2.0-raa-train-model

May 31, 2025

[1]:
```
%load_ext autoreload
%autoreload 2
```

[ ]:
```
import os
import tensorflow as tf
```

[4]:
```
dataset_dir = os.path.join('data', 'dataloader')
```

[ ]:
```
img_height = 224
img_width = 224
batch_size = 32
```

[ ]:
```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="training",
    seed=42,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="validation",
    seed=42,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

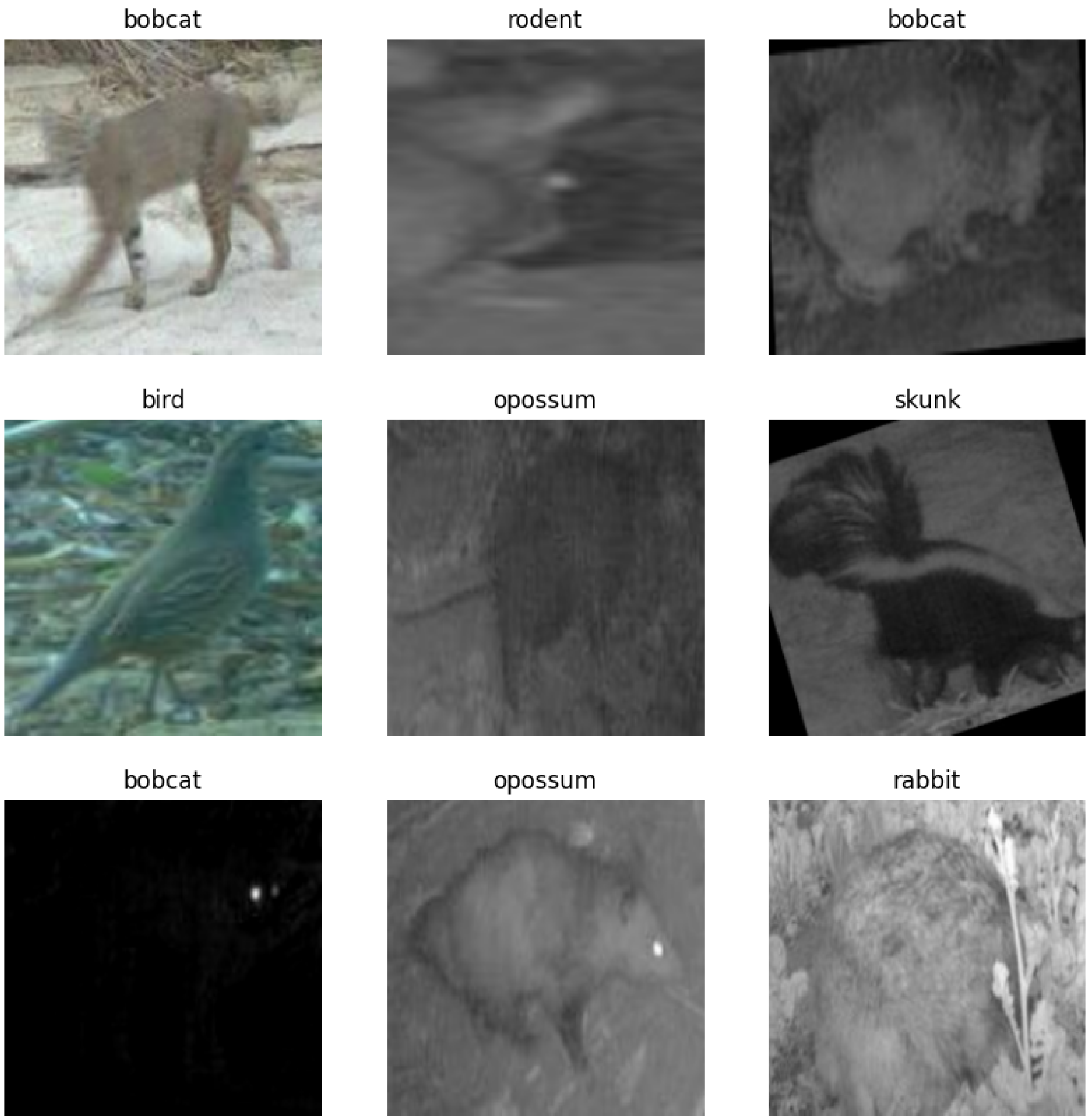
```
Found 19833 files belonging to 12 classes.
Using 15867 files for training.
Found 19833 files belonging to 12 classes.
Using 3966 files for validation.
```

[7]:
```
class_names = train_ds.class_names
print(f"Clases detectadas: {class_names}")
```

