Clasificación de nubes – Inteligencia Artificial

Imanol Muñiz Ramirez A01701713

Índice

Contents

Clasificación de nubes – Inteligencia Artificial 1
Índice 1
Objetivo 2
Sobre las nubes 2
Clear Sky (Cielo despejado) 2
Cirriform Clouds (Nubes cirriformes) 3
Cumulonimbus Clouds (Nubes cumulonimbos) 3
Cumulus Clouds (Nubes cúmulos) 4
High Cumuliform Clouds (Nubes cumuliformes altas) 5
Stratiform Clouds (Nubes estratiformes) 5
Stratocumulus Clouds (Nubes estratocúmulos) 6
Arquitectura del modelo 7
Estructura del proyecto 8
Dataset 8
Modelo primera iteración 9
Implementación 9
Resultados entrenamiento y validación 10
Resultados prueba 10
Modelo segunda iteración 12
Mejoras segunda iteración 12
Resultados de entrenamiento y validación segunda iteración 13
Resultados de prueba segunda iteración 13
Modelo tercera iteración 14

Mejoras tercera iteración 14
Resultados de entrenamiento y validación tercera iteración 15
Resultados de prueba tercera iteración 15
Configuraciones y modelos generados 16
Referencias 16
Explicación arquitectura procesamiento del lenguaje natural. 16

Objetivo

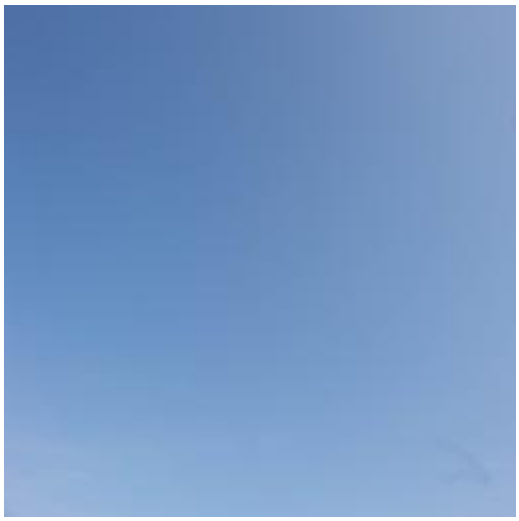
Generar un modelo que utilice aprendizaje profundo capaz de categorizar imágenes de nubes en 7 tipos distintos: cirriform clouds, clear sky, cumulonimbus clouds, cumulus clouds, high cumuliform clouds, stratiform clouds y stratocumulus clouds.

Sobre las nubes

Identificar los distintos tipos de nubes es importante porque cada tipo representa condiciones atmosféricas específicas y puede indicar cambios en el tiempo, estabilidad del aire, y posibles fenómenos meteorológicos. Los meteorólogos y climatólogos utilizan esta información para predecir el clima, estudiar la dinámica atmosférica, y analizar patrones de precipitación o formación de tormentas.

Clear Sky (Cielo despejado)

- Qué representa: Ausencia significativa de nubes. Indica alta presión atmosférica y estabilidad del aire.
- Importancia: Suele asociarse con buen tiempo, pero también puede implicar pérdida de calor nocturno (por falta de cobertura nubosa) y formación de heladas en zonas frías.

Cirriform Clouds (Nubes cirriformes)

- Ejemplos: Cirrus, cirrostratus, cirrocumulus.
- Altitud: Altas (más de 6,000 m).
- Qué representan: Compuestas principalmente por cristales de hielo; se forman en aire muy frío y seco.
- Importancia: Indican cambios en la atmósfera superior. Su presencia puede anticipar la llegada de un frente cálido o tormenta en 24–48 horas.


Cumulonimbus Clouds (Nubes cumulonimbos)

- Altitud: Se desarrollan verticalmente desde niveles bajos hasta la tropopausa (capa de transición entre la troposfera y la estratosfera) (pueden alcanzar 12–18 km).

- Qué representan: Nubes de desarrollo vertical intenso, con corrientes ascendentes fuertes, capaces de producir tormentas eléctricas, granizo, lluvia intensa y tornados.
- Importancia: Son las principales responsables del tiempo severo. Identificarlas ayuda a anticipar tormentas y fenómenos peligrosos.


Cumulus Clouds (Nubes cúmulos)

- Altitud: Nubes bajas o de nivel medio (1–2 km).
- Qué representan: Nubes algodonosas formadas por convección (aire caliente que asciende).
- Importancia: Indican inestabilidad moderada. Si crecen mucho verticalmente, pueden evolucionar a cumulonimbus (tormentas). Son típicas de buen tiempo diurno con calor.


High Cumuliform Clouds (Nubes cumuliformes altas)

- Ejemplos: Altocumulus, cirrocumulus.
- Altitud: Nivel medio-alto (3,000–6,000 m).
- Qué representan: Indican inestabilidad en niveles medios de la atmósfera.
- Importancia: Pueden señalar inicios de convección o cambios en la humedad de capas medias, preludio de tormentas o frentes.


