DRAGONVISION – CNN CON CIFAR-10

José Nicolás García Castillo

Digitech FP Málaga

Sistemas de Aprendizaje Automático

12/12/2025

ÍNDICE

1. Objetivos y contexto

El objetivo principal de la práctica es diseñar, entrenar y evaluar una red neuronal convolucional (CNN) que tenga la capacidad de clasificar imágenes en un conjunto de datos (dataset) CIFAR-10. Este dataset tiene un gran uso para la visión por computador y aprendizaje profundo, ya que permite evaluar la capacidad de los modelos para reconocer patrones visuales complejos en imágenes reales a un tamaño reducido.

Para la realización de esta práctica, se han seguido y realizado los siguientes pasos:

1. Carga y exploración del conjunto de datos CIFAR-10.

2. Evaluación y preprocesado de las imágenes y su clasificación.

3. Creación y definición de una arquitectura CNN basada en capas convolucionales y densas.

4. Entrenamiento del modelo utilizando técnicas de optimización y de regularización.

5. Evaluación del rendimiento del modelo mediante métricas de precisión y análisis de resultados.

Con estos pasos, lo que se quiere conseguir es automatizar la clasificación de imágenes en 10 categorías diferentes, obteniendo un modelo sólido y con buena capacidad de generalización sobre los datos que se obtienen.

2. Conjunto de datos

A raíz de la información que disponemos sobre el conjunto de datos de CIFAR-10, tenemos más de 60.000 imágenes a color, de tamaño 32x32 píxeles, distribuidas en 10 clases diferentes, con unas 6.000 imágenes por cada clase, que están compuestas por:

- Avión
- Automóvil
- Pájaro
- Gato
- Ciervo
- Perro
- Rana
- Caballo
- Barco
- Camión

Estas clases representa una variedad de objetos comunes, desde vehículos hasta animales, lo que hace que el conjunto de datos sea adecuado para realizar la evaluación de la capacidad de los modelos de visión artificial para poder distinguir diferentes categorías visuales.

Además, el dataset se ha dividido de la siguiente manera:

- 50.000 imágenes de entrenamiento, que se usa para entrenar el modelo, y así permitir el aprendizaje de las características y patrones visuales de las diferentes clases.

- 10.000 imágenes de prueba, que sirve para evaluar el rendimiento del modelo entrenado, dando lugar a su capacidad de clasificar datos no vistos previamente.

Debido a la calidad de las imágenes y su gran variedad, el dataset CIFAR-10 es un conjunto de datos adecuado para la evaluar modelos de visión artificial de forma sencilla.

3. Preprocesado de datos

Para realizar el entrenamiento del modelo, se ha realizado un preprocesado de los datos, con el objetivo de mejorar la estabilidad y el rendimiento de la red neuronal, como podemos ver en la siguiente imagen:

Preprocesado de datos. CNN

```
X_train_cnn = x_train[:5000]
y_train_cnn = y_train[:5000]

X_val_cnn = x_train[4000:5000]
y_val_cnn = y_train[4000:5000]

# El test se usa completo
X_test_cnn = x_test
y_test_cnn = y_test

# 2) Normalizar al rango [0,1]
X_train_cnn = X_train_cnn.astype("float32") / 255.0
X_val_cnn   = X_val_cnn.astype("float32") / 255.0
X_test_cnn  = X_test_cnn.astype("float32") / 255.0

# 3) Asegurar formato correcto para Keras (enteros)
y_train_cnn = y_train_cnn.astype("int32")
y_val_cnn   = y_val_cnn.astype("int32")
y_test_cnn  = y_test_cnn.astype("int32")

# Mostrar formas finales
print("X_train_cnn:", X_train_cnn.shape)
print("y_train_cnn:", y_train_cnn.shape)

print("X_val_cnn:", X_val_cnn.shape)
print("y_val_cnn:", y_val_cnn.shape)

print("X_test_cnn:", X_test_cnn.shape)
print("y_test_cnn:", y_test_cnn.shape)
```

Esta imagen corresponde a la creación del preprocesado de las imágenes, en el que se han realizado la normalización de las imágenes, escalando los valores de los píxeles al rango [0,1], dividiéndolos entre 255, y se han utilizado unas 5000 imágenes para el preprocesado de los datos, realizando una separación entre el conjunto de entrenamiento y el conjunto de prueba.

