usada en este trabajo consiste en elegir la clase cuyo modelo de el valor más alto de la salida.

3.4.1. Ensamble OVA como problema no balanceado

Un problema que comúnmente surge a en el entrenamiento de los modelos binarios del ensamble OVA es un desequilibrio en el número de observaciones que representan a cada clase. En el problema multi-clase, las 29 clases están casi perfectamente balanceadas: 354 ejemplos por clase, salvo la clase 1, con 355, y la clase 21, con 353). Esto implica que, tomando todos los datos, para un problema binario que discrimine si se trata de una clase específica o no, el *imbalance ratio* es prácticamente de 28/1.

Las redes convolucionales, al igual que la mayoría de algoritmos basados en redes neuronales, no se adaptan automáticamente a esta casuística. Por este motivo, deben aplicarse soluciones de clasificación no balanceada para obtener un modelo binario y un ensemble OVA eficientes.

De las posibles soluciones, aquellas que consideramos más bondadosas son las siguientes:

- Random Undersampling. Eliminar imágenes aleatoriamente de la clase positiva, preservando el balance entre el número de imágenes eliminadas de cada clase original del problema. Un beneficio de esta solución sería la reducción del coste computacional. Por otro lado, para producir un balance cercano a 1, habría que eliminar un gran número de imágenes de cada clase original, pasando de 353-355 a 12 o 13.

- Modificar la función de coste. Llamaremos función de coste a la suma de los valores de la función de pérdidas para cada instancia de entrenamiento (que esté incluida en el *batch* del paso o *step* actual). La función de coste en el problema multi-clase, era la suma de entropías cruzadas para cada instancia la entropía cruzada. La solución de modificar la función de coste para el problema no balanceado consistiría en dar mayor peso a las pérdidas de las instancias de la clase positiva. Esta solución tiene el mismo efecto que un oversampling de todas las instancias de la clase minoritaria. El peso se elige en cada modelo para que el ratio entre la clase negativa y positiva sea 1/1.

Aunque la segunda solución no alivia la carga computacional, se eligió esta para evitar la pérdida de información.

3.4.2. La necesidad de una predicción positiva segura

El hecho de que la clase predicha por el ensamble OVA se elija en función del modelo que de la salida más alta implica que no sólo se necesitan modelos binarios que predigan bien, sino que también es necesario que, si se trata de la clase positiva, la salida sea lo más cercana posible a 1.

Para ello, hubiera sido interesante probar el resultado de usar una función de pérdidas como el AUCROC. Maximizarla significaría que para cualquier umbral que separe entre una predicción positiva o negativa, la precisión es máxima. Esto implicaría que la clase negativa se predice siempre con salida 0,0 y que la clase positiva se predice siempre con salida 1,0. Sin embargo, el carácter no derivable del AUCROC imposibilita elegirlo como función de pérdidas, ya que esta es un requerimiento para la retropropagación.

Finalmente, se eligió como función de pérdidas la entropía cruzada, ya que la penalización que asigna también hace que los pesos se ajusten acercando las predicciones a los valores extremos (0.0 y 1.0), según la expresión siguiente, donde y_i es la etiqueta binaria de la observación y p_i es la predicción para la observación (número real):

$$y_i ln(p_i) + (1 - y_i) ln(1 - p_i) \tag{2}$$

3.4.3. Otras especificaciones del experimento

La arquitectura y preprocesamiento escogidos para los modelos binarios son aquellos para los que se obtenía mejor precisión hasta el momento: el pipeline de preprocesamiento descrito en la Figura 2, learning rate de 0.0003 y se incluyen los histogramas.

El número de épocas escogidas fue relativamente bajo, de 60, para aliviar la carga computacional y permitir el entrenamiento de 29 modelos en un tiempo abarcable.

3.4.4. Trabajo futuro

Otras posibilidades que podrían haberse hecho son las siguientes:

- Undersampling en lugar de modificar los pesos, reduciendo así el coste computacional y permitiendo un mayor número de épocas.
- Cambiar la forma de elegir la clase predicha por el ensemble. Por ejemplo, podría haberse tenido en cuenta, no sólo la clase para la cual la salida es mayor, sino también cuál es la salida que ha dado un modelo de una clase parecida. De esta forma, podría tenerse en cuenta una agregación de las salidas de clases parecidas, como la media.

3.5. Unión y separación de etiquetas

Otra de las arquitecturas exploradas se basa en la idea de que existen clases muy parecidas entre sí. Por ejemplo, existen clases distintas para extensiones de agua (que tienen un uso distinto del terreno).

Esto sugiere que podría ser más eficiente entrenar un modelo que, en lugar de aprender clases parecidas por separado, aprenda una clase genérica que abarque a ambas y, por separado, un modelo que distinga únicamente entre estas clases parecidas.

