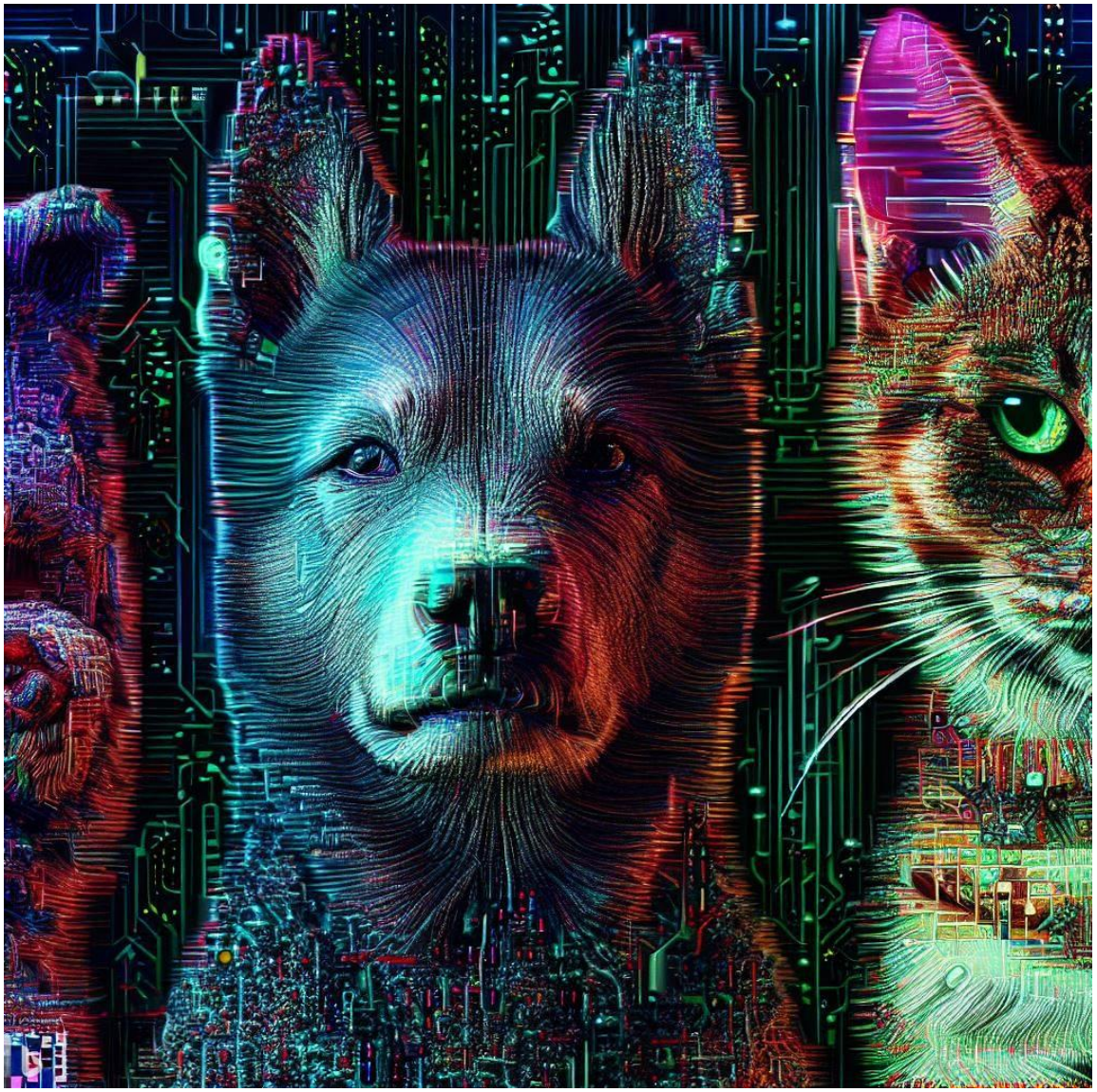
Reconeixement d'Imatges amb xarxes neuronals convolucionals