```
Clases detectadas: ['bird', 'bobcat', 'car', 'cat', 'coyote', 'dog', 'opossum',
'rabbit', 'raccoon', 'rodent', 'skunk', 'squirrel']
```

[8]:
```python
import matplotlib.pyplot as plt
```

[9]:
```python
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



0.1 Justificación del enfoque experimental

Aunque hemos preparado un dataset con casi 20k recortes, sigue siendo **relativamente pequeño** para entrenar redes neuronales profundas (DNNs) desde cero. Sería idóneo utilizar otras arquitecturas.

¿Por qué?

- Modelos como ResNet, EfficientNet o Vision Transformers se entrenan sobre datasets enormes como **ImageNet** (1M+ imágenes).
- Las DNNs requieren grandes volúmenes de datos para evitar underfitting y aprender desde patrones básicos (bordes) hasta complejos (formas).
- En datasets pequeños, la práctica común es usar **transfer learning**, aprovechando pesos preentrenados y ajustando solo las capas finales.

0.1.1 Referencias relevantes

- EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (arXiv)
- Deep Residual Learning for Image Recognition — ResNet (arXiv)
- Keras Applications: ResNet Models (Keras.io)
- TensorFlow Tutorial: Transfer Learning (TensorFlow.org)
- CS231n Stanford Lecture 17: Transfer Learning and Domain Adaptation (PDF)

Sin embargo, antes de saltar al transfer learning, comprobaremos **empíricamente** qué rendimiento podemos lograr construyendo una CNN sencilla desde cero.

Esto nos dará: - Una línea base de rendimiento.
- Comprensión de las limitaciones del entrenamiento puro.
- Datos para justificar (o no) el uso de modelos preentrenados.

Próximo paso:
Construiremos y entrenaremos una CNN simple sobre nuestro dataset y analizaremos los resultados antes de decidir si migrar a transferencia.

[10]:
```
from tensorflow.keras import layers, models
```

[12]:
```
img_height = 224
img_width = 224
batch_size = 32
epochs = 20
num_classes = len(class_names)
```

[16]:
```
earlystop_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
```

```
    patience=3,
    restore_best_weights=True
)
```

[20]:
```
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),

    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.BatchNormalization(),

    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.BatchNormalization(),

    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.BatchNormalization(),


    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

[21]:
```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

[22]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling_3 (Rescaling)	(None, 224, 224, 3)	0
conv2d_6 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 111, 111, 32)	0
batch_normalization_4 (BatchNormalization)	(None, 111, 111, 32)	128

```
conv2d_7 (Conv2D)                 (None, 109, 109, 64)          18,496

max_pooling2d_5 (MaxPooling2D)    (None, 54, 54, 64)                 0

batch_normalization_5             (None, 54, 54, 64)               256
(BatchNormalization)

conv2d_8 (Conv2D)                 (None, 52, 52, 128)           73,856

max_pooling2d_6 (MaxPooling2D)    (None, 26, 26, 128)                0

batch_normalization_6             (None, 26, 26, 128)              512
(BatchNormalization)

flatten_1 (Flatten)               (None, 86528)                      0

dense_1 (Dense)                   (None, 128)               11,075,712

dropout (Dropout)                 (None, 128)                        0

dense_2 (Dense)                   (None, 12)                     1,548
```

Total params: 11,171,404 (42.62 MB)

Trainable params: 11,170,956 (42.61 MB)

Non-trainable params: 448 (1.75 KB)

