Informe sobre el proyecto del clasificador de imágenes climáticas de Roma

Miguel Ángel Massiris Fernández

28 de ago. De 2023

Prof. Emmanuel Iarussi

Indice

Introducción y Objetivo

Para mostrar todos los conocimientos vistos durante el curso “Deep Learning” en la Universidad Nacional del Sur lleve a cabo un clasificador de imágenes del clima, la fuente de datos a utilizar se extrajo desde la página Kaggle. La implementación del clasificador se llevó a cabo en Google Colaboratory, aprovechando la disponibilidad de una unidad de procesamiento gráfico (GPU) gratuita proporcionada por Google. En el ámbito del aprendizaje profundo, esta plataforma se torna particularmente beneficiosa, al acelerar tanto los tiempos de procesamiento como el entrenamiento de redes neuronales.

Exploración de los datos:

Conforme se mencionó previamente, el conjunto de datos fue extraído de Kaggle y se encuentra disponible en el siguiente enlace: https://www.kaggle.com/datasets/rogeriovaz/rome-weather-classification. El conjunto de datos está compuesto por cinco directorios, cada uno contiene 50 imágenes.

Estos directorios representan distintos tipos de condiciones climáticas en la ciudad de Roma: Soleado, Lluvioso, Nublado, Brumoso y Nevado. Se obtiene entonces unos datos balanceados, pero con pocos ejemplos, sin embargo, se busca utilizar modelos pre-entrenados para obtener mejores resultados.

Como los datos ya están clasificados es posible organizar todas las imágenes según su etiqueta en un solo dataframe, compuesto 2 columnas:

- File_Path: En esta primera columna se registran las rutas de acceso a cada una de las imágenes.
- Labels: Tiene la etiqueta correspondiente a esa imagen.

Preparación de los datos

En esta etapa se busca aplicar data augmentation para variar las imágenes para cada época del entrenamiento, inicialmente solo aplicaba 3 layers aprovechando la librería keras.Sequential que aplicaba:

- RandomFlip: Esta capa invierte aleatoriamente las imágenes.
- RandomRotation: Aplica una rotación aleatoria a las imágenes en función de un factor, dando así una rotación de tal como: factor*180. Para entenderlo mejor dejo un ejemplo: Si el factor es 0.2 genera en una rotación de salida por una cantidad aleatoria en el rango [-20% * 2pi, 20% * 2pi].
- RandomContrast: Ajusta aleatoriamente el contraste en función del factor dado, entonces para cualquier píxel x en el canal, la salida se calculará como (x - media) * factor + media, donde la media es el valor promedio del canal.

Pero durante el entrenamiento de los modelos pre-entrenados encontré que cada modelo tiene su propia implementación de preprocess_input para asegurarse de que los datos de entrada estén en el formato correcto y se ajusten a las expectativas del modelo. Estas funciones pueden realizar diferentes transformaciones en las imágenes, como normalización, redimensionamiento y ajuste de escala, dependiendo de las necesidades del modelo específico. Por lo tanto, es recomendable utilizar la función preprocess_input correspondiente al modelo que estés utilizando para obtener los mejores resultados de clasificación.

Aprovechando se crea una función llamada generación_de_datos, la cual pide como datos de entrada: el respectivo process_input del modelo elegido, el dataframe de entrenamiento y el dataframe de testeo. Dicha función se encarga de generar datos de entrenamiento, validación y prueba para cada modelo.

Primero, se definen dos objetos train_datagen y test_datagen utilizando la clase ImageDataGenerator de Keras. Estos objetos se utilizan para realizar transformaciones en las imágenes durante el entrenamiento y la prueba del modelo. La función “pre” se pasa como argumento preprocessing_function para aplicar una función de preprocesamiento personalizada de cada modelo a las imágenes. Dentro del objeto train_datagen además se splitea un 20% del dataset para validación.

Luego, se crea un generador de datos para el entrenamiento utilizando el método flow_from_dataframe del objeto train_datagen. Se especifica el DataFrame train que contiene la información de las imágenes y las etiquetas. Se proporciona el nombre de la columna que contiene las rutas de los archivos de imagen (x_col) y el nombre de la columna que contiene las etiquetas (y_col). También se especifica el tamaño de las imágenes después de redimensionar (target_size), el tipo de tarea de clasificación categórica (class_mode), el tamaño del lote (batch_size), entre otros parámetros. Además, se aplican transformaciones de aumento de datos como rotación aleatoria, deformación aleatoria y volteo horizontal.