Kirian Rodríguez Alonso

Algorísmia i Programació Audiovisual

1 de juny de 2023

Kirian Rodríguez Alonso

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

ÍNDEX

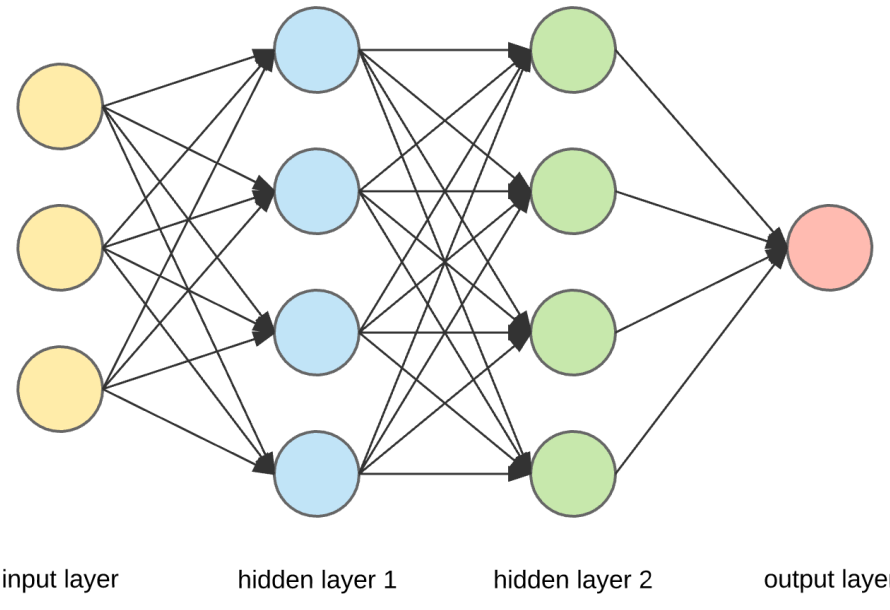
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

1. Objectiu

L'objectiu final d'aquest projecte és aconseguir que a partir d'una sèrie d'imatges, el programa sigui capaç de classificar una imatge donada segons les classes de CIFAR-10.

Un cop assolit, fent ús de la interfície tkinter, crearé una aplicació amb interfície gràfica per poder veure d'una forma gràfica el funcionament del projecte.

La idea seria la següent: Iniciar l'app i seleccionar un directori. En aquest directori ha d'haver-hi imatges preparades per fer de test. En el meu cas, les imatges haurien de ser de la classe CIFAR-10. I que mitjançant el codi creat amb python, que tingui un output de la predicció del model.



Imatge 1. Font: Opengenus

2. Disseny i desenvolupament de l'aplicació

CIFAR-10

El dataset utilitzat i únic protagonista del codi és el CIFAR-10. L'elecció de CIFAR-10 és degut al fet que ja vaig fer ús a l'assignatura de Gestió de Sistemes Audiovisuals (GDSA) d'aquest dataset. Per tant, ja tinc una mica d'experiència en el seu ús.

El dataset CIFAR-10 consisteix en 60000 imatges 32x32 a color i de 10 classes, per tant, cada classe té 6000 imatges. Com farem ús només de gats i gossos, seran un total de 12000, de les quals seran 10000 d'entrenament i 2000 per als tests.

És important esmentar que no existeix 'overlap', o sigui, que cada classe és totalment exclusiva del seu tipus.

Per tant, si fes ús de la classe 'truck' i 'automobile', no hi hauria overlap, ja que a 'truck' només hi ha camions grans dimensions i a 'automobile' només sedans, SUV, etc. Cap dels dos tenen pick-ups per evitar problemes.

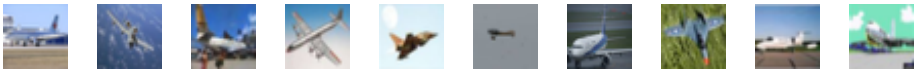
Més endavant faré una comparació dels resultats obtinguts amb imatges de pick-up, encara que probablement el sistema predirà (segons la posició o distribució de la imatge) 'truck' o 'automobile'.