[23]:
```python
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[earlystop_callback]
)
```

```
Epoch 1/20
496/496              403s 803ms/step -
accuracy: 0.2463 - loss: 20.2863 - val_accuracy: 0.2239 - val_loss: 2.4814
Epoch 2/20
496/496              378s 761ms/step -
accuracy: 0.2228 - loss: 3.0947 - val_accuracy: 0.2665 - val_loss: 2.1480
Epoch 3/20
496/496              387s 780ms/step -
accuracy: 0.2435 - loss: 2.3733 - val_accuracy: 0.2506 - val_loss: 4.1784
```

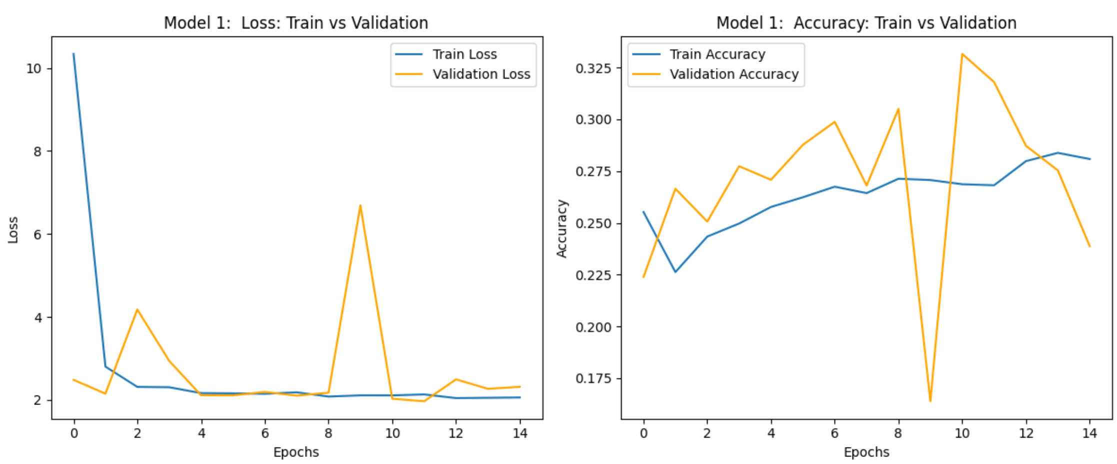
```
Epoch 4/20
496/496              385s 775ms/step -
accuracy: 0.2533 - loss: 2.3134 - val_accuracy: 0.2774 - val_loss: 2.9400
Epoch 5/20
496/496              381s 768ms/step -
accuracy: 0.2584 - loss: 2.1813 - val_accuracy: 0.2708 - val_loss: 2.1100
Epoch 6/20
496/496              385s 777ms/step -
accuracy: 0.2652 - loss: 2.1524 - val_accuracy: 0.2877 - val_loss: 2.1066
Epoch 7/20
496/496              392s 790ms/step -
accuracy: 0.2598 - loss: 2.1828 - val_accuracy: 0.2988 - val_loss: 2.1902
Epoch 8/20
496/496              373s 751ms/step -
accuracy: 0.2639 - loss: 2.2735 - val_accuracy: 0.2680 - val_loss: 2.0997
Epoch 9/20
496/496              335s 676ms/step -
accuracy: 0.2739 - loss: 2.0743 - val_accuracy: 0.3051 - val_loss: 2.1709
Epoch 10/20
496/496              352s 710ms/step -
accuracy: 0.2744 - loss: 2.0924 - val_accuracy: 0.1639 - val_loss: 6.6888
Epoch 11/20
496/496              359s 724ms/step -
accuracy: 0.2676 - loss: 2.1081 - val_accuracy: 0.3316 - val_loss: 2.0265
Epoch 12/20
496/496              355s 715ms/step -
accuracy: 0.2732 - loss: 2.0800 - val_accuracy: 0.3180 - val_loss: 1.9643
Epoch 13/20
496/496              357s 720ms/step -
accuracy: 0.2797 - loss: 2.0451 - val_accuracy: 0.2872 - val_loss: 2.4930
Epoch 14/20
496/496              356s 718ms/step -
accuracy: 0.2888 - loss: 2.0273 - val_accuracy: 0.2753 - val_loss: 2.2652
Epoch 15/20
496/496              373s 751ms/step -
accuracy: 0.2807 - loss: 2.0677 - val_accuracy: 0.2388 - val_loss: 2.3134
```