Stratiform Clouds (Nubes estratiformes)

- Ejemplos: Stratus, altostratus, nimbostratus.
- Altitud: Bajas a medias (hasta 6,000 m).
- Qué representan: Nubes extendidas horizontalmente, asociadas con estabilidad atmosférica y precipitación ligera o continua.
- Importancia: Indican tiempo nublado o lluvioso prolongado. Suelen formarse por levantamiento suave de aire húmedo.


Stratocumulus Clouds (Nubes estratocúmulos)

- Altitud: Bajas (1–2 km).
- Qué representan: Nubes grandes y onduladas que cubren el cielo parcial o totalmente.
- Importancia: Indican inestabilidad débil, con posibilidad de lloviznas ligeras. Son comunes tras el paso de frentes fríos o durante el amanecer.


Arquitectura del modelo

El modelo utilizado en este trabajo es un “vision trasformer” (ViT). Es una arquitectura similar a la empleada en un transformer de procesamiento de lenguaje natural. Este consta de los pasos descritos en la imagen de la derecha.

1. Generación de patches: parte la imagen en porciones y cada una la convierte en un vector.
2. Embedding lineal: hace una reducción de la dimensionalidad de los vectores.
3. Codificación posicional: Crea vectores de la misma dimensión con la información de la posición original de los patches y lo añade.
4. Codificador transformer: Repite 12 veces un bloque de codificación compuesto por un un sub-bloque atencional, un bloque residual, una red neuronal y finalmente otro bloque residual.
    a. El sub-bloque atencional se encarga de calcular las relaciones entre los diferentes vectores o conjuntos de vectores según la composición original de la imagen. Tal cual como sucede en un modelo de procesamiento del lenguaje natural en el que se calculan las relaciones entre las palabras y frases. Al final devuelve vectores con la información más importante de las relaciones de cada vector de entrada. Este sub-bloque se hace múltiples veces para considerar en diferentes niveles las relaciones entre vectores.
    b. Los sub-bloques residuales toman una entrada y salida del sub-bloque anterior, la suman y normaliza. Esto sirve para evitar la degradación del modelo cuando se va haciendo muy profundo.
    c. El sub-bloque neuronal toma toda la información atencional y la consolida en un conjunto único de vectores.
    Para una explicación más detallada de esta parte mira este video *https://www.youtube.com/watch?v=Wp8NocXW_C4*

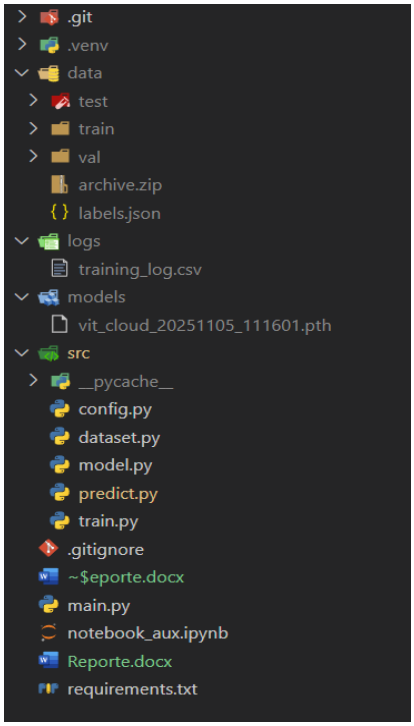
    Al final de este paso tenemos un arreglo de vectores que contiene la información posicional y relacional de cada uno de ellos, lo que se conoce como codificaciones atencionales.
5. Pooling: Se toman las codificaciones atencionales y se compactan a un solo vector promediando por columnas.
6. Cabecera de salida: Utiliza una red neuronal que aprende a clasificar este vector compacto en categorías. Útil para hacer *fine-tunning*

Para una explicación más detallada de este proceso mira este video https://www.youtube.com/watch?v=A-6DF9mkDuQ.

La versión utilizada en específico para este proyecto es vit-base-patch16-224 (86.6 millones de parámetros). Un modelo entrenado con ImageNet-21k (14 millones de imágenes, 21,843 clases). El modelo se presentó por primera vez en el artículo https://arxiv.org/pdf/2010.11929 de Dosovitskiy

Estructura del proyecto

El archivo main.py es el encargado de entrenar el modelo utilizando clases y funciones de los archivos de la carpeta src a excepción de predict.py que es el que hace la inferencia de una imagen que indiquemos. Genera el modelo dentro de la carpeta models y un log de la configuración en el archivo dentro de la carpeta de logs. Las imágenes se encuentran en la carpeta de data divididas en train, val y test. En cada una de estas deben estar las imágenes divididas en 7 carpetas con los nombres de los diferentes tipos de nubes.