Kirian Rodríguez Alonso

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

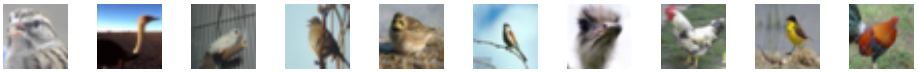
CIFAR- 10 té un total de 10 classes que es divideixen de la següent forma:

airplane

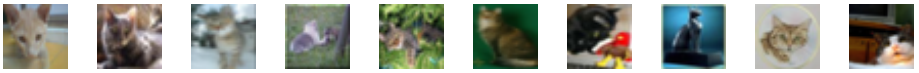


automobile

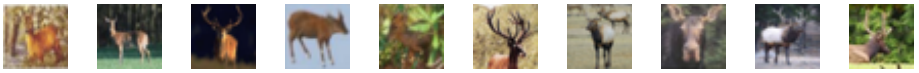
bird



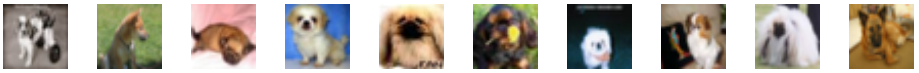
cat



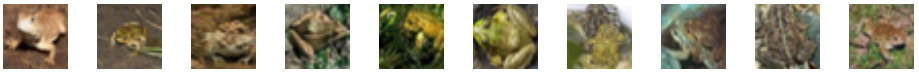
deer



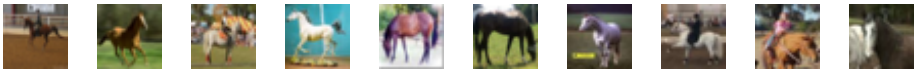
dog



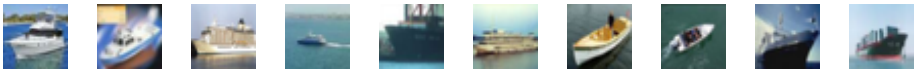
frog



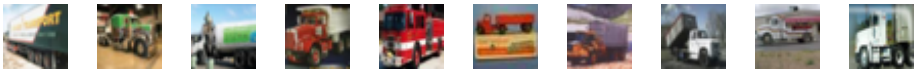
horse



ship



truck



Font: [Computer Science University Of Toronto]

MODEL

Respecte al model, he de dir que les pràctiques de GDSA m'han ajudat a triar les capes convolucionals.

En primer lloc, el model es basa en capes convolucionals de 64 fins a 512 filtres de mida 3x3 amb funció d'activació ReLU (0-1), ja que indica si una imatge determinada és una classe o no. Aquestes son seguides de capes MaxPooling (per reduir dimensionalitat i extreure característiques rellevants).

A continuació s'utilitza una capa Flatten per passar la informació dels mapes 2D a un vector unidimensional per a permetre l'entrada a les capes Fully Connected.

Finalment s'afegeixen una sèrie de capes Fully Connected per a extreure característiques de les imatges encara més complexes, aquestes fan ús de la funció d'activació ReLU, menys l'última utilitzant una funció d'activació Softmax per a produir la distribució de probabilitat per a la classificació de les 10 classes del CIFAR-10.

Cada capa fa ús de Dropout per a aconseguir evitar el *overfitting*, per tant fa que la xarxa sigui més robusta i dongui millors prediccions. Un cop entrenat, el model es desa en un fitxer anomenat *modelo_completo.h5* i utilitzar per al programa *app.py*.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Aplicació reconeixement d'imatges

Per a fer ús de l'aplicació és necessari executar el programa *app.py*. Com he dit abans, s'ha d'ajustar la ruta del directori per a accedir a l'arxiu *modelo_completo.h5*.

Una vegada executat el programa, aquest carrega el model pre-entrenat. A continuació una vegada l'usuari a importat la imatge a classificar, aquesta es mostra a l'aplicació. Per continuar, l'usuari ha de pressionar el botó *predir*. En aquest punt, l'aplicació utilitza el model carregat per a realitzar la predicció de la imatge, i el resultat és mostrat per text indicant la classe que considera més semblant.