Los beneficios potenciales de esta unión y posterior discriminación de clases son dos:

- Capacidad de entrenar modelos por separado que discriminen adecuadamente clases que son difíciles de discriminar
- El modelo principal puede centrarse en aprender a diferenciar clases que no son tan parecidas entre sí y realizar mejor esta tarea

3.5.1. Otras especificaciones del experimento

Para seleccionar las clases a unir, se visualizó la matriz de confusión de la Fig. 11, obtenida mediante validación cruzada con 5 pliegues o *folds*.

Las clases que se decidió unir son las que más se confundieron en validación cruzada: 10-11 y 21-22.

El preprocesamiento y la arquitectura elegida tanto para los modelos binarios como para el modelo principal fueron aquellos para los que se obtuvo mejor precisión hasta la fecha: se usa el pipeline de preprocesamiento descrito en la Figura 2, con learning rate de 0.0003 y se incluyen los histogramas.

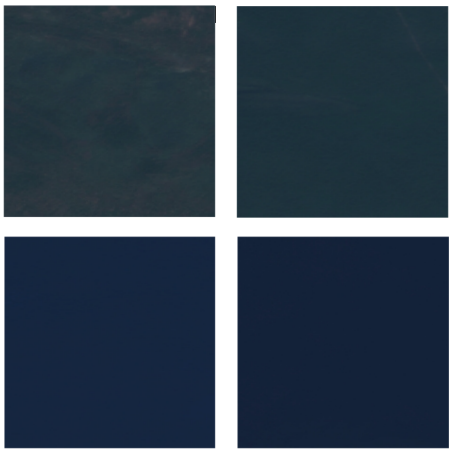


Figura 10: Imágenes parecidas de clases distintas. Arriba, las imágenes son de clase 10 y 11, o ForestsClDeNe y ForestDeDeNe. Abajo, son de clase 21 y 22, o WaterBodyMari y WaterBodyCont.

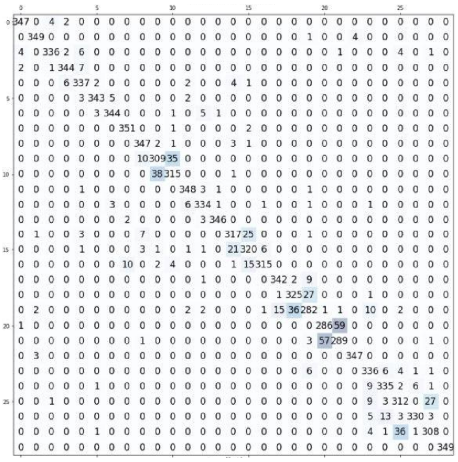


Figura 11: Matriz de confusión obtenida mediante validación cruzada. Los colores del fondo resaltan los números fuera de la diagonal. Los índices están en notación de 0 a 28. Eje y: etiquetas. Eje x: predicciones.

4. Optimizaciones y pos-procesado

En esta sección se describen las optimizaciones y el pos-procesado realizado a los modelos, con el objetivo de intentar mejorar los resultados de los modelos.

4.1. Rotación aleatoria

Como se ha explicado en la Sección 2, a las imágenes se les han aplicado transformaciones aleatorias, entre ellas la rotación.

La rotación añade un borde negro, que puede influir en el entrenamiento si se recorta la imagen en esa zona. Para evitar este caso se recorta la imagen tras la rotación, obteniendo el rectángulo de área máxima sin bordes negro. Para conseguir este resultado se emplea la librería AugLy[11], con una implementación compatible con Pytorch.

(a) (b) (c)

Figura 12: Image original (a) frente a la imagen rotada (b) 25 grados como se describe en el preprocesamiento y la imagen rotada (c) 25 grados y posteriormente recortada.

Con esto se evita añadir imágenes de mala calidad con trozos en negro, viendo una ligera mejora de resultado pasando de 98,76 % a 98,89 % en test. Esto demuestra que eliminar el borde negro ayuda a obtener un modelo ligeramente mejor al eliminar imágenes de mala calidad.

4.2. Evaluación mediante parches

En el momento de clasificar una imagen, esta es dividida en parches tal y como se indica en el proceso descrito en la Sección 2. Se obtiene una predicción de cada parche utilizando una única red, siendo utilizadas estas predicciones para obtener la etiqueta final de la imagen.

Se han probado 2 formas de obtener la etiqueta final:

- **Voto:** Se obtiene la etiqueta para cada parche y se realiza un voto por mayoría.
- **Suma:** Se suman las probabilidades de cada parche y se obtiene como etiqueta el que mayor valor tenga.

Realizar la suma de las probabilidades beneficia a las situaciones en las que duda en bastantes parches, obteniendo probabilidades muy similares para varias clases.