En el archivo notebook.aux.ipynb están alguno scripts pequeños para conocer alguna información o ejecutar una tarea como generar el archivo labels.json que contiene los nombres de las categorías de nubes.

Finalmente, el requirements.txt contiene las librerías y las versiones que se utilizaron para este proyecto.

Dataset

El dataset para entrenamiento y validación fue extraído de kaggle (https://www.kaggle.com/datasets/zeesolver/cloiud-dataset).

Contiene 960 imágenes divididas en entrenamiento y validación. Cada una de estas con los 7 diferentes tipos de nubes. 486 en entrenamiento y 474 en validación.

La distribución de imágenes está de la siguiente manera

Categoría	Entrenamiento	Validación
clear sky	60	64
cirriform clouds	66	77
cumulonimbus clouds	12	11
cumulus clouds	91	120

high cumuliform clouds	133	103
stratiform clouds	49	40
stratocumulus clouds	61	71

Modelo primera iteración

Implementación

Para este primer modelo la implementación del vision transformer y los datos es bastante simple. Podemos ver todo el procedimiento desde el código del archivo main.py

```python
def main():
    cfg = CONFIG
    device = cfg["device"]

    # 1️⃣ Cargar datos
    train_loader, val_loader, class_names = get_data_loaders(
        cfg["data_dir"], cfg["batch_size"], cfg["img_size"]
    )
    print(f"📂 Clases detectadas: {class_names}")

    # 2️⃣ Modelo ViT
    model = create_vit_model(cfg["model_name"], len(class_names))
    model.to(device)

    # 3️⃣ Pérdida y optimizador
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=cfg["learning_rate"])

    # 4️⃣ Entrenamiento principal
    for epoch in range(cfg["epochs"]):
        print(f"\n🚀 Época {epoch + 1}/{cfg['epochs']}")
        train_loss, train_acc = train_one_epoch(model, train_loader, optimizer, criterion, device)
        val_loss, val_acc, val_f1 = validate(model, val_loader, criterion, device)
```

Primero cargamos la configuración del archivo config.py. Este contiene información como el número de épocas, el learning rate, la versión del modelo etc. Posteriormente cargamos los datos con las funciones que definimos en el archivo dataset.py, adicionalmente en esta misma función hacemos un aumento de los datos de entrenamiento pues nuestro conjunto original no es muy grande (Rotaciones, flips, colores, traslaciones, perspectivas). Procedemos a inicializar el modelo con la función que definimos en el archivo model.py, definimos la función de perdida (cross entropy) y el optimizador del learning rate. Finalmente hacemos un ciclo de la cantidad de épocas que definimos en la configuración y ejecutamos en cada iteración las funciones de entrenamiento y validación que definimos en el archivo train.py. Estas nos devuelven las métricas de desempeño del modelo y va guardando los pesos del entrenamiento.

Como se explicó en la parte de arquitectura del modelo, la última parte de este modelo es una red neuronal que categoriza por defecto en 1000 clases. Sin embargo, podemos ajustarlo a la cantidad que necesitemos. En este caso 7.

Finalmente guardamos las métricas en un log del archivo training_log.csv y el modelo entrenado en la carpeta de models.

Para probar la capacidad de inferencia del modelo utilizamos el archivo predict.py que transforma una imagen al formato requerido, carga el modelo y hace la inferencia.

Resultados entrenamiento y validación

Para esta primera versión del modelo obtuvimos las siguientes métricas

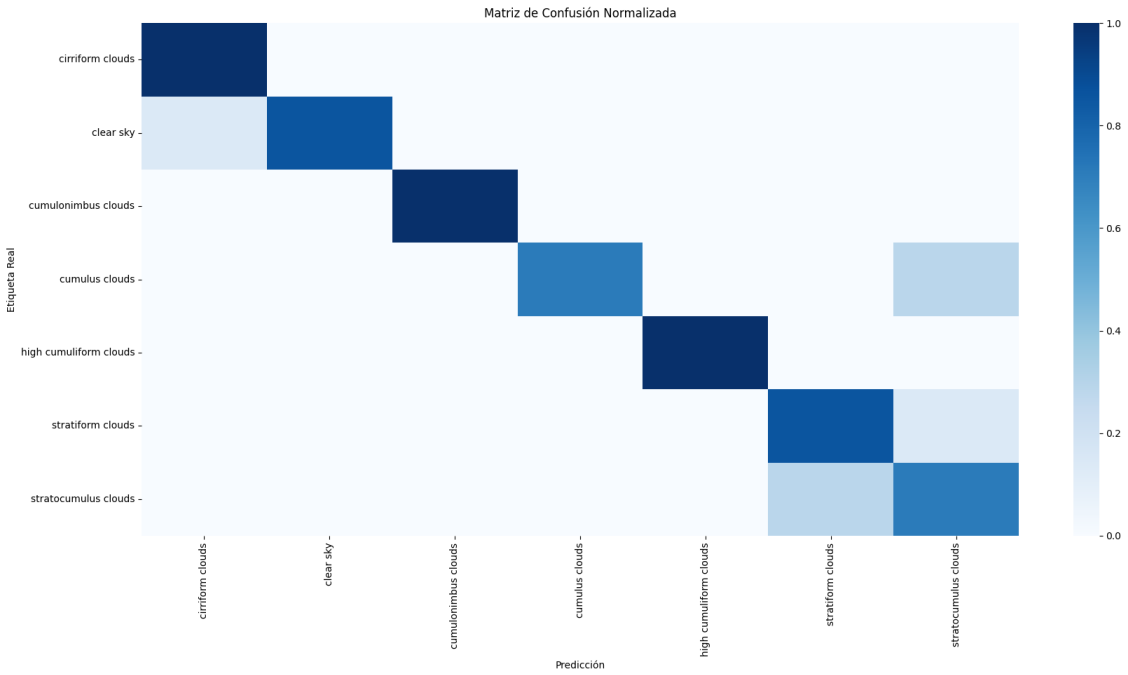
	Accuracy	F1
Entrenamiento	0.98	
Validación	0.94	0.94