L'aplicació mostra un títol i sota una llista de les classes del conjunt CIFAR-10. L'usuari ha de fer clic al botó *Cargar imagen* per a poder seleccionar la imatge a clasificar. Una vegada seleccionada la imatge, el botó *Predecir* s'habilitarà, si l'usuari en fa clic l'aplicació amb el model prèviament carregat farà una predicció de la imatge seleccionada. Per finalitzar la classe de la imatge sera mostrada amb el seguent missatge de text: *La imagen se parece a CLASSE_CIFAR10*. On *CLASSE_CIFAR10* serà una de les 10 classes del dataset.

Aquesta interfície gràfica proporciona una forma clara i molt intuïtiva per a carregar i predir les imatges que vulguem, mostrant el resultat de la predicció a la pantalla de l'usuari.

3. Prestacions i limitacions

Prestacions

El programa no necessita descarregar l'arxiu sencer del dataset CIFAR-10 (163 MB), ja que aquest fa ús de la llibreria *cifar10*. Amb aquesta llibreria podem carregar tot el dataset utilitzant la funció *cifar10.load_data()* que incorpora Keras Datasets.

El codi fa ús de la biblioteca *tkinter* per a poder crear interfície gràfica, això ens permet crear una app molt senzilla que permet a l'usuari classificar imatges de forma que no hagi de pensar en res més.

Una prestació que penso que és molt útil és la previsualització de la imatge a l'aplicació una vegada carregada, així l'usuari s'assegura al 100% que la imatge seleccionada és la que realment volia classificar.

Vaig incorporar l'opció que una vegada obert el programa, l'usuari només pot pressionar un botó, que és l'encarregat d'obrir la imatge. El botó encarregat de predir només s'habilitarà una vegada s'ha carregat correctament la imatge. En cas que l'usuari volgués carregar una altra imatge ho pot fer sense cap problema pressionant de nou el botó de carregar imatge.

En cas que vulguem utilitzar un altre model, només haurem de canviar la ruta del nou arxiu al codi, per tant, no hem d'incorporar tot el model al codi evitant així errors i manipulacions innecessàries al codi.

Limitacions

Si volem compilar el codi, hem de tenir instal·lats les dependències *matplotlib*, *sklearn* i *tensorflow*. Per fer-ho s'ha d'executar al terminal aquestes dues comandes:

```Python
pip install tensorflow
pip install scikit-learn
pip install matplotlib
```

S'ha de dir que per a compilar el programa s'ha de fer ús de la GPU, per tant, si tens una GPU de baixos recursos, el programa compilarà, però el temps serà excessivament alt. En aquests casos, recomano utilitzar Google Colaboratory. Amb *Colab* el procés és molt ràpid, ja que fa ús de processament al núvol per compilar el programa, igualment ens hem d'assegurar que aquest fa ús de la GPU que té l'entorn d'execució. Per activar aquesta opció hem de seguir els següents passos:
Entorn d'execució -> Canviar tipus entorn d'execució -> Accelerador per Hardware -> GPU -> Guardar

Per a fer ús de l'aplicació ens hem d'assegurar de canviar el directori a un on es troba l'arxiu '*modelo_completo.h5*'. Es troba a la següent línia de codi:

```Python
modelo=tf.keras.models.load_model("C:\\modelo_completo.h5")
```

Respecte a la predicció del sistema, una característica que té és que no contempla les furgonetes o pick-ups, per tant, segons la distribució de la foto o segons l'element que més destaqui a la imatge, serà la predicció que prengui.

	
Camión	Automóvil

També hi ha errors constants en els camions tipus cisterna, on els detecta com avions, o en els cérvols sense banyes que els classifica com a cavalls.

Al mateix temps he observat que hi ha errors constants en el sistema de classificació de les imatges. Els camions tipus cisterna són detectats erròniament com a avions, cosa que pot generar confusió en el procés de classificació degut a la forma de la cisterna. A més a més, hi ha situacions en què els cérvols sense banyes són incorrectament classificats com a cavalls.