Con esta forma de evaluar las imágenes se pretende obtener un modelo más robusto.

4.3. Batch learning en últimas épocas

Los beneficios del *Mini-batch learning* frente al *Batch learning* o *Stochastic Gradient Descent* (tamaño de batch = 1) son conocidos: favorece la convergencia a un mejor óptimo y una convergencia más rápida. Sin embargo, el hecho de que en cada paso de entrenamiento se actualizan los pesos teniendo en cuenta sólo las imágenes de un *batch*, presenta un problema desde el punto de vista teórico: las actualizaciones de los pesos no están teniendo en cuenta a todo el conjunto, de forma que se puede estar aprendiendo en cada paso un clasificador que mejora las pérdidas para una parte de las observaciones, pero no para otra. Intuitivamente, esto es es especialmente relevante cuando se llega a las últimas etapas del entrenamiento. Para ilustrarlo, regurrimos a la Fig. 13.

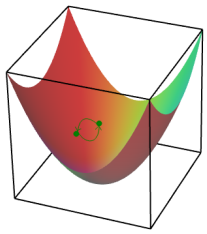


Figura 13: Pérdidas de la red convolucional para los distintos puntos del hiperespacio de los pesos. En verde, los puntos a los que podría llegarse al actualizar los pesos en cada step.

Como se muestra en la figura, podría ocurrir que mediante las actualizaciones de los pesos en los distintos pasos no se llegara a la convergencia global, ya que, para las imágenes distintas que hay en distintos *batch*, los intereses de actualización de los pesos son distintos. Por ello, podría ser positivo que, en las últimas épocas, el *batch* fuera el conjunto completo de las imágenes (también conocido como *Batch learning*).

4.3.1. Dificultades en la implementación

Para implementar este experimento, no se pudo cambiar el tamaño del *batch* en las últimas épocas, ya que tanto *Tensorflow* como *Pytorch* hubieran intentado cargar todas las imágenes del *batch* en la GPU, desbordando la memoria. En cambio, la solución consistió en guardar el valor de las funciones de pérdidas de cada *step* o paso y, tras computar las pérdidas del último paso, actualizar los pesos teniendo en cuenta todas las pérdidas.

4.3.2. Especificaciones del experimento

La configuración de arquitectura y preprocesamiento elegida para el experimento fue aquella mediante la cual se había obtenido la mejor precisión hasta el momento: el pipeline de preprocesamiento descrito en la Figura 2, con learning rate de 0.0003 y se incluyen los histogramas.

El batch learning se aplicó a partir de la época 181 y hasta la 190 incluída (última época).

5. Estudios realizados

En esta sección se describen diversos estudios realizados. Entre estos estudios se encuentra la comparación de arquitecturas básicas, la generación de imágenes para data augmentation y el tamaño de entrada de la red.

5.1. Comparación de arquitecturas CNN

En este estudio se comparan las arquitecturas descritas en la Sección 3.1 con el objetivo de escoger la mejor arquitectura de CNN para este problema.

Dentro de cada arquitectura se ha seleccionado un modelo pre-entrenado con Imagenet. A continuación se enumeran los modelos seleccionados:

- VGG16
- ResNet50
- DenseNet169
- EfficientNet B0

Para el entrenamiento se emplea un tamaño de entrada de 224x224 píxeles, utilizando un 80 % de los datos para entrenamiento y el 20 % para validación. Los modelos se han entrenado con un tamaño de batch de 16 imágenes durante 50 épocas utilizando el optimizador Adam por defecto. Adicionalmente se ha optado por entrenar todas las capas de la red.

Modelo	Precisión Train	Precisión Val
VGG16	0.890	0.595
ResNet50	0.980	0.663
DenseNet169	0.958	0.604
EfficientNet B0	0.978	0.758

Cuadro 1: Precisión de los modelos CNN sobre el conjunto de entrenamiento y de validación.

En el Cuadro 1 se puede observar que todos los modelos realizan bastante sobreajuste sobre los datos de entrenamiento, obteniendo los mejores resultados

en el conjunto de validación para el modelo EfficientNet B0. A partir de estos resultados, el resto de experimentos se centran en la arquitectura EfficientNet para modelos CNN.

5.2. Comparación entre generación de imágenes

En este estudio se realiza una comparación entre los métodos de generación de imágenes descritos en la Sección 2. Entre estos métodos se emplea la generación mediante parches y la repetición de imágenes originales.

Como modelo se emplea EfficientNet B0 + histogramas, con una entrada de tamaño 112x112.

Modelo	Precisión Test
Base	0.9901
Parches	0.9833
Repetir	0.9876

Cuadro 2: Resultados al añadir imágenes al conjunto de entrenamiento en test.