De estos valores podemos concluir que hasta este punto el modelo tiene un buen desempeño acertando casi en su totalidad. No denota problemas de undefitting ni overfitting y al hacer inferencias únicas ha mostrado con más de 90% de seguridad la respuesta correcta.

Resultados prueba

Para evaluar el modelo en un ambiente de prueba, se compiló un diverso conjunto de nuevas imágenes del cielo por fotografía o descargadas de la web con las que se puso a prueba el modelo. Este conjunto de datos contiene 7 imágenes por categoría.

Los resultados fueron los siguientes:



RESULTADOS GENERALES:

	precision	recall	f1-score	support
cirriform clouds	0.875	1.000	0.933	7
clear sky	1.000	0.857	0.923	7
cumulonimbus clouds	1.000	1.000	1.000	7
cumulus clouds	1.000	0.714	0.833	7
high cumuliform clouds	1.000	1.000	1.000	7
stratiform clouds	0.750	0.857	0.800	7
stratocumulus clouds	0.625	0.714	0.667	7
accuracy			0.878	49
macro avg	0.893	0.878	0.879	49
weighted avg	0.893	0.878	0.879	49

- F1-macro: 0.8795
- F1-micro: 0.8776
- F1-weighted: 0.8795

Como se observa en las imágenes de arriba, el desempeño del modelo con imágenes que nunca ha visto decrementa en comparación con la etapa de validación en alrededor de un 7% y un 11% respecto al entrenamiento lo que nos indica un ligero problema de overfitting. A esta situación le atribuyo estas posibles causas

1. Las imágenes de entrenamiento y validación son significativamente diferentes a las de prueba.
2. Le falta trabajo al conjunto de imágenes que extrajimos de kaggle o es posible que se haya usado información adicional a su apariencia para categorizarlas como su altura, posición geográfica, temperatura etc. que no está en la imagen. Al hacer una revisión manual es común encontrar imágenes que al juzgar su forma debería pertenecer a otra categoría.
3. No he etiquetado perfectamente cada una de las imágenes que recolecté para el conjunto de imágenes de prueba. Esta causa me hace especial sentido pues me fue

complicado encontrar imágenes de la categoría “stratocumulus clouds” y diferenciarlas tanto con “stratiform clouds” como con “high cumuliform clouds” pues cuando yo veo el conjunto de estas imágenes algunas parecen una combinación de ambas, pero otras podrían encajar en alguna de las otras dos categorías. Esta situación se ve reflejada en la matriz de confusión pues es justamente esta categoría la que tiene muchos problemas.

4. El conjunto de entrenamiento no tiene suficientes ejemplos para generalizar algunas clases de mejor manera.

Por otra parte, podemos concluir que el modelo se adecua bien a la tarea pues logra un buen desempeño de entrenamiento validación e incluso pruebas en este primer intento por lo que no será necesario cambiar a otro modelo, sino enfocarnos en mejorar este.

Modelo segunda iteración

Mejoras segunda iteración

Para intentar mejorar el desempeño del modelo vamos a implementar un plan orientado en atender las causas descritas en la sección anterior. Cabe destacar que también se mejorarán aspectos ajenos al modelo con el objetivo de lograr el objetivo de una mejor forma.