[5]:
```python
def plot_training_history(history, title_prefix=""):
    """
    Grafica loss y accuracy de entrenamiento y validación.

    Args:
        history: history object (keras.callbacks.History) después de model.
 ↪fit().
        title_prefix: (opcional) texto para añadir al inicio de los títulos de␣
 ↪las gráficas.
    """
```

```python
    plt.figure(figsize=(12, 5))

    # Loss
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
    plt.title(f'{title_prefix} Loss: Train vs Validation')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='orange')
    plt.title(f'{title_prefix} Accuracy: Train vs Validation')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

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

[34]:

```python
plot_training_history(history, title_prefix="Model 1: ")
```



Diagnóstico

- El modelo ya tocó su límite de aprendizaje.
- No hay mejora significativa con más epochs.
- Hay alta varianza en validación; podría ser underfitting, mala arquitectura, o ambos.

Seguir con esta arquitectura, solo si disponemos de más datos

0.1.2 Aumentando dataset

[16]:
```
from vision.config import DATALOADER, ANNOTATIONS
```

[17]:
```
import json
import pandas as pd
import os
```

[18]:
```
json_files = [
    'cis_test_annotations.json',
    'cis_val_annotations.json',
    'trans_test_annotations.json',
    'trans_val_annotations.json',
    'train_annotations.json'
]
```

[19]:
```
from tqdm import tqdm
```

[20]:
```
def load_and_merge_annotations(base_path, json_files):
    merged_dfs = []

    for file in tqdm(json_files, desc="Processing JSON files"):
        file_path = os.path.join(base_path, file)
        with open(file_path, 'r') as f:
            data = json.load(f)

        images_df = pd.DataFrame(data['images'])
        categories_df = pd.DataFrame(data['categories'])
        annotations_df = pd.DataFrame(data['annotations'])

        merged_df = annotations_df.merge(
            images_df, left_on='image_id', right_on='id', suffixes=('_ann',␣
↪'_img')
        )
        merged_df = merged_df.merge(
            categories_df, left_on='category_id', right_on='id', suffixes=('',␣
↪'_cat')
        )

        merged_dfs.append(merged_df)

    final_df = pd.concat(merged_dfs, ignore_index=True)
    return final_df
```

[21]:
```
combined_df = load_and_merge_annotations(ANNOTATIONS, json_files)
```

```
Processing JSON files: 100%|          | 5/5 [00:00<00:00,  7.77it/s]
```

[22]:
```
combined_df.head()
```

[22]:
```
                               image_id  category_id  \
0  5a263af8-23d2-11e8-a6a3-ec086b02610b           30
1  5971faf4-23d2-11e8-a6a3-ec086b02610b           30
2  59ffbd00-23d2-11e8-a6a3-ec086b02610b            1
3  599d7b4a-23d2-11e8-a6a3-ec086b02610b            1
4  599fbfe7-23d2-11e8-a6a3-ec086b02610b           30

                                 id_ann  \
0  343db02b-7d5b-11e7-884d-7845c41c2c67
1  343db02c-7d5b-11e7-884d-7845c41c2c67
2                                 36132
3                                 11320
4  343db02f-7d5b-11e7-884d-7845c41c2c67

                                                bbox  \
0                                                NaN
1                                                NaN
2  [1118.72, 570.88, 328.96000000000004, 180.4800…
3  [513.28, 500.47999999999996, 153.6000000000000…
4                                                NaN