En la Tabla 2 se puede observar que los mejores resultados se obtienen con el conjunto de datos de entrenamiento original tras aplicarle el pipeline de preprocesamiento.

5.3. Comparación entre tamaño de entrada

En este estudio se compara el tamaño de entrada, evaluando el modelo con la metodología descrita en la Sección 4.2. El objetivo principal de este estudio es ver cómo afecta el tamaño de los parches al resultado.

En la Tabla 3 se puede observar que se obtienen mejores resultados con un tamaño mayor de imagen, a pesar de evaluar mediante un voto de todos los parches de la imagen. Esto indica que cuanto menor sea el tamaño de entrada es más fácil es de que se confunda con otra clase.

Modelo	Precisión
Entrada 112	0.9901
Entrada 56	0.9895
Entrada 32	0.9703

Cuadro 3: Resultados obtenidos de entrenar la arquitectura EfficientNet B0 + histogramas con el pipeline de preprocesamiento.

6. Resultados

Los resultados obtenidos se pueden resumir en el Cuadro 4 y de manera gráfica en la Figura 14, donde:

- **Experimento 1**, Es la arquitectura EfficientNet-B0 usando el pipeline de preprocesamiento sin corrección gamma, con learning rate de 0.001 y random crop de 128×128.
- **Experimento 2**, Se usa todo el pipeline de preprocesamiento, con learning rate de 0.001 y random crop de 128×128.
- **Experimento 3**, Se usa todo el pipeline de preprocesamiento, con learning rate de 0.0003 y random crop de 112×112.
- **Experimento 4**, Se usa el pipeline de preprocesamiento descrito en la Figura 3, con learning rate de 0.0003 y se incluyen los histogramas.
- **Experimento 5**, Se usa el pipeline de preprocesamiento descrito en la Figura 3, con learning rate de 0.0003, inicialización Xavier en la red de histogramas y en las últimas épocas se usa la metodología de la sección 4.3.
- **Experimento 6**, Se usa el pipeline de preprocesamiento descrito en la Figura 3, con learning rate de 0.0003 y en las últimas épocas se usa la metodología 4.3.
- **Experimento 7**, Se usa el pipeline de preprocesamiento descrito en la Figura 3, con learning rate de 0.0003, inicialización Xavier en la red de histogramas, en las últimas épocas se usa la metodología de la sección 4.3 y se hace un ensemble con 3 modelos.
- **Experimento 8**, Se usa el modelo generado en el Experimento 6 y solo se entrena la red de histogramas cambiando la corrección gamma para usar valores de gamma entre [0,9, 1,3].
- **Experimento 9**, Ensemble OVA.
- **Experimento 10**, Unión y separación de clases.
- **Experimento 11**, Visión Transformers, la configuración se indica más adelante
- **Experimento 12**, Visión Transformers, se cambia la configuración de algunos hiper-parámetros
- **Experimento 13**, Swin Transformers.
- **Experimento 14**, Swin Transformers.
- **Experimento 15**, Visión Transformers.

- **Experimento 16**, Ensemble de los modelos anteriores experimentos 13,14 y 15 por voto mayoritario.
- **Experimento 17**, Usando la misma arquitectura a la del Experimento 4, se entrena un modelo con entrada 112x112 y otro con entrada 56x56. Utilizando la metodología explicada en la Sección 4.2 para evaluar los modelos utilizando parches del mismo tamaño a su entrada. Se realiza un Ensemble entre estos dos modelos y el modelo del Experimento 8, siendo el modelo con mejores resultados.
- **Experimento 18**, Se realiza un ensemble de 5 modelos, entre ellos los 3 modelos del Experimento 17. Se entrena un modelo siguiendo el preprocesamiento para generar imágenes mediante parches, con entrada de tamaño 56x56 y evaluando con la metodología descrita en 4.2. Como quinto modelo, se repite el mejor modelo para añadirle un mayor peso a sus resultados.

Cuadro 4: Resultados obtenidos en la competición

Experimento	LB público	LB privado
Experimento 1	0.96597	0.97530
Experimento 2	0.97731	0.97839
Experimento 3	0.98453	0.98148
Experimento 4	0.98969	0.98611
Experimento 5	0.98969	0.99074
Experimento 6	0.99072	0.98919
Experimento 7	0.99175	0.99228
Experimento 8	0.99175	0.99537
Experimento 9	0.70679	0.72061
Experimento 10	0.67901	0.68556
Experimento 11	0.73402	0.73919
Experimento 12	0.77268	0.76851
Experimento 13	0.76507	0.74074
Experimento 14	0.70824	0.74074
Experimento 15	0.76185	0.76851
Experimento 16	0.78865	0.77932
Experimento 17	0.99537	0.99381
Experimento 18	0.99537	0.99484