Mejora	Razón / Objetivo
Se eliminará la categoría “stratocumulus clouds”.	Esta categoría tiene muchos problemas de definición en cuanto a su visualización. En la web podemos encontrar muchas instituciones climatológicas con diferentes categorizaciones de nubes por lo que opté por quedarme únicamente con aquellas que sí sean visualmente definibles por mí para no depender de un experto.
Se revisarán manualmente las categorías y se eliminarán las imágenes que ocasionen ruido.	Que cada categoría esté bien definida.
El conjunto de entrenamiento tendrá el 80% de las imágenes de kaggle con el objetivo de ver si con más ejemplos puede generalizar mejor. El de validación tendrá el 20% restante. El conjunto de prueba se queda igual	Es posible que el conjunto de entrenamiento y validación contenga ejemplos muy sencillos o similares entre sí, por lo que opté por no revolver el de prueba con estos

Ahora la distribución de los datos quedó de la siguiente forma

Categoría	Entrenamiento	Validación	Prueba
clear sky	115	28	7
cirriform clouds	97	24	7
cumulonimbus clouds	20	5	7
cumulus clouds	152	37	7
high cumuliform clouds	178	44	7
stratiform clouds	71	17	7
stratocumulus clouds	0	0	0

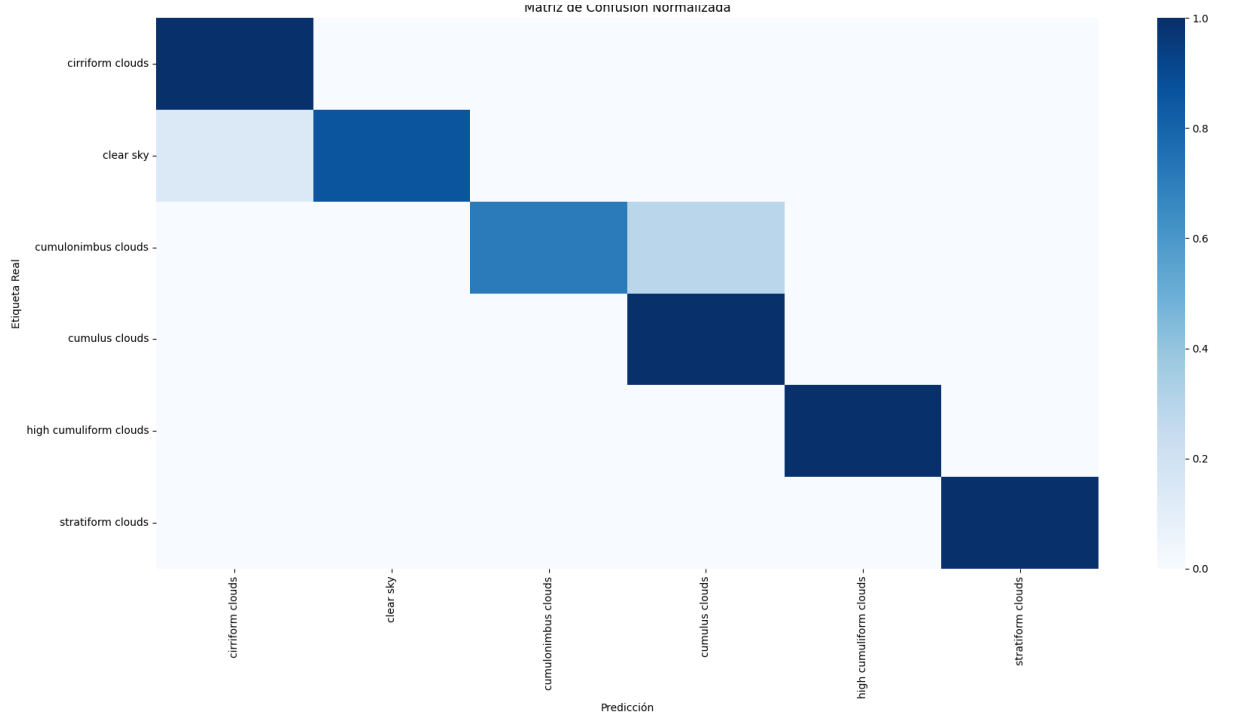
Posterior a estas mejoras volveremos a evaluar el desempeño del modelo y haremos nuevas mejoras.

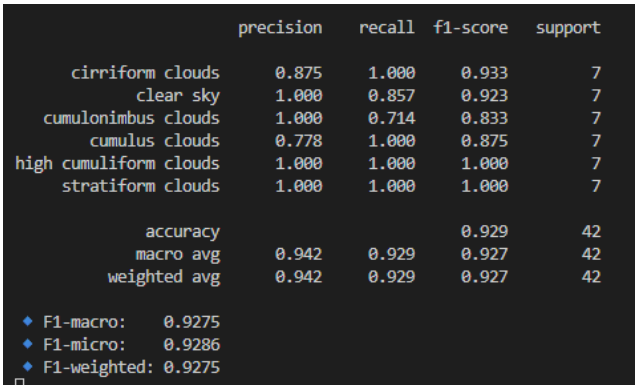
Resultados de entrenamiento y validación segunda iteración

El incremento tanto en accuracy como F1-score aumentó en alrededor de 2% en comparación al entrenamiento y validación de la primer iteración.