                                   file_name rights_holder  height  width  \
0  5a263af8-23d2-11e8-a6a3-ec086b02610b.jpg   Justin Brown    1494   2048
1  5971faf4-23d2-11e8-a6a3-ec086b02610b.jpg   Justin Brown    1494   2048
2  59ffbd00-23d2-11e8-a6a3-ec086b02610b.jpg   Justin Brown    1494   2048
3  599d7b4a-23d2-11e8-a6a3-ec086b02610b.jpg   Justin Brown    1494   2048
4  599fbfe7-23d2-11e8-a6a3-ec086b02610b.jpg   Justin Brown    1494   2048

   frame_num         date_captured  location  seq_num_frames  \
0          2  2012-01-29 12:35:56        38               3
1          1  2012-01-23 16:56:32        38               3
2          1  2012-02-11 01:48:19        38               3
3          3  2012-02-01 12:24:08        38               3
4          2  2012-02-08 03:21:39        38               3

                                 seq_id                                id_img  \
0  6f026d3d-5567-11e8-960a-dca9047ef277  5a263af8-23d2-11e8-a6a3-ec086b02610b
1  6f025d61-5567-11e8-85d1-dca9047ef277  5971faf4-23d2-11e8-a6a3-ec086b02610b
2  6f0298b8-5567-11e8-8804-dca9047ef277  59ffbd00-23d2-11e8-a6a3-ec086b02610b
3  6f0278ba-5567-11e8-8041-dca9047ef277  599d7b4a-23d2-11e8-a6a3-ec086b02610b
4  6f028e68-5567-11e8-a478-dca9047ef277  599fbfe7-23d2-11e8-a6a3-ec086b02610b
```

```
   id     name
0  30    empty
1  30    empty
2   1  opossum
3   1  opossum
4  30    empty
```

[23]:
```python
from vision.dataset import rescale_bounding_boxes
```

[24]:
```python
rescaled_df = rescale_bounding_boxes(combined_df, target_width=1024,␣
 ↪target_height=747)
rescaled_df[['file_name', 'bbox_scaled']].head()
```

[24]:
```
                                     file_name  \
0  5a263af8-23d2-11e8-a6a3-ec086b02610b.jpg
1  5971faf4-23d2-11e8-a6a3-ec086b02610b.jpg
2  59ffbd00-23d2-11e8-a6a3-ec086b02610b.jpg
3  599d7b4a-23d2-11e8-a6a3-ec086b02610b.jpg
4  599fbfe7-23d2-11e8-a6a3-ec086b02610b.jpg

                                         bbox_scaled
0                                               None
1                                               None
2  [559.36, 285.44, 164.48000000000002, 90.240000…
3  [256.64, 250.23999999999998, 76.80000000000001…
4                                               None
```

[25]:
```python
from vision.plots import show_random_image_with_bbox, show_image_with_multi_bbox
from vision.config import RAW
```

[45]:
```python
show_random_image_with_bbox(rescaled_df, RAW)
```

Category: coyote

[48]:
```
show_image_with_multi_bbox(rescaled_df, RAW)
```

File: 59b7a6f1-23d2-11e8-a6a3-ec086b02610b.jpg

[49]:
```
valid_df = rescaled_df[rescaled_df['bbox_scaled'].notnull()]
class_counts = valid_df['name'].value_counts()
class_counts
```

[49]:
```
name
opossum     11970
raccoon      7076
rabbit       4995
coyote       4700
bobcat       4320
cat          4303
dog          2641
car          2613
squirrel     2479
bird         1579
skunk         722
rodent        482
deer          186
badger         29
fox             8
```

```
Name: count, dtype: int64
```

0.1.3 Análisis de balance de clases

Además de excluir `fox`, `badger` y `deer` por bajo número de ejemplos, será bueno también excluir `rodent` (218) y `skunk` (508), porque siguen siendo muy minoritarios frente a las clases principales.

También se aplicara *data augmentation*

[50]:
```
from vision.dataset import save_recortes_by_class
from vision.config import RAW, CLIPS
```

[51]:
```
save_recortes_by_class(valid_df, RAW, CLIPS)
```