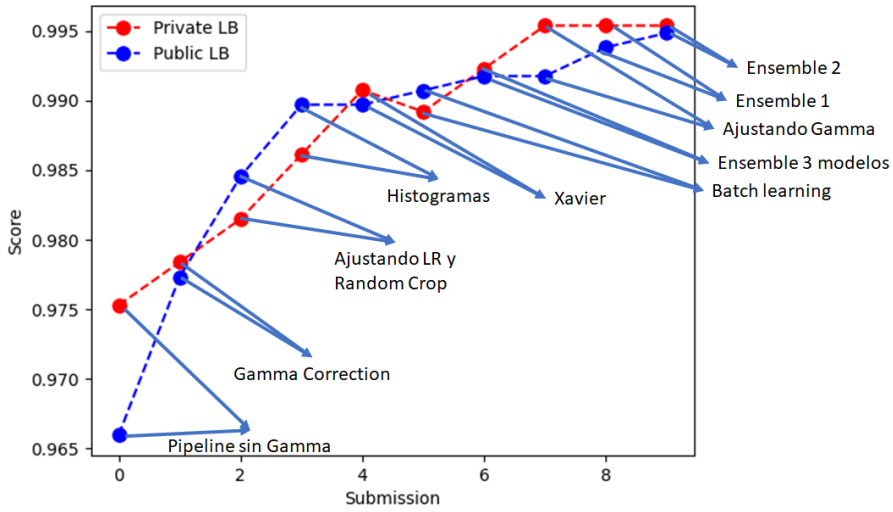


Figura 14: Resultados obtenidos en la plataforma Kaggle.

6.1. Resultados con los transformers y comparativa con las CNNs

6.1.1. Preprocesamiento en los transformers

Se realiza los siguientes procedimientos para el aumento de datos.

- Rotaciones de 90 a cada imagen, con numpy (np.rot90()) tras transformarlo en un tensor (decodificación)
- Añadido capas de crop, volteo horizontal y vertical, y una pequeña rotación. Al modelo con las capas.

6.1.2. Preprocesamiento en las redes convolucionales

Con respecto a los modelos CNN . El procesamiento se realiza a cabo usando los parámetros de la cabecera en ImageDataGenerator de Keras. Por ello se organiza la carpeta en los datos de test de igual forma que en los datos de training.

- Volteos horizontales
- Volteos verticales
- Cambios en las rotaciones de forma aleatoria desde 0^{o} hasta 360^{o}
- Aumentos en el brillo

6.1.3. Visión Transformers

Seguiremos las indicaciones que vienen descritas en [13] que está basada en el artículo [9]. Los resultados se muestran en la tabla anterior, para el caso de los transformers tan solo se muestra la configuración usada.

Experimento 11:
Los hiperparámetros más importantes

- AdamW, del módulo de tensorflow addons con un learning rate 0.0003.
- Embedding dim 40
- Épocas 70 Se disminuye por su coste computacional. Además, se observa de las resultados anteriores que la progresión es bastante menor a partir de 20 o 30 epochs.
- Attention head 5.
- Validation split = 0.1

Experimento 12:

- Learning rate 0.0007
- Embedding dim 32.
- Épocas 50

Experimento 15:

- Learning rate 0.0007
- Embedding dim 32.
- Épocas 15

6.1.4. Swin Transformers

Dado que la arquitectura ya está implementada por el autor [1], basada en los articulos [9]. Previamente cambiamos los datos y los preprocesamos con data aumentation, al igual que en los visión Transformers. Se centrará más en cambiar los hiperparámetros correspondientes:

Experimento 13:
Los hiperparámetros más importantes

- AdamW, del módulo de tensorflow addons con un learning rate 0.0003.
- Embedding dim 32.
- Épocas 200.
- Attention head 8.

- Batch size 32.
- Patch size (2,2)

Experimento 14:

- Learning rate 0.0007
- Embedding dim 32.
- Épocas 50

Experimento 16 modelo de ensemble: El modelo se realiza con tomando los 3 mejores resultados anteriores y realizando un voto por mayoria.

6.1.5. Redes convolucionales

Nos centraremos en entrenar los modelos como comentamos antes, desde 0 para realizar un estudio comparativo entre ambos modelos de forma más equitativa. Se ha experimentado con las arquitecturas de EfficientNet, DenseNet, Resnet etc. Se ha lanzado varios modelos experimentales **con los pesos iniciados aleatoriamente**. para posteriormente realizar una comparación con los modelos anteriores. Se guarda los modelos que mejores resultados ofrezcan durante la validación de cada arquitectura. Probaremos diferentes arquitecturas, DenseNet, EfficientNetB0, EfficientNetb1, Resnet e InceptionV3. Dichos resultados no se subieron a kaggle, pues no es el objetivo aquí, como mencionamos anteriormente sino buscar una comparativa entre ambas redes neuronales.