	Accuracy	F1
Entrenamiento	0.994	
Validación	0.961	0.96

Resultados de prueba segunda iteración





El F1-score aumentó alrededor de un 5%. Hay dos imágenes que deberían ser categorizadas como cumulonimbus pero las manda a cumulus. Esto podría ser un error causado por un desbalance en los datos pues "cumulonimbus clouds" es la categoría con menos imágenes del conjunto de imágenes y es común que estas clases con pocos ejemplos sean absorbidas por otras categorías.

Modelo tercera iteración

Mejoras tercera iteración

Dado que es posible que afrontemos un problema de desbalance de datos, lo lógico es aplicar una técnica de regularización de balanceo. Para esto vamos a calcular el peso que debe tener cada imagen dependiendo de la categoría a la que pertenezca, de esta forma las clases con pocos ejemplos tienen un mayor peso y obliga al modelo prestarles más atención. Adicionalmente cambiaremos la función de perdida por "Label smooth cross entropy". Esta función suaviza los labels. Cuando se está entrenando el modelo, en lugar de marcar la respuesta correcta con 1 y las demás con 0, toma un porcentaje del 1 y lo divide entre el resto de las clases. Esto hace que nuestro modelo se vuelva menos confiado al estar prediciendo. En casos desbalanceados como este, generalmente la precisión va aumentando conforme va diciendo que es la clase con mayor número de ejemplos (por eso las clases grandes absorben a las pequeñas), al suavizar los labels, en las imágenes que este dudando, ese porcentaje de incertidumbre que se reparte puede ayudar a cambiar la decisión de siempre estar tomando la clase más grande.

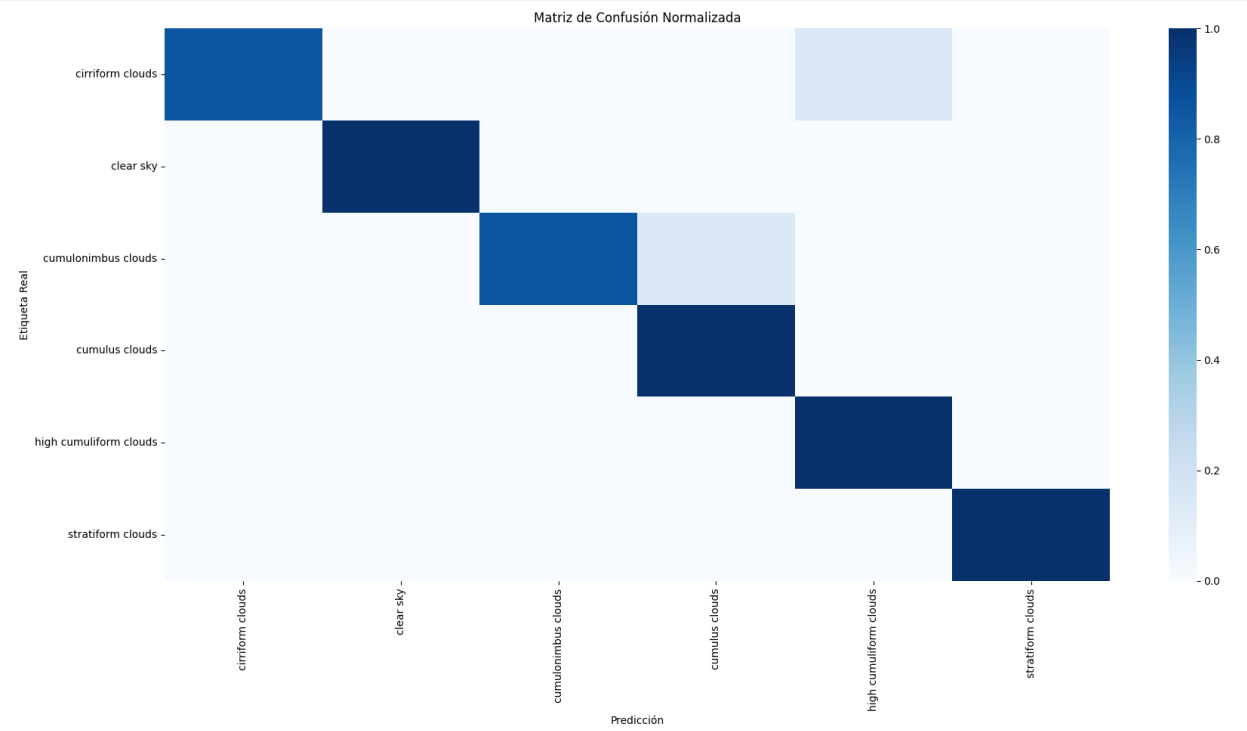
Mejora	Razón / Objetivo
Técnica de regularización: Balanceo de pesos de las imágenes dependiendo de su categoría	Las clases pequeñas aparentemente están siendo absorbidas
Cambiar la función de perdida de cross entropy por label smooth cross entropy	Puede ayudar con clases desbalanceadas,