```
Saving crops: 100%|      | 48103/48103 [20:41<00:00, 38.73it/s]
```

[52]:
```
from vision.dataset import count_files_per_class
```

[53]:
```
counts = count_files_per_class(CLIPS)
counts
```

[53]:
```
{'opossum': 11970,
 'raccoon': 7076,
 'rabbit': 4995,
 'coyote': 4700,
 'bobcat': 4320,
 'cat': 4303,
 'dog': 2641,
 'car': 2613,
 'squirrel': 2479,
 'bird': 1579,
 'skunk': 722,
 'rodent': 482,
 'deer': 186,
 'badger': 29,
 'fox': 8}
```

Clases a excluir: fox, badger, deer, rodent, skunk

además…

[54]:
```
from vision.plots import show_random_image_by_class
```

[55]:
```
show_random_image_by_class(valid_df, 'car', RAW)
```

Category: car

En datasets de fauna (como cámaras trampa) es común encontrar categorías "no-animal" como car, human, vegetation, etc.

¿Por qué? Porque esos elementos aparecen accidentalmente en el entorno.

Conviene mantener clases distractoras (como car, human, etc.) porque enseñan al modelo a diferenciar fauna de objetos no relevantes, evitando falsos positivos y mejorando la generalización.

Aplicando transformaciones

[61]:
```
import albumentations as A
```

[62]:
```
augment_dog_car_squirrel = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.Affine(translate_percent=0.05, scale=1.1, rotate=10, p=0.4),
    A.GaussNoise(p=0.3),
])
```

[63]:
```
augment_bird = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    A.Rotate(limit=20, p=0.6),
    A.Affine(translate_percent=0.1, scale=1.2, rotate=15, p=0.5),
    A.GaussNoise(p=0.4),
    A.Blur(blur_limit=3, p=0.2),
])
```

[64]:
```
from vision.dataset import apply_augmentations_for_class
```

[65]:
```
from vision.config import AUGMENTED
```

[66]:
```
apply_augmentations_for_class('dog', CLIPS, AUGMENTED,␣
 ↪augment_dog_car_squirrel, num_augmentations=2)
apply_augmentations_for_class('car', CLIPS, AUGMENTED,␣
 ↪augment_dog_car_squirrel, num_augmentations=2)
apply_augmentations_for_class('squirrel', CLIPS, AUGMENTED,␣
 ↪augment_dog_car_squirrel, num_augmentations=2)
apply_augmentations_for_class('bird', CLIPS, AUGMENTED, augment_bird,␣
 ↪num_augmentations=4)
```

```
Augmenting dog: 100%|      | 2641/2641 [00:41<00:00, 63.89it/s]
Augmenting car: 100%|      | 2613/2613 [02:52<00:00, 15.16it/s]
Augmenting squirrel: 100%|      | 2479/2479 [00:38<00:00, 64.19it/s]
Augmenting bird: 100%|      | 1579/1579 [00:27<00:00, 56.97it/s]
```

Encapsulamos únicamente las clases que tienen una muestra representativa

[67]:
```
allowed_classes = [
    'opossum', 'rabbit', 'coyote', 'cat', 'squirrel', 'raccoon',
    'dog', 'bobcat', 'car', 'bird'
    # NOTA: ahora estamos excluyendo fox, badger, deer, rodent, skunk
]
```

[70]:
```
import shutil
import random
```

[71]:
```
split_ratio = 0.25  # 25% valid, 75% train
```

[74]:
```
for class_name in allowed_classes:
    src_dir = os.path.join(CLIPS, class_name)
    train_dir = os.path.join(DATALOADER, 'train', class_name)
    valid_dir = os.path.join(DATALOADER, 'valid', class_name)

    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(valid_dir, exist_ok=True)
```