Experimento 1: Empleamos EfficientNetb0. Se ha observado que el entrenamiento en general es más rápido respecto a los modelos de los Transformers. Esto se debe a que las convolucionales no proyectan cada parche sobre un espacio dimensional superior como el caso de los transformers, con lo que tiene muchos menos más datos que comparar. Por otro lado, data aumentation no se ha hecho con las rotaciones desde numpy con rot90.

Los datos de validación-evaluación (no pasan por el entrenamiento) se emplearán sobre el conjunto de test. Así, ya dispondremos del rendimiento real del modelo durante el entrenamiento y no es necesario particionar los datos de train.

Fine tuning:

- Optimizador Adam con un learning rate 0.0003.
- Épocas 200. Hice varias ejecuciones con entrenamientos pequeños. En los Cnns guardaba el modelo. Por lo que podía retomar el entrenamiento, sin embargo, olvidé poner las gráficas, durante este periodo de entrenamiento por lo que, los resultados que se muestran corresponden a épocas posteriores de 200.
- Se guarda el mejor modelo durante el entrenamiento en lugar de realizar un early stopping.

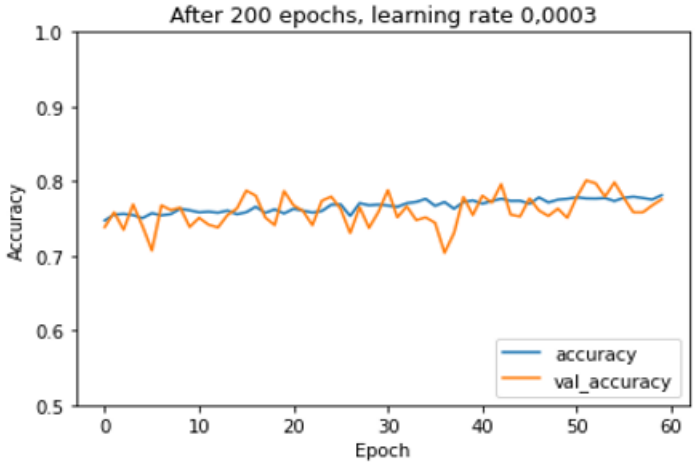


Figura 15: Experimento 1. Resultados con EfficientNet0

- 0.8 de accuracy máximo obtenido

Experimento 2: Empleamos DenseNet 121. En está ocasión las épocas corresponden desde 0.

- Preprocesamiento de imágenes igual que en el apartado anterior.
- Fine-Tuning igual que en el modelo anterior salvo que cambiamos el número de épocas épocas 150
- 0.79 de accuracy máximo obtenido

Experimento 3: Empleamos Efficient Net b2. La configuración es idéntica al apartado anterior:

- 0.7 de accuracy máximo obtenido

Experimento 4: Empleamos ahora Inception V3, la configuración es idéntica:

- 0.82 de accuracy máximo obtenido

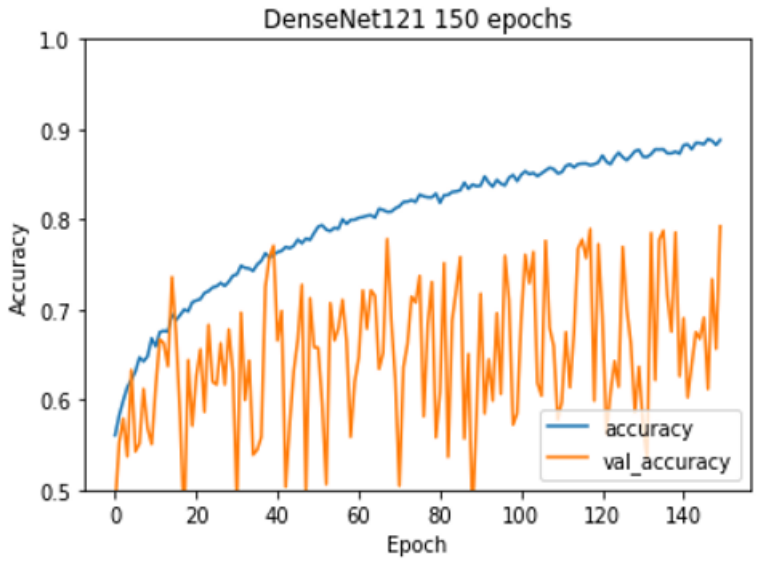


Figura 16: Experimento2. Resultados con DenseNet 121

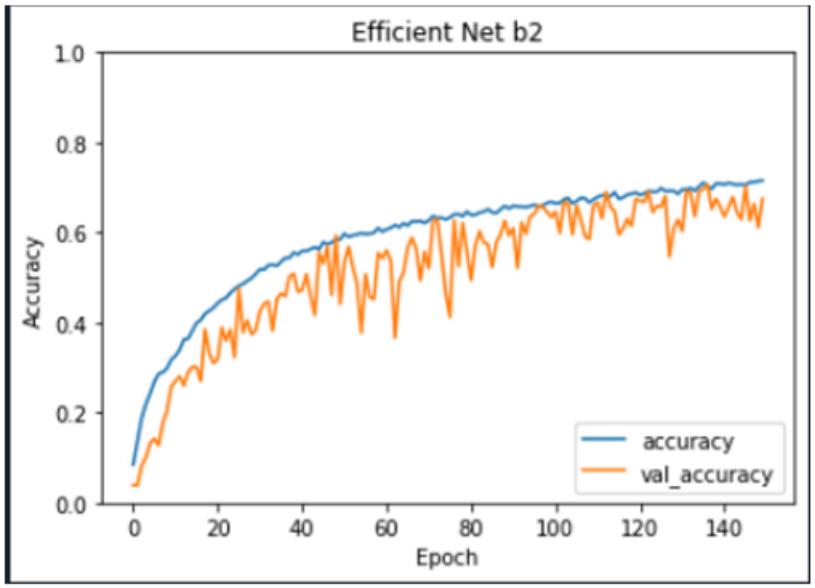


Figura 17: Experimento 3. Resultados con Efficient Net b2

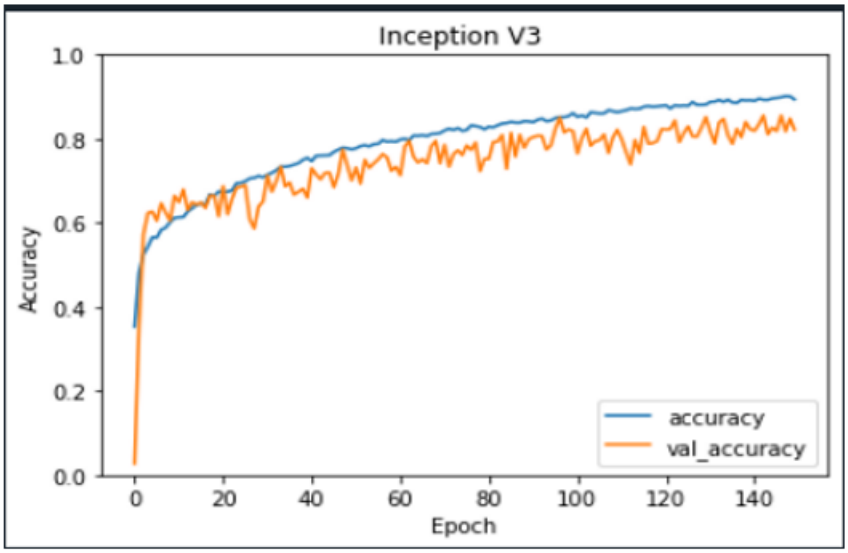


Figura 18: Experimento4. Resultados con Inception V3

7. Conclusiones

Algo interesante que se puede apreciar en el preprocesamiento es que no es necesario usar toda la imagen para entrenar, y como se ve en el resultado de los experimentos, una imagen más pequeña obtiene una precisión más alta en la tarea, además de que permite generar una base de datos mucho mayor ya que de una imagen original de 224×224 se podrían obtener 4 imágenes diferentes, suponiendo que no hay solapamiento entre las imágenes generadas.

Una vez que se parte de una arquitectura robusta que es capaz de lograr una buena precisión, incluir más características de la imagen permitió incrementar la precisión en la tarea de clasificación, si bien algunas de las clases eran similares y solo usar histogramas hubiera resultado en una precisión muy baja, aprovechar los descriptores generados por EfficientNet y la red de histogramas permitió casi alcanzar el 99 % de precisión.

Por último, se observa el potencial de los ensembles, permitiendo mejorar los resultados de los modelos individuales y alcanzar un resultado superior al 99 %. Cabe destacar la importancia de la diversidad en resultados de los modelos al realizar un ensemble, dependiendo bastante de la calidad de los modelos individuales e intentar que cada modelo usado sea lo suficientemente distintos para potenciar la puntuación.

