Clasificación de nubes – Inteligencia Artificial

Imanol Muñiz Ramirez A01701713

Índice

Contents

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).

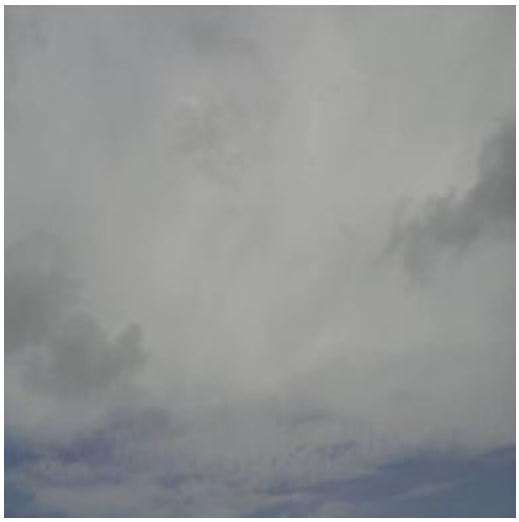
- Importancia: Indican inestabilidad moderada. Si crecen mucho verticalmente, pueden evolucionar a cumulonimbus (tormentas). Son típicas de buen tiempo diurno con calor.


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*

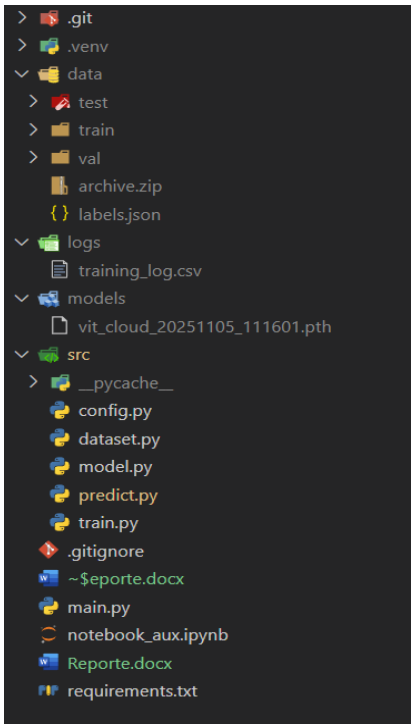
Al final de este paso tenemos un arreglo de vectores que contiene la información posicional y relacional de cada uno de ellos, lo que se conoce como codificaciones atencionales.

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).

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

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, el número de clases, la versión del modelo etc. Posteriormente cargamos los datos con las funciones que definimos en el archivo dataset.py. Procedemos a inicializar el modelo con la función que definimos en el archivo model.py, definimos la función de perdida 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, reales y sintéticas con las que se puso a prueba el modelo. Este conjunto de datos contiene

[Insertar tabla de distribución de imágenes]

[Insertar matriz de confusión]

[Insertar conclusiones]

Referencias

https://www.youtube.com/watch?v=A-6DF9mkDuQ

https://www.youtube.com/watch?v=Wp8NocXW_C4

https://arxiv.org/pdf/2010.11929