Resultados de entrenamiento y validación tercera iteración

Podemos ver cómo estos cambios mejoraron en 1.9% el F1-score en validación

	Accuracy	F1
Entrenamiento	0.988	
Validación	0.98	0.979

Resultados de prueba tercera iteración



	precision	recall	f1-score	support
cirriform clouds	1.000	0.857	0.923	7
clear sky	1.000	1.000	1.000	7
cumulonimbus clouds	1.000	0.857	0.923	7
cumulus clouds	0.875	1.000	0.933	7
high cumuliform clouds	0.875	1.000	0.933	7
stratiform clouds	1.000	1.000	1.000	7
accuracy			0.952	42
macro avg	0.958	0.952	0.952	42
weighted avg	0.958	0.952	0.952	42

- F1-macro: 0.9521
- F1-micro: 0.9524
- F1-weighted: 0.9521

En pruebas el problema cambió. Aunque se mitigó que los errores se produzcan en una sola clase, ahora la confusión también proviene de otras categorías y las métricas de desempeño quedaron prácticamente igual (hubo una mejora en las métricas, pero se debió a la

corrección de una imagen mal categorizada). Mi intuición es que algunas de las imágenes contienen más de un tipo de nube y es complicado que encajen en una sola categoría.

Configuraciones y modelos generados

El archivo logs/training_log.csv contiene las configuraciones del modelo y su desempeño.

```
timestamp,model_name,epochs,batch_size,lr,img_size,train_loss,train_acc,val_loss,val_acc,val_f1,smoothing
2025-11-05 11:16:02,google/vit-base-patch16-224,25,16,0.0001,224,0.0492,0.9873,0.2227,0.9424,0.9415,0.0
2025-11-17 16:58:28,google/vit-base-patch16-224,25,16,0.0001,224,0.0267,0.9937,0.1506,0.9613,0.9596,0.0
2025-11-17 19:03:44,google/vit-base-patch16-224,25,16,0.0001,224,0.5100,0.9889,0.5696,0.9806,0.9792,0.1
2025-11-17 19:55:55,google/vit-base-patch16-224,25,16,0.0001,224,0.8737,0.9860,1.0034,0.9740,0.9730,0.2
2025-11-17 20:41:35,google/vit-base-patch16-224,20,16,0.0001,224,0.0864,0.9731,0.1929,0.9355,0.9415,0.0000
```

Referencias

Zeesolver. (s.f.). *CLOIUD – Cloud Dataset* [Conjunto de datos]. Kaggle. https://www.kaggle.com/datasets/zeesolver/cloiud-dataset

Yacine Brinis. (2023). *Cloud types explained with real images* [Video]. YouTube. https://www.youtube.com/watch?v=A-6DF9mkDuQ

MetOffice UK. (2020). *Types of clouds and their characteristics* [Video]. YouTube. https://www.youtube.com/watch?v=Wp8NocXW_C4

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An image is worth 16×16 words: Transformers for image recognition at scale*. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2010.11929

Explicación arquitectura procesamiento del lenguaje natural.

Con el objetivo de generar evidencia sobre el conocimiento adquirido durante el curso sobre las arquitecturas de procesamiento del lenguaje natural, me fue requerido explicar un breve código de una arquitectura para este propósito. El código a continuación fue extraído de mi evidencia *"Utilización, procesamiento y visualización de grandes volúmenes de datos"* en la que hago un modelo de análisis de temas de textos con el framework ml de PySpark.

La primera parte consiste en preparar los textos. La parte importante es la función clean_data donde estandarizamos los textos quitando las mayúsculas, cambiando las tabulaciones, saltos de linea y espacios por ' ', finalmente limpiamos filtrando aquellos

registros que estén vacíos o muy cortos (menos de 20 caracteres). Esto ayuda a procesar los textos más eficientemente cuando no es importante comprender la puntuación de las oraciones.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import RegexTokenizer, StopWordsRemover, CountVectorizer, IDF, StringIndexer, NGram, VectorAssembler
from pyspark.sql.functions import length, col, lower, trim, regexp_replace

df_valid = spark.read.csv(
    r"D:\Tec\Septimo\Cloud\Global Superstore Analysis\WikiCAT_dataset\valid_esv5.csv",
    header=True,
    inferSchema=True,
    encoding="UTF-8"
)

df_train = df_wikicat.withColumnRenamed("sentence", "texto") \
                     .withColumnRenamed("label", "tema")

df_test = df_valid.withColumnRenamed("sentence", "texto") \
                  .withColumnRenamed("label", "tema")

def clean_data(df):
    df = df.withColumn("texto", lower(col("texto")))
    df = df.withColumn("texto", trim(col("texto")))
    df = df.withColumn("texto", regexp_replace(col("texto"), r'\s+', ' '))  # Normalizar espacios