4. Conclusions

Respecte a les conclusions, és important esmentar l'objectiu final de desenvolupar una aplicació amb les eines de classificar imatges amb CIFAR-10. Mitjançant l'ús de la interfície gràfica *tkinter*, s'ha pogut fer de forma que permet visualitzar de forma molt senzilla el funcionament del projecte.

Aquest projecte romandrà pujat al meu repositori de GitHub, ja que hi ha oportunitats per a futures millores. Millores com l'augment de la precisió del model o l'incorporació a noves funcionalitats com així la capacitat de classificar una serie d'imatges alhora.

5. Annex 1: Com executar l'aplicació

En primer lloc, s'han d'instal·lar les llibreries TensorFlow, scikit-learn i matplotlib manualment, per tant s'ha d'introduir al terminal aquestes comandes:

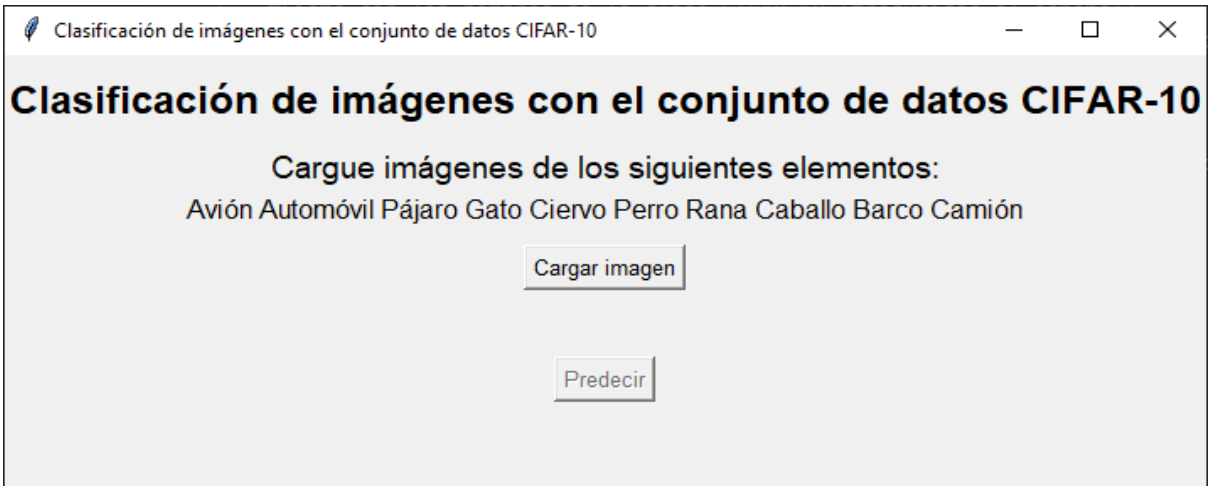
```Python
pip install tensorflow
pip install scikit-learn
pip install matplotlib
```

En segon lloc, s'ha de executar el codi *modelo.py*. Aquest codi una vegada entrenat el model i estigui el codi sencer compilat, guardara a la ruta de l'aplicació un arxiu anomenat *modelo_completo.h5*. A més a més, també imprimirà la precisió de cada classe i donara dues gràfiques una de la curva d'error, i un altre amb la de precisió.