Después se crea el generador de datos para la validación, nuevamente se utiliza el método Flow_from_dataframe y se le ingresa el mismo dataframe train pero especificando que se use el subconjunto de datos de validación (subset='validation'). Cabe aclarar que este generador no aplica data augmentation a los datos.

Por último, se utiliza el método Flow_from_dataframe pero esta vez se utiliza el dataframe test. Se especifican los mismos parámetros que el generador de datos de entrenamiento, pero no se aplican transformaciones de aumento de datos.

Cerrando la función generación_de_datos se retornan los generadores de datos: train_gen, valid_gen y test_gen.

Dentro de esta función también se manejan hiperparametros como el batch_size=32 e image_size=(100,100). Se escogieron estos valores ya que son valores habituales y validos para los modelos que se utilizarán.

Entrenamiento del modelo

Dado que todos los modelos de aprendizaje profundo siguen un patrón similar al entrenarlos se decide crear una función llamada crear_modelo que englobe los 5 pasos más comunes.

1. La primera parte toma como parámetro el nombre del modelo que se desea utilizar (por ejemplo, Resnet50), también toma el tamaño de entrada de las imágenes y carga los pesos en ImageNet.
2. Luego, establezco que no se entrenarán las capaz del modelo preentrenado por medio del atributo trainable. Para hacerlo se lo establece en False para "congelar" sus capas, evitando entonces que los pesos sean actualizados sobre todo porque no tengo una gran cantidad de datos.
3. A continuación, se crea una sección de tres capas personalizadas para conectarlas con el modelo preentrenado. Las dos primeras ayudan al modelo a reducir la dimensionalidad para mapear el vector de salida a uno más pequeño, mientras la última capa tiene una neurona por cada clase con la función sofrmax para representar una distribución de probabilidad entre las clases.
4. Se compila el modelo indicando la función de perdida, en este caso: categorical_crossentropy. Esta función es comúnmente utilizada en problemas de clasificación multiclase. Luego se especifica el optimizador Adam y las métricas (accuracy).
5. Después, se prepara una función que evita el sobreajuste o overfiting del modelo, dicha función revisa si la métrica de validación (val_loss) deja de mejorar. Apenas nota esa falla en performance para el entrenamiento del modelo.
6. Por último, la función retorna el modelo personalizado y la lista de detención temprana (Early Stopping).

Graficar y evaluación del modelo

En esta función se presentan dos objetivos importantes: Graficar la performance del modelo durante la fase de entrenamiento y evaluar el modelo aplicando el conjunto de prueba.

Graficar: Esta función genera dos gráficos, los cuales van a ser explicados a continuación:

- La primera gráfica muestra la precisión (accuracy) del modelo en el conjunto de entrenamiento y validación a lo largo de las épocas.
- La segunda gráfica muestra la pérdida (loss) del modelo en el conjunto de entrenamiento y validación a lo largo de las épocas. Esta visualización ayudar a entender cómo está aprendiendo y mejorando el modelo con cada época.

Evaluación del modelo: Se presenta la evaluación del modelo por dos vías.

- Predicción: La función utiliza el modelo entrenado para hacer predicciones en el conjunto de prueba. Selecciona la clase con la probabilidad más alta para cada predicción (utilizando np.argmax) y luego convierte los índices de clases predichos en las etiquetas de clase reales.

- Informe de clasificación: Finalmente, la función genera un informe de clasificación y una matriz de confusión utilizando las verdaderas etiquetas de clase y las etiquetas de clase predichas. Este informe incluye métricas como precisión, recall, f1-score, que pueden ayudarte a entender el rendimiento del modelo.
- Testeo del modelo: Se evalúa el modelo con los datos de testeo, se imprime tanto la pérdida (los) como la precisión (accuracy) del modelo.