    df = df.filter(col("texto").isNotNull())
    df = df.filter(col("texto") != "")
    df = df.filter(length(col("texto")) > 20)
    df = df.filter(col("tema").isNotNull())

    return df

train_data = clean_data(df_train)
test_data = clean_data(df_test)
```

Después creamos el pipeline por el que pasará nuestros dataframes para tokenizar y convertir a vectores las palabras de las oraciones.

1. Primero utilizamos RegexTokenizer para definir cuáles serán nuestros tokens. Para este trabajo vamos a utilizar el patrón \W que divide las oraciones en palabras separadas por simbolos de puntuación como espacios, comas, puntos y otros símbolos como preguntas, exclamaciones, guiones etc. Adicionalmente definimos minTokenLength=3 para indicar que no queremos asignar token a palabras de 3 letras o menos como "a", "de", "les", "y" etc. Esto lo hacemos debido a que generalmente estas palabras no aportan significado al tema del texto por lo que así eficientamos y quitamos ruido al proceso.
2. Con el mismo objetivo del punto anterior ahora utilziamos StopWordsRemover que nos elimina aquellas palabras que no aportan significado a la oración pero que pueden contener más de 3 letras, por ejemplo "como", "pero" etc.
3. Posteriormente usamos CountVectorizer para contar las palabrás únicas entre todos los registros de nuestro dataset. Tomamos únicamente las 10,000 más frecuentes y las que aparezcan al menos en 2 oraciones.
4. Luego, utilizamos IDF para calcular para calcular que tan importante son las palabras para predecir el tema del texto, si una palabra está en todos los textos se le asigna un peso pequeño, si tiene pocas apariciones puede ser que sea importante para el

análisis. Únicamente aplicamos este peso a aquellas palabras que tengan al menos 2 apariciones en distintos textos. Esto evita que las palabras que aparezcan en un solo texto tengan demasiado peso en la predicción.

5. Finalmente usamos StringIndexer para convertir las etiquetas en valores númericos.

```
tokenizer = RegexTokenizer(
    inputCol="texto",
    outputCol="palabras",
    pattern="\\W",
    minTokenLength=3
)

remover = StopWordsRemover(
    inputCol="palabras",
    outputCol="palabras_limpias",
    stopWords=StopWordsRemover.loadDefaultStopWords("spanish")
)

cv = CountVectorizer(
    inputCol="palabras_limpias",
    outputCol="rawFeatures",
    vocabSize=10000,
    minDF=2.0
)

idf = IDF(
    inputCol="rawFeatures",
    outputCol="features",
    minDocFreq=2
)

indexer = StringIndexer(
    inputCol="tema",
    outputCol="label",
    handleInvalid="keep"
)

preprocessing_pipeline = Pipeline(stages=[tokenizer, remover, cv, idf, indexer])
```

Con el pipeline de preprocesamiento listo, procedemos a pasar nuestros datos a través de este y transformamos a vectores las oraciones de nuestros textos.

```
preprocessing_model = preprocessing_pipeline.fit(train_data)

train_processed = preprocessing_model.transform(train_data)
test_processed = preprocessing_model.transform(test_data)
```

```
print(train_processed.columns)
print(train_processed.select("features").show(1, truncate=False))
```

```
['id', 'tema', 'texto', 'palabras', 'palabras_limpias', 'rawFeatures', 'features', 'label']
+-------------------------------------------------------------------------------------------------------------
|features
+-------------------------------------------------------------------------------------------------------------
|(10000,[4,5,18,23,36,46,50,56,63,80,86,101,119,120,130,141,188,194,197,203,207,208,236,266,291,314,376,443,452,455,588,590,697,716,821,834,
```

Ya con nuestras secuencias de vectores listas procedemos a configurar un modelo de regresión logística que clasifique las secuencias generadas del conjunto de train y probamos el desempeño con los datos de evaluación. La configuración es la siguiente

- featuresCol: indica las variables independientes, en nuestro caso las secuencias de vectores.

- labelCol: indica la columna con las variables objetivo, en este caso el índice de los temas de los textos.
- maxIter: número de épocas
- regParam: Coeficiente de regularización. Controla la penalización aplicada a los pesos del modelo para evitar overfitting. Valores más altos penalizan más
- elasticNetParam: penalización L1, L2.
- tol: Detiene el entrenamiento cuando las mejoras son muy pequeñas
- standarization: normaliza los valores de los vectores
- family: indica la función de perdida multinomial, cross entropy en este caso.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql.functions import when, col

modelo_clasificacion = LogisticRegression(
    featuresCol="features",
    labelCol="label",
    maxIter=100,
    regParam=0.001,
    elasticNetParam=0.1,
    tol=1e-6,
    standardization=True,
    family="multinomial"
)

modelo_entrenado = modelo_clasificacion.fit(train_processed)
# Obtener predicciones sobre el conjunto de prueba
predictions = modelo_entrenado.transform(test_processed)
```