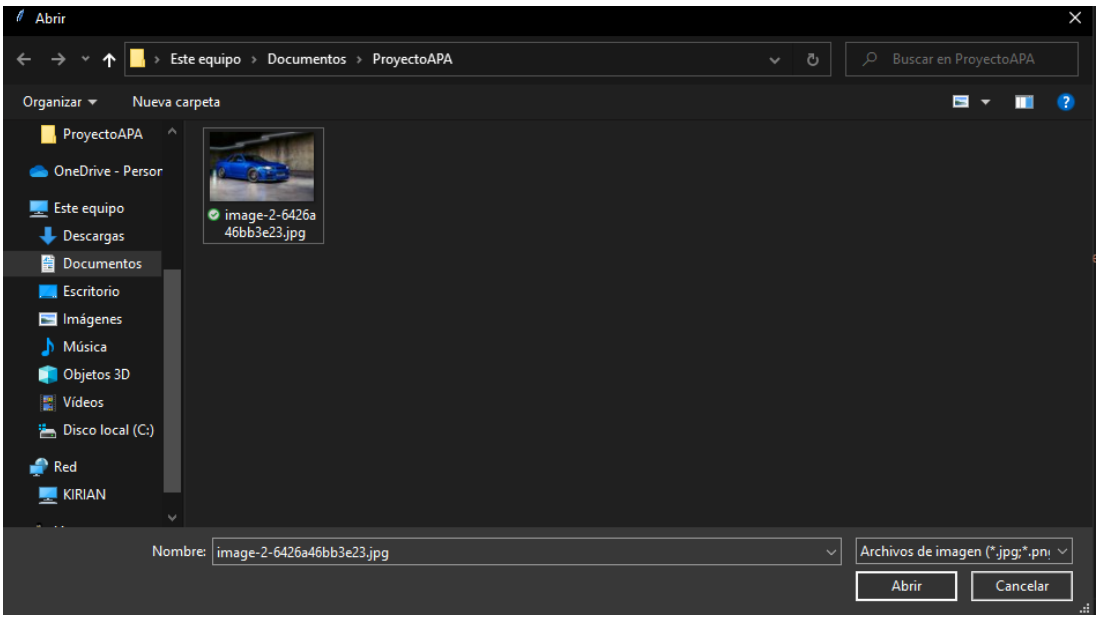
En tercer lloc, haurem de canviar al codi *app.py*. En aquest codi, hem de modificar la ruta de l'arxiu i posar la del nostre sistema. A continuació un exemple de com hauria de ser:

```python
modelo = tf.keras.models.load_model("C:\\Users\\Kirian\\Downloads\\modelo_completo.h5")
```

En quart lloc, una vegada modificada la ruta, haurem d'executar el codi. L'aplicació s'obrirà, i mostrarà la següent finestra:



Com a cinquè pas, s'ha de clicar l'opció *Cargar imagen*. S'obrirà un quadre per seleccionar la imatge que vulguem dels nostres arxius. Una vegada triada la imatge a testejar, li donem a *Abrir archivo*. S'ha d'esmentar que només accepta arxius en format JPG y PNG.



Com a sisè pas, hem de pressionar el botó de *Predecir*, aquest botó farà que l'aplicació ens digui el resultat de la predicció per text a l'aplicació. L'aplicació permet repetir el procés amb diferents imatges, per obtenir les prediccions de les noves imatges.

