› 1) Configuración del entorno

[ ] ↳ 2 celdas ocultas

⌄ 2) Cargar y explorar los datos

```
ruta_train = '/content/drive/MyDrive/Ev01/data/train'
ruta_test = '/content/drive/MyDrive/Ev01/data/test'
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train_cat.shape}")
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 10)
```

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_5"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_17 (Conv2D)              │ (None, 30, 30, 32)     │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_15 (MaxPooling2D) │ (None, 15, 15, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_11          │ (None, 15, 15, 32)     │           128 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_14 (Dropout)            │ (None, 15, 15, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_18 (Conv2D)              │ (None, 13, 13, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_16 (MaxPooling2D) │ (None, 6, 6, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_12          │ (None, 6, 6, 64)       │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_15 (Dropout)            │ (None, 6, 6, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_19 (Conv2D)              │ (None, 4, 4, 128)      │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_13          │ (None, 4, 4, 128)      │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_16 (Dropout)            │ (None, 4, 4, 128)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_20 (Conv2D)              │ (None, 2, 2, 256)      │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_14          │ (None, 2, 2, 256)      │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_17 (Dropout)            │ (None, 2, 2, 256)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_5 (Flatten)             │ (None, 1024)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_10 (Dense)                │ (None, 512)            │       524,800 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_18 (Dropout)            │ (None, 512)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_11 (Dense)                │ (None, 10)             │         5,130 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 920,266 (3.51 MB)
 Trainable params: 919,306 (3.51 MB)
```

```python
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100000, decay_rate=0.96, staircase=True)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='categorical_crossentropy',
              metrics=['accuracy'])



datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.2,
)

datagen.fit(x_train)

history = model.fit(datagen.flow(x_train, y_train_cat, batch_size=128),
      epochs=30,
      validation_data=(x_test, y_test_cat),
      callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)]
)
```

```
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
391/391 ━━━━━━━━━━━━━━━━━━━━ 45s 91ms/step - accuracy: 0.2356 - loss: 2.5210 - val_accuracy: 0.1187 - val_loss: 4.4226
Epoch 2/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.3890 - loss: 1.6817 - val_accuracy: 0.4318 - val_loss: 1.5459
Epoch 3/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 31s 78ms/step - accuracy: 0.4427 - loss: 1.5474 - val_accuracy: 0.5032 - val_loss: 1.3937
Epoch 4/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.4721 - loss: 1.4687 - val_accuracy: 0.5611 - val_loss: 1.2249
Epoch 5/30
```