Evaluación de modelos multiclase

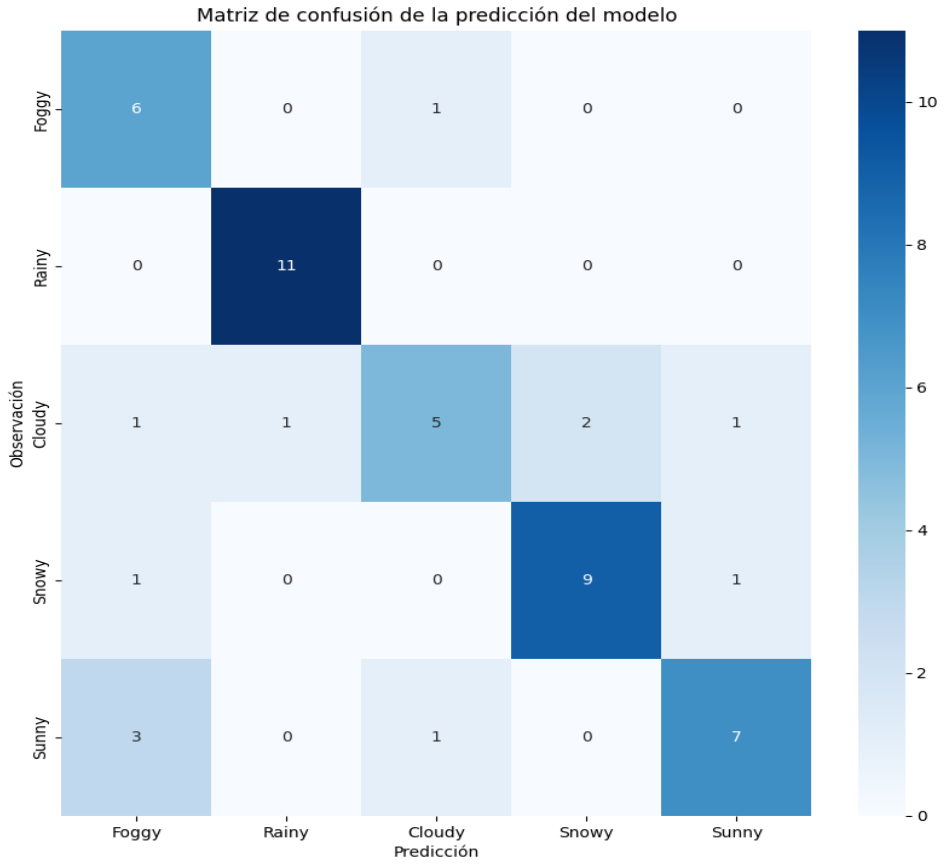
En esta sección, se detallan los resultados de la implementación y evaluación de cinco modelos ampliamente reconocidos en la tarea de clasificación multiclase. Cada modelo fue evaluado en un conjunto de datos que contiene cinco clases distintas. Los modelos seleccionados para esta evaluación son los siguientes: ResNet50, MobileNet, VGG19, ResNet101 y DenseNet201.

1. Modelo ResNet-50

ResNet50 es una red neuronal convolucional profunda que se caracteriza por su arquitectura de "skip connections" o conexiones de salto. Tras entrenar y evaluar ResNet50 en nuestro conjunto de datos, se obtuvo:

Test Loss: 0.68

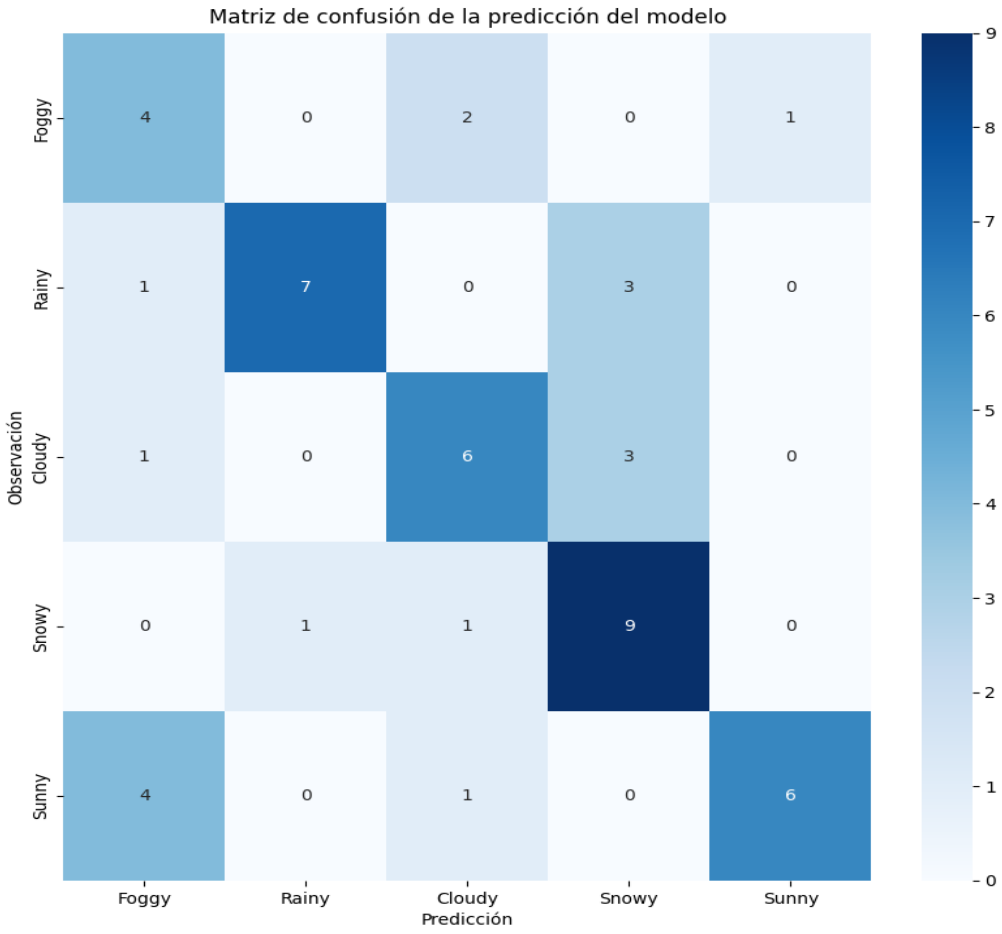
Test Accuracy: 76.00%



2. Modelo VGG19

Es conocida por su arquitectura profunda y uniforme, con capas convolucionales y de agrupación. Aunque es más pesada en términos de parámetros en comparación con otros modelos modernos, sigue siendo un punto de referencia en muchas tareas de visión por computadora. Luego de entrenar este modelo se obtienen los siguientes resultados:

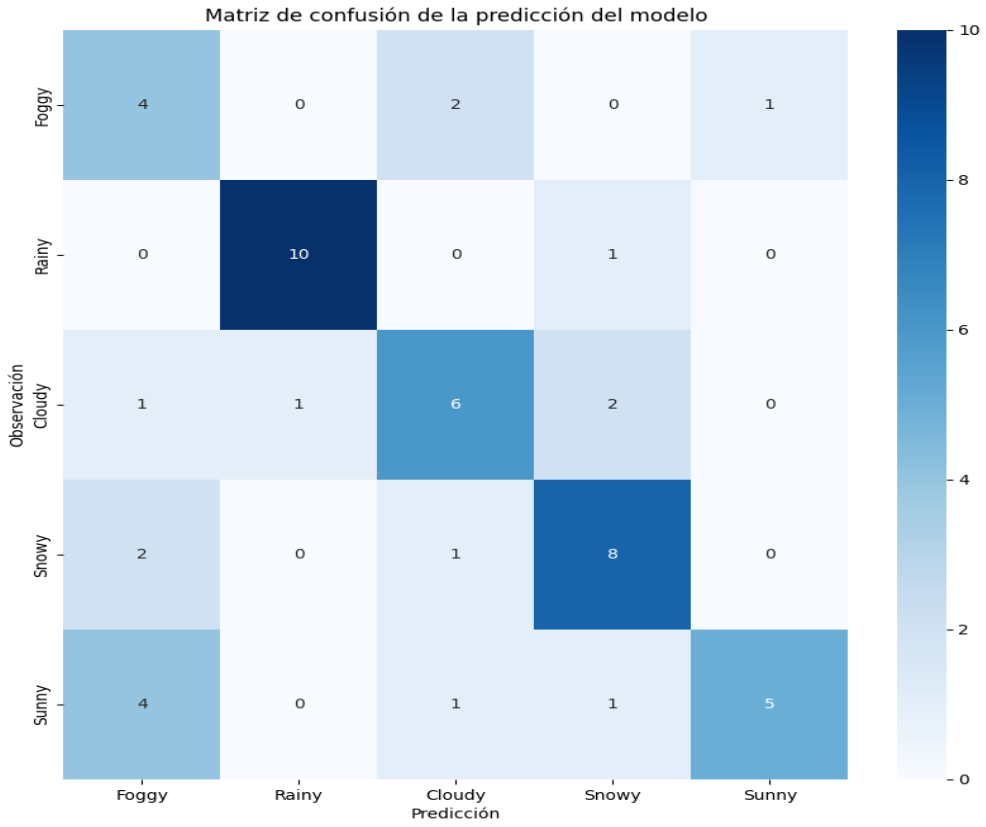
Test Loss: 1.65754
Test Accuracy: 64.00%