7.1. Conclusiones de los resultados Transformers vs Cnns

Según las distintas literaturas véase [12], los Transformers son modelos que requieren muchos más datos que las redes convolucionales, para entrenarse bien. Además, se ha observado que, al proyectar sobre un espacio dimensional superior para cada parche de la imagen aumenta significativamente el coste computacional, por lo que necesita más recursos. Por el contrario, a diferencia de las Cnns que usan filtros (convolución) sobre una región de la imagen y por tanto solo extrae características de forma local en la imagen, los Transformers pueden extraer características globales, al tener dividir la imagen y proyectarlo (embedding) sobre un espacio [12]. Esto podría ser importante especialmente en esta práctica, ya que las imágenes por satélites a simple vista son más homogéneas (etiquetas 1) y podrían requerir extraer características globales en lugar de características locales.

A raíz de los resultados anteriores obtenidos se observa, que, pese a que los Transformers requieran más cómputo, en apenas pocas épocas logra la convergencia de los resultados más rápido que en el caso de los Cnns en los datos de training. En torno a $10 - 15$ épocas a partir de allí los resultados mejoran levemente. Por el contrario, los modelos convolucionales tardan más en mejorar el modelo en torno 50-60 épocas apenas se observa cambios en la pendiente.

Posiblemente esto se deba a que el número de parámetros es significativamente menor, es decir hay menos pesos que ajustar en el modelo en el caso de los Transformers, comparado con los modelos convolucionales como EfficientNet. Por lo que no tendremos en cuenta este factor en la comparación.

En general se podría concluir que los rendimientos de ambas redes son muy

similares, aunque notemos que los Transformers, requieren de un coste computacional alto, en compensación introducen algunas ligeras mejoras como hemos mencionado anteriormente. También, hay que comentar que el diseño estas redes en un principio no se pretendía usar en visión por computador, sino en el procesamiento del lenguaje natural.

8. Trabajo realizado

En esta sección se describe el trabajo realizado por cada integrante del equipo.

1. **Introducción:** David Herrera Poyatos
2. **Preprocesamiento:** Adán Cruz Vargas y David Herrera Poyatos
3. **Modelos empleados**
   a) **Convolutional Neural Networks:** David Herrera Poyatos
   b) **EfficientNet + Histogramas:** Adán Cruz Vargas
   c) **OVA:** Andrés Herrera Espino
   d) **Unión y separación de etiquetas:** Andrés Herrera Espino
   e) **Batch learning en últimas épocas:** Andrés Herrera Espino
   f) **Las redes de atención, Transformers** Zewei Wang Chen
4. **Optimizaciones y pos-procesado:** David Herrera Poyatos
5. **Estudios realizados:** David Herrera Poyatos
6. **Resultados (Experimento 1-8):** Adán Cruz Vargas
7. **Resultados (Experimentos 9-10):** Andrés Herrera Espino
8. **Resultados (Experimentos 11-16:** Zewei Wang Chen
9. **Resultados (Experimento 17-18):** David Herrera Poyatos
10. **Resultados con los transformers y comparativa con las CNNs** Zewei Wang Chen
11. **Conclusiones:** Adán Cruz Vargas y David Herrera Poyatos (ensembles)
12. **Conclusiones de los resultados Transformers vs CNNs:** Zewei Wang Chen

Referencias

[1] Rishit Dagli. *Image classification with Swin Transformers*. 2021. URL: https://keras.io/examples/vision/swin_transformers/.

[2] Jia Deng y col. "ImageNet: A large-scale hierarchical image database". En: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, págs. 248-255. DOI: 10.1109/CVPR.2009.5206848.

[3] Alexey Dosovitskiy, Lucas Beye y Alexander Kolesnikov. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". En: *CoRR* (2020). URL: https://arxiv.org/abs/2010.11929.

[4] Xavier Glorot y Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Yee Whye Teh y Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May de 2010, págs. 249-256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[5] K. He y col. "Deep Residual Learning for Image Recognition". En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[6] Gao Huang y col. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[7] Diederik P. Kingma y Jimmy Ba. "Adam: A Method for Stochastic Optimization". En: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. por Yoshua Bengio y Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[8] Zewen Li y col. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. 2020. arXiv: 2004.02806 [cs.CV].

[9] Ze Liu y col. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". En: *CoRR* (2021). URL: https://arxiv.org/abs/2103.14030.

[10] Ilya Loshchilov y Frank Hutter. "SGDR: Stochastic Gradient Descent with Restarts". En: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983.

[11] Zoe Papakipos y Joanna Bitton. *AugLy: Data Augmentations for Robustness*. 2022. arXiv: 2201.06494 [cs.AI].

[12] P Radhakrishnan. "Why Transformers are Slowly Replacing CNNs in Computer Vision". En: (2021). URL: https://becominghuman.ai/transformers-in-vision-e2e87b739feb.

[13] Khalid Salama. *Image Clasification with visión Transformers*. 2021. URL: https://keras.io/examples/vision/image_classification_with_vision_transformer/.

[14] Mingxing Tan y Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].