```
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.4922 - loss: 1.4159 - val_accuracy: 0.5968 - val_loss: 1.1100
Epoch 6/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5094 - loss: 1.3672 - val_accuracy: 0.5716 - val_loss: 1.1927
Epoch 7/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5268 - loss: 1.3232 - val_accuracy: 0.6281 - val_loss: 1.0484
Epoch 8/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 75ms/step - accuracy: 0.5466 - loss: 1.2859 - val_accuracy: 0.5410 - val_loss: 1.3163
Epoch 9/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 42s 78ms/step - accuracy: 0.5472 - loss: 1.2662 - val_accuracy: 0.5213 - val_loss: 1.4209
Epoch 10/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5615 - loss: 1.2324 - val_accuracy: 0.6186 - val_loss: 1.0827
```

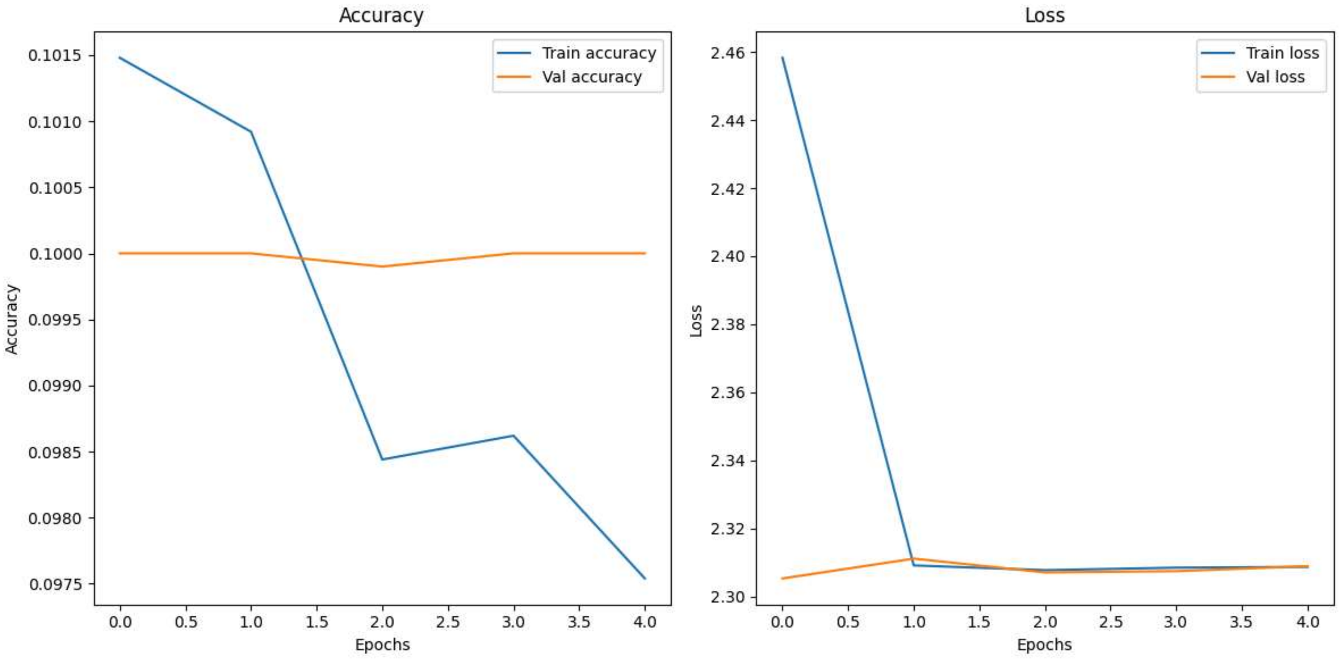
```
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f"Test accuracy: {test_acc * 100:.2f}%")

y_pred = model.predict(x_test)
y_pred_labels = np.argmax(y_pred, axis=1)

print(classification_report(np.argmax(y_test_cat, axis=1), y_pred_labels))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6258 - loss: 1.0420
Test accuracy: 62.81%
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
              precision    recall  f1-score   support

           0       0.74      0.60      0.66      1000
           1       0.78      0.70      0.74      1000
           2       0.65      0.39      0.49      1000
           3       0.57      0.29      0.38      1000
           4       0.61      0.50      0.55      1000
           5       0.58      0.56      0.57      1000
           6       0.54      0.86      0.66      1000
           7       0.59      0.76      0.67      1000
           8       0.75      0.76      0.75      1000
           9       0.58      0.86      0.69      1000

    accuracy                           0.63     10000
   macro avg       0.64      0.63      0.62     10000
weighted avg       0.64      0.63      0.62     10000
```