4. Diseño y entrenamiento de la CNN

EL modelo que se ha implementado es una red neuronal convencional secuencial, compuestas por varias capas que permiten extraer características jerárquicas de las imágenes. El diseño que se ha realizado se puede ver en la siguiente imagen:

Diseño de modelo. CNN

```
cnn_model = keras.Sequential()

cnn_model.add(layers.Input(shape=(32, 32, 3)))

cnn_model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
cnn_model.add(layers.MaxPool2D((2, 2)))

# Segunda capa de convolución + max pooling
cnn_model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
cnn_model.add(layers.MaxPool2D((2, 2)))

# (Opcional) tercera convolución para extraer características más complejas
cnn_model.add(layers.Conv2D(256, (3, 3), activation="relu", padding="same"))
cnn_model.add(layers.MaxPool2D((2, 2)))

cnn_model.add(layers.Conv2D(256, (3, 3), activation="relu", padding="same"))
cnn_model.add(layers.MaxPool2D((2, 2)))

# Aplanar y capas densas finales
cnn_model.add(layers.Flatten())
cnn_model.add(layers.Dense(128, activation="relu"))
#model.add(layers.Dropout(0.5))  # para reducir sobreajuste
cnn_model.add(layers.Dense(10, activation="softmax"))

# Al final, mostrar el resumen del modelo
cnn_model.summary()
```

Para el diseño del modelo, primero se crea un modelo secuencial, lo que hace que se añada una capa de tras de otra, en orden lineal. Después, se define la forma de las entradas de las imágenes, que en este caso es de 32x32, con 3 canales que representan el color RGB, lo que indica al modelo cómo son los datos de entrada. Una vez definida las entradas, creamos las capas convolucionales del modelo. En este caso, se ha creado un modelo con 4 capas convolucionales. Todas las capas conservan el tamaño espacial de la imagen (padding = “same”), y utilizan MaxPooling para disminuir la exigencia del modelo y evitar el sobreajuste. Además, se han utilizado en las capas un número diferente de filtros, buscando un equilibrio y el número óptimo de filtros para el entrenamiento del modelo, consiguiendo de esta manera la mayor precisión posible del modelo. Una vez creadas las capas convolucionales, se hace uso del

aplanado de las capas para poder conectar estas capas a las capas densas. Después, se añaden capas densas finales para la realización del modelo.

Una vez que se ha diseñado el modelo de entrenamiento, procedemos a compilarlo y al entrenamiento el modelo. Para la compilación del modelo, se ha utilizado el optimizador Adam, en el que se han obtenido mejores resultados, y usamos la métrica de precisión (accuracy) para medir la precisión del modelo. Para el entrenamiento, se han usado 22 épocas y una cantidad de 64 filtros, que es el número de filtros óptimo que se ha encontrado para evitar el sobreajuste. Todo esto lo podemos ver en las siguientes imagenes:

Compilación y entrenamiento. CNN

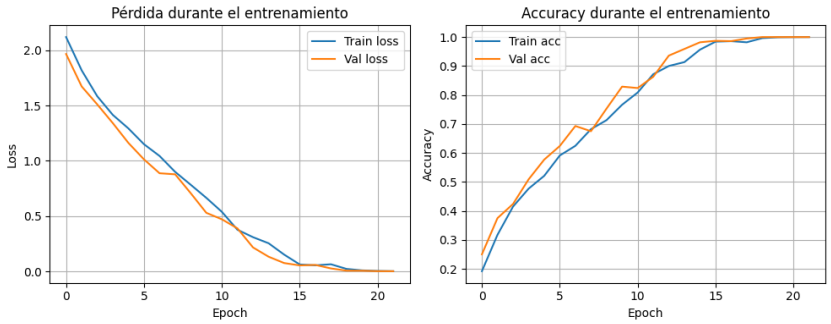
```
optimizer_name = "adam"# TODO: prueba a cambiarlo
loss_name = "sparse_categorical_crossentropy"
metrics_list = ["accuracy"]  # TODO: puedes añadir más métricas si quieres

cnn_model.compile(
    optimizer=optimizer_name,
    loss=loss_name,
    metrics=metrics_list
)
```

```
epochs = 22
batch_size = 64

history = cnn_model.fit(
    X_train_cnn, y_train_cnn,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_val_cnn, y_val_cnn),
    verbose=1
)
```

Los resultados que se han obtenido son los siguientes:



Precisión y pérdidas. CNN

```
Pérdida en test: 2.4558
Accuracy en test: 0.6250
```

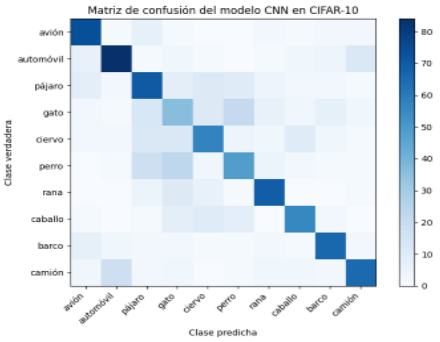
En este modelo, se ha obtenido una precisión de 0.6250, y como podemos observar en la imagen, el modelo alcanza su precisión máxima y sus pérdidas mínimas a partir de las 20 épocas, lo que da a entender que a partir de ese número el modelo tendría sobreajuste.