6. Annex 2: Codi de l'aplicació

Codi *modelo.py*

Python

```
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

def crear_modelo():

    #Creacion de la secuencia
    modelo = Sequential()
     #Capa convolucional de 32 filtros 3x3. padding = 'same' para que
el output tenga el mismo tamaño que input. ReLU para 0 o 1
     modelo.add(Conv2D(32,(3,3),padding='same',input_shape=(32,32,3),
activation = 'relu'))
    modelo.add(Conv2D(32,(3,3), activation = 'relu'))
    #Agrega Capa Maxpooling
    modelo.add(MaxPooling2D(pool_size=(2,2)))
    #Evita overfitting
    modelo.add(Dropout(0.25))

    modelo.add(Conv2D(64,(3,3),padding='same', activation = 'relu'))
    modelo.add(Conv2D(64,(3,3), activation = 'relu'))
    modelo.add(MaxPooling2D(pool_size=(2,2)))
    modelo.add(Dropout(0.25))

     #Capa Flatten o de aplanamiento para pasar de mapas 2D a vector
1D
    modelo.add(Flatten())
     #Fully connected para ayudar a aprender las caracteristicas y
relaciones mas complejas
    modelo.add(Dense(512, activation = 'relu'))
    modelo.add(Dropout(0.25))
    #Softmax para sacar la distribucion de probabilidad, tamaño 10 ->
Cifar-10 tiene 10 clases
```

```
    modelo.add(Dense(10, activation = 'softmax'))

    return modelo

def    entrenar_modelo(modelo,    x_entrenamiento,    y_entrenamiento,
x_prueba, y_prueba):
     #Compilacion del modelo anterior, 'categorical_crossentropy' ->
clasificacion de multiples clases
    modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    #Entrenamiento especificando los hiperparametros necesarios, este
training se guarda en 'historial'
         historial   =   modelo.fit(x_entrenamiento,   y_entrenamiento,
batch_size=200, epochs=10, validation_data=(x_prueba, y_prueba))

    # Extraen las métricas de entrenamiento y prueba del historial de
entrenamiento
    perdida_entrenamiento = historial.history['loss']
    perdida_prueba = historial.history['val_loss']
    precision_entrenamiento = historial.history['accuracy']
    precision_prueba = historial.history['val_accuracy']

    # Representar gráficamente las curvas de error de entrenamiento y
prueba
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(perdida_entrenamiento, label='Entrenamiento')
    plt.plot(perdida_prueba, label='Prueba')
    plt.title('Curva de error')
    plt.xlabel('Épocas')
    plt.ylabel('Error')
    plt.legend()

    # Representar gráficamente la curva de precisión de prueba
    plt.subplot(1, 2, 2)
    plt.plot(precision_prueba)
    plt.title('Curva de precisión')
    plt.xlabel('Épocas')
    plt.ylabel('Precisión')

    plt.tight_layout()
    plt.show()
```

```
    # Definicion de las Id de las clases
        cifar10_clases = ['avión', 'automóvil', 'pájaro', 'gato', 'ciervo', 'perro', 'rana', 'caballo', 'barco', 'camión']

     # Obtencion de las predicciones del modelo utilizando los datos de prueba
    y_prediccion = modelo.predict(x_prueba)
    # Argmax -> obtiene los índices con mas % de acierto
    y_prediccion = np.argmax(y_prediccion, axis=1)
    # Se compara las predicciones con los valores reales
      precision_clases = accuracy_score(np.argmax(y_prueba, axis=1), y_prediccion)
    # Resultados de cada clase
         reporte_clases = classification_report(np.argmax(y_prueba, axis=1), y_prediccion, output_dict=True)

    # Mostrar la precisión de cada clase
    for class_id, class_metrics in reporte_clases.items():
          if class_id.isdigit(): # Necesario ya que 'accuracy' se encuentra dentro de reporte_clases.items()
        class_name = cifar10_clases[int(class_id)]
        accuracy = class_metrics['precision']
            print(f'Precisión de la clase {class_name}: {accuracy * 100:.2f}%')



def main():

    # Cargar el dataset CIFAR-10
        (x_entrenamiento, y_entrenamiento), (x_prueba, y_prueba) = cifar10.load_data()

    # Normalizar las imágenes (0-1) y
    x_entrenamiento = x_entrenamiento.astype('float32') / 255
    x_prueba = x_prueba.astype('float32') / 255

    # Convertir las etiquetas a one-hot encoding
         # One-hot -> Convierte las etiquetas en vectores binarios de tamaño 10 donde  el índice de la clase es 1 y el resto  0
    y_entrenamiento = to_categorical(y_entrenamiento)
    y_prueba = to_categorical(y_prueba)
```

```
    # Crear el modelo de la CNN
    modelo = crear_modelo()

    # Entrenar el modelo y obtener métricas
         entrenar_modelo(modelo,  x_entrenamiento,  y_entrenamiento,
x_prueba, y_prueba)

    # Guardar modelo
    modelo.save("./modelo_completo.h5")
    print("Modelo guardado correctamente.")

if __name__ == '__main__':
    main()
```