```
for lr in [0.001, 0.01, 0.1]:
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    history = model.fit(datagen.flow(x_train, y_train_cat, batch_size=128),
                        epochs=5,
                        validation_data=(x_test, y_test_cat))
```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
391/391 ━━━━━━━━━━━━━━━━━━━━ 47s 93ms/step - accuracy: 0.5277 - loss: 1.3270 - val_accuracy: 0.5777 - val_loss: 1.3052
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 75ms/step - accuracy: 0.5426 - loss: 1.2840 - val_accuracy: 0.6098 - val_loss: 1.1100
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5579 - loss: 1.2543 - val_accuracy: 0.6140 - val_loss: 1.0764
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5643 - loss: 1.2301 - val_accuracy: 0.6486 - val_loss: 1.0277
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5766 - loss: 1.2000 - val_accuracy: 0.6034 - val_loss: 1.2162
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 44s 90ms/step - accuracy: 0.2949 - loss: 1.9502 - val_accuracy: 0.3269 - val_loss: 2.1056
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.3940 - loss: 1.6976 - val_accuracy: 0.3215 - val_loss: 2.3465
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.4273 - loss: 1.6100 - val_accuracy: 0.4566 - val_loss: 1.8348
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.4504 - loss: 1.5744 - val_accuracy: 0.4240 - val_loss: 1.6816
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.4587 - loss: 1.5341 - val_accuracy: 0.5682 - val_loss: 1.3093
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 92ms/step - accuracy: 0.1081 - loss: 3.0286 - val_accuracy: 0.1000 - val_loss: 2.3053
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 28s 73ms/step - accuracy: 0.1026 - loss: 2.3082 - val_accuracy: 0.1000 - val_loss: 2.3111
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 73ms/step - accuracy: 0.0961 - loss: 2.3084 - val_accuracy: 0.0999 - val_loss: 2.3070
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.1007 - loss: 2.3085 - val_accuracy: 0.1000 - val_loss: 2.3074
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 41s 73ms/step - accuracy: 0.0993 - loss: 2.3085 - val_accuracy: 0.1000 - val_loss: 2.3089
```

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train accuracy')
plt.plot(history.history['val_accuracy'], label='Val accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train loss')
plt.plot(history.history['val_loss'], label='Val loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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



```
model.save('/content/drive/MyDrive/Deep/modelo.h5')
```

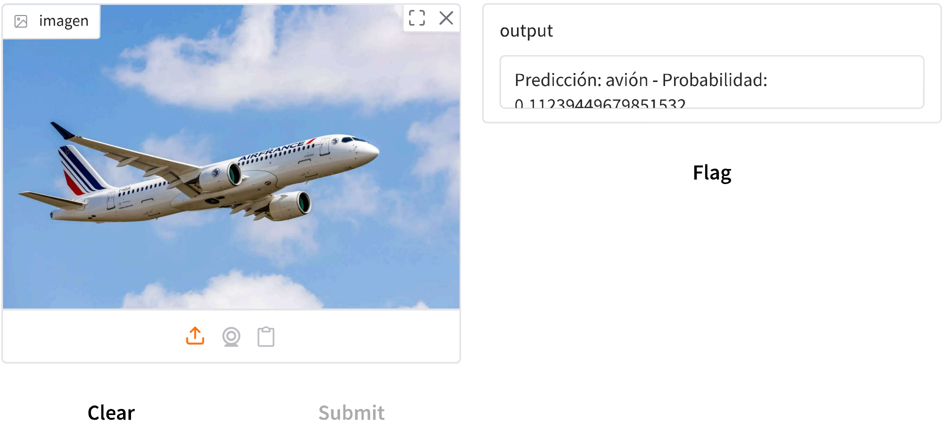
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is (
```

```
model = tf.keras.models.load_model('/content/drive/MyDrive/Deep/modelo.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
```

```
def cargar_imagen(imagen):
    img = image.load_img(imagen, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array.astype('float32') / 255.0
    return img_array

def predecir_imagen(imagen):
    img_preprocesada = cargar_imagen(imagen)
    predicciones = model.predict(img_preprocesada)
    clase_predicha = np.argmax(predicciones, axis=1)
    clases = ['avión', 'automóvil', 'pájaro', 'gato', 'ciervo', 'perro', 'rana', 'caballo', 'barco', 'camión']
    return f"Predicción: {clases[clase_predicha[0]]} - Probabilidad: {np.max(predicciones)}"

interface = gr.Interface(fn=predecir_imagen, inputs=gr.Image(type="filepath"), outputs="text")

interface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatica

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://f663f64c6b7122b29c.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```



Predicción: avión - Probabilidad: 0.11239449679851532