5. Análisis de errores de la CNN

Para estudiar la precisión y mejora del modelo, se hace un análisis de errores de la red convolucional. Primero realizamos una matriz de confusión para obtener las predicciones de la CNN sobre el conjunto de test, y así poder observar qué clases pueden tener más errores. Los resultados se pueden ver en las siguientes imágenes:

Resultados de la matriz de confusión. CNN

```
Shape y_true: (1000,)
Shape y_pred: (1000,)
Matriz de confusión (valores absolutos):
[[74  1  7  1  0  0  2  1  2  2]
 [ 6 84  1  3  1  0  1  3  4 12]
 [ 8  2 70  8 11 10  4  2  2  1]
 [ 2  1 13 36 11 21  6  3  7  3]
 [ 2  2 12 12 57  4  3 10  3  1]
 [ 0  1 18 23  3 48  5  2  1  1]
 [ 0  0  5 11  6  1 69  0  0  1]
 [ 1  0  1  8 10  8  0 56  2  0]
 [ 7  3  2  2  1  1  2  1 66  2]
 [ 3 18  2  3  0  1  3  3  2 65]]
```



Como podemos observar, la clase que menores errores puede cometer en las predicciones es la de avión, mientras que la clase con más errores es la de gato.

Para confirmar los aciertos y errores del modelo de predicción sobre las clases, realizamos un algoritmo en el que se encarga de visualizar las clases mejor y peor clasificadas mediante las imágenes del conjunto de datos. Para ello se han cogido 1.000 imágenes de clases diferentes, y los resultados que se han obtenido son los siguientes:

Resultados del modelo. CNN

```
Clase 0 (avión): accuracy = 0.8222  (muestras: 90)
Clase 1 (automóvil): accuracy = 0.7304  (muestras: 115)
Clase 2 (pájaro): accuracy = 0.5932  (muestras: 118)
Clase 3 (gato): accuracy = 0.3495  (muestras: 103)
Clase 4 (ciervo): accuracy = 0.5377  (muestras: 106)
Clase 5 (perro): accuracy = 0.4706  (muestras: 102)
Clase 6 (rana): accuracy = 0.7419  (muestras: 93)
Clase 7 (caballo): accuracy = 0.6512  (muestras: 86)
Clase 8 (barco): accuracy = 0.7586  (muestras: 87)
Clase 9 (camión): accuracy = 0.6500  (muestras: 100)

Mejor clase:
  0 (avión), accuracy = 0.8222
Peor clase:
  3 (gato), accuracy = 0.3495

=== Pares de clases más confundidas (real -> predicha) ===
Top 5 pares de clases más confundidas:

1. Real: 5 (perro)  Predicha: 3 (gato),  veces: 23
2. Real: 3 (gato)  Predicha: 5 (perro),  veces: 21
3. Real: 5 (perro)  Predicha: 2 (pájaro),  veces: 18
4. Real: 9 (camión)  Predicha: 1 (automóvil),  veces: 18
5. Real: 3 (gato)  Predicha: 2 (pájaro),  veces: 13

=== Visualización de algunas imágenes mal clasificadas ===
Número total de imágenes mal clasificadas: 375
```

Como podemos observar en la imagen, los resultados obtenidos son similares a los que nos muestra la matriz de confusión, que nos muestra que la clase con mejor precisión es la de avión con una precisión de 0.8222, mientras que la peor clase es la de gato, con una precisión de 0.3495. Además, nos muestra el número de veces que el modelo ha confundido la clasificación de las imágenes, lo que coincide con la precisión que se ha obtenido del modelo de convolución, que era de 0.625. Aunque el modelo tiene todavía un margen de mejora, la CNN demuestra una buena capacidad de generalización y una correcta separación entre la mayoría de las clases, lo que indica que el comportamiento es coherente con las limitaciones de CIFAR-10 y con la arquitectura del modelo que se ha utilizado y que, además, ha sido la que mejores resultados se han obtenido y, dependiendo del número de capas o de filtros, el modelo puede variar sus resultados, dando lugar a que sean peores o mejores.

6. Conclusión

En conclusión, en esta práctica se ha demostrado la eficacia de las **redes neuronales convolucionales** para la clasificación automática de imágenes utilizando el conjunto de datos CIFAR-10. El modelo es capaz de aprender características visuales relevantes y generalizar su conocimiento a datos nuevos. Se puede concluir que el preprocesado adecuado de los datos es importante y clave para el buen rendimiento del modelo, y que las redes convolucionales son especialmente adecuadas para las tareas de visión por computador frente a modelos tradicionales, y que pueden existir clases con mayor dificultad visual, lo que hace que la precisión máxima del modelo sea limitada y se haga complicado sacar una mayor precisión y mejores resultados. Como posibles mejoras que se pueden aplicar en el modelo, se podrían introducir otro tipo de técnicas u otras arquitecturas más profundas, con el objetivo de mejorar aún más la precisión del modelo y mejorar resultados.