3. Modelo MobileNet

Es un modelo diseñado para ser eficiente en términos de consumo de recursos computacionales y memoria, lo que lo hace adecuado para dispositivos con limitaciones. Después de completar el proceso de entrenamiento y evaluación en nuestro conjunto de datos, MobileNet obtuvo los siguientes resultados:

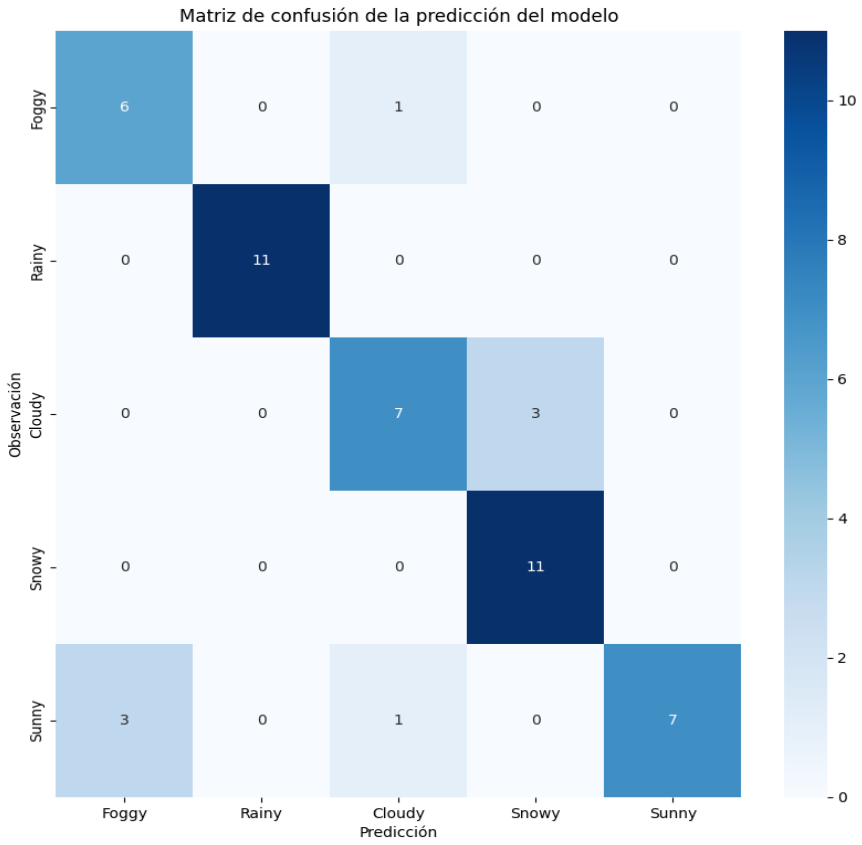
Test Loss: 0.97579
Test Accuracy: 66.00%



4. Modelo ResNet-101

Es una versión más profunda que ResNet50, lo que permite capturar patrones y características aún más complejas en los datos. Después de completar el proceso de evaluación, ResNet101 logró:

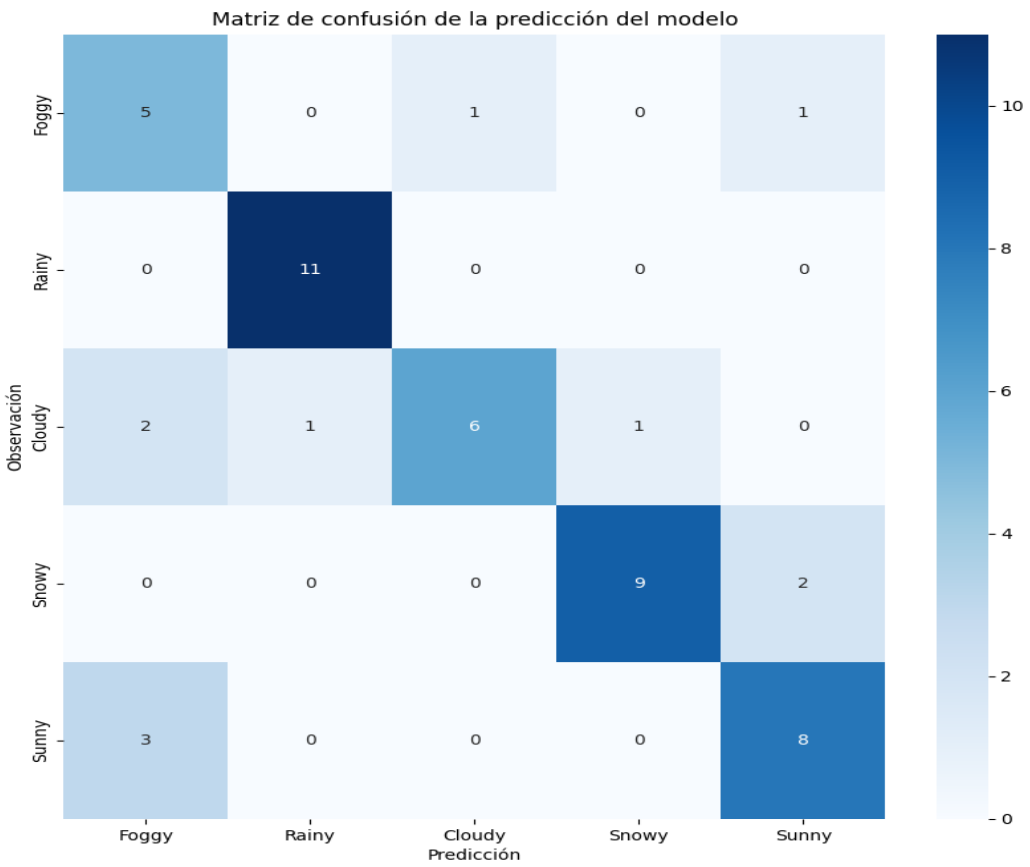
Test Loss: 0.55185
Test Accuracy: 84.00%



5. Modelo DenseNet-201

se caracteriza por su arquitectura densamente conectada, donde cada capa recibe entradas directas de todas las capas anteriores. Esta estructura promueve el flujo de información y características a través de la red. Después de la evaluación, DenseNet201 mostró el siguiente rendimiento:

Test Loss: 0.55127
Test Accuracy: 78.00%



Discusión

Al revisar los resultados anteriormente presentados podemos ver que:

- Al menos tres modelos (ResNet50, MobileNet y ResNet101) tienen un buen rendimiento en las clases "Foggy" y "Snowy". Seguramente es debido a la calidad de las imágenes de esta clase con respecto a las otras tres.
- El modelo ResNet101 obtuvo la precisión global más alta (84%) con respecto a los otros 4 modelos. Además, logra altos valores de precisión y recall cercanos a 1 en las clases “Foggy” y “Snowy”, mostrando entonces una gran performance en estas clases.
- DenseNet201 obtiene un buen equilibrio en la clasificación de todas las clases, con valores de precisión y recall buenos en general.
- La precisión global del modelo VGG19 es más baja en comparación con otros modelos, con una precisión promedio del 64%.

Conclusión

En esta evaluación comparativa de cinco modelos populares para la clasificación multiclase, ResNet101 se destacó con la mayor precisión promedio, seguido de cerca por DenseNet101. Estos resultados sugieren que las arquitecturas más profundas y densamente conectadas tienden a capturar mejor las características discriminativas en conjuntos de datos complejos como el que se utilizó en este estudio. Sin embargo, contar con un conjunto de datos más amplio y de mayor calidad podría proporcionar una mejor performance de cada modelo.