Codi *app.py*

Python

```
import numpy as np
from PIL import Image, ImageOps, ImageTk
import tkinter as tk
from tkinter import filedialog as fd #no va sin esta libreria
import tensorflow as tf

#Lista con los nombres de la clase CIFAR-10 (traducidos al español)
nombres_clase = ["Avión", "Automóvil", "Pájaro", "Gato", "Ciervo",
"Perro", "Rana", "Caballo", "Barco", "Camión"]

#Carga modelo ya entrenado por un archivo en una ruta determinada
def carga_modelo():
                                                modelo            =
tf.keras.models.load_model("C:\\Users\\Kirian\\\Documents\\\GitHub\\
Reconeixement-de-Imatges-amb-CNN\\modelo_completo.h5")
    return modelo

#Recibe una imagen y el modelo ya cargado
def importacion_prediccion(datos_imagen, modelo):
    #Ajuste a 32x32
    tamaño = (32, 32)
```

```
    imagen = ImageOps.fit(datos_imagen, tamaño, Image.LANCZOS)
    #Se convierte en array
    img = np.asarray(imagen)
    #Reajuste para que sea compatible con el modelo
    img_reshape = img[np.newaxis, ...]
    #Prediccion
    prediccion = modelo.predict(img_reshape)
    return prediccion

#Permite seleccionar el archivo mediante el comando 'askopenfilename'
def explorar_archivo():
    global ruta_imagen
        ruta_imagen = fd.askopenfilename(filetypes=[("Archivos de imagen", "*.jpg;*.png")])
    if ruta_imagen:
        imagen = Image.open(ruta_imagen)
         imagen = imagen.resize((300, 300))  # Ajustar el tamaño de la imagen para que se vea correctamente
          imagen_foto = ImageTk.PhotoImage(imagen) # Hace compatible la imagen con tkinter
          etiqueta_imagen.configure(image=imagen_foto) #Incorpora la imagen
              etiqueta_imagen.image = imagen_foto  # Mantener una referencia a la imagen para evitar que se elimine de la memoria
        habilitar_boton_prediccion() #Activa el boton

def habilitar_boton_prediccion():
                          boton_prediccion.configure(state="normal", command=predecir_imagen)

#Al presionar el anterior boton, se activa esta funcion
def predecir_imagen():
    #Abre imagen ya seleccionada
    imagen = Image.open(ruta_imagen)
    #Predicciones incorporadas
    predicciones = importacion_prediccion(imagen, modelo)
      #indice_clase -> Indica de que clase es una imagen segun prediccion. ex: indice_clase[0] = avión
        #argmax para mayor probabilidad
    indice_clase = np.argmax(predicciones)
    clase_predicha = nombres_clase[indice_clase]
    #Actualizacion texto
```

```
    etiqueta_resultado.configure(text="La imagen se parece a: " +
clase_predicha)

# Crea la app con tkinter
app = tk.Tk()
app.title("Clasificación de imágenes con el conjunto de datos
CIFAR-10")

# Carga del modelo
modelo = carga_modelo()

# Crear los elementos de la interfaz gráfica
etiqueta_titulo = tk.Label(app, text="Clasificación de imágenes con
el conjunto de datos CIFAR-10", font=("Helvetica", 18, "bold"))

etiqueta_titulo.pack(pady=10)

etiqueta_encabezado = tk.Label(app, text="Cargue imágenes de los
siguientes elementos:", font=("Helvetica", 14))
etiqueta_encabezado.pack()

etiqueta_clase = tk.Label(app, text=nombres_clase,
font=("Helvetica", 12))
etiqueta_clase.pack()

boton_explorar = tk.Button(app, text="Cargar imagen",
command=explorar_archivo, font=("Helvetica", 10))
boton_explorar.pack(pady=10)

etiqueta_imagen = tk.Label(app)
etiqueta_imagen.pack()

boton_prediccion = tk.Button(app, text="Predecir", state="disabled",
font=("Helvetica", 10))
boton_prediccion.pack(pady=10)

etiqueta_resultado = tk.Label(app, font=("Helvetica", 10))
etiqueta_resultado.pack(pady=10)

# Ejecutar aplicacion
app.mainloop()
```