```python
    if not os.path.exists(src_dir):
        print(f"La carpeta {src_dir} no existe, saltando...")
        continue

    files = [f for f in os.listdir(src_dir) if os.path.isfile(os.path.
 ↪join(src_dir, f))]
    random.shuffle(files)

    n_valid = int(len(files) * split_ratio)
    valid_files = files[:n_valid]
    train_files = files[n_valid:]

    for f in tqdm(train_files, desc=f"{class_name} - train", leave=True):
        src = os.path.join(src_dir, f)
        dst = os.path.join(train_dir, f)
        shutil.copy2(src, dst)

    for f in tqdm(valid_files, desc=f"{class_name} - valid", leave=True):
        src = os.path.join(src_dir, f)
        dst = os.path.join(valid_dir, f)
        shutil.copy2(src, dst)

print("Distribución train/valid completada.")
```

```
opossum - train: 100%|      | 8978/8978 [02:05<00:00, 71.40it/s]
opossum - valid: 100%|      | 2992/2992 [00:42<00:00, 71.02it/s]
rabbit - train: 100%|      | 3747/3747 [00:50<00:00, 74.62it/s]
rabbit - valid: 100%|      | 1248/1248 [00:16<00:00, 75.02it/s]
coyote - train: 100%|      | 3525/3525 [00:50<00:00, 70.03it/s]
coyote - valid: 100%|      | 1175/1175 [00:17<00:00, 69.01it/s]
cat - train: 100%|      | 3228/3228 [00:44<00:00, 73.15it/s]
cat - valid: 100%|      | 1075/1075 [00:15<00:00, 69.50it/s]
squirrel - train: 100%|      | 1860/1860 [00:23<00:00, 78.91it/s]
squirrel - valid: 100%|      | 619/619 [00:07<00:00, 79.82it/s]
raccoon - train: 100%|      | 5307/5307 [01:09<00:00, 76.61it/s]
raccoon - valid: 100%|      | 1769/1769 [00:22<00:00, 78.29it/s]
dog - train: 100%|      | 1981/1981 [00:25<00:00, 78.38it/s]
dog - valid: 100%|      | 660/660 [00:08<00:00, 79.45it/s]
bobcat - train: 100%|      | 3240/3240 [00:39<00:00, 82.15it/s]
bobcat - valid: 100%|      | 1080/1080 [00:13<00:00, 81.40it/s]
car - train: 100%|      | 1960/1960 [00:30<00:00, 63.53it/s]
car - valid: 100%|      | 653/653 [00:10<00:00, 61.99it/s]
bird - train: 100%|      | 1185/1185 [00:14<00:00, 83.28it/s]
bird - valid: 100%|      | 394/394 [00:04<00:00, 92.12it/s]

Distribución train/valid completada.
```

[77]:
```python
from vision.config import TRAIN, VALID
```

[78]:
```python
counts = count_files_per_class(TRAIN)
counts
```

[78]:
```
{'opossum': 8978,
 'raccoon': 5307,
 'rabbit': 3747,
 'coyote': 3525,
 'bobcat': 3240,
 'cat': 3228,
 'dog': 1981,
 'car': 1960,
 'squirrel': 1860,
 'bird': 1185}
```

[79]:
```python
counts = count_files_per_class(VALID)
counts
```

[79]:
```
{'opossum': 2992,
 'raccoon': 1769,
 'rabbit': 1248,
 'coyote': 1175,
 'bobcat': 1080,
 'cat': 1075,
 'dog': 660,
 'car': 653,
 'squirrel': 619,
 'bird': 394}
```

[ ]:
```python
from vision.config import AUGMENTED
```

[82]:
```python
counts = count_files_per_class(AUGMENTED)
counts
```

[82]:
```
{'bird': 6316, 'dog': 5282, 'car': 5226, 'squirrel': 4958}
```

[80]:
```python
for cls in allowed_classes:
    augmented_dir = os.path.join(AUGMENTED, cls)
    train_dir = os.path.join(TRAIN, cls)
    os.makedirs(train_dir, exist_ok=True)

    if os.path.exists(augmented_dir):
        for fname in tqdm(os.listdir(augmented_dir), desc=f"{cls} - augmented",␣
↪leave=True):
            src = os.path.join(augmented_dir, fname)
            dst = os.path.join(train_dir, fname)
            shutil.copy2(src, dst)
```

```
squirrel - augmented: 100%|          | 4958/4958 [00:43<00:00, 113.12it/s]
dog - augmented: 100%|          | 5282/5282 [00:45<00:00, 116.22it/s]
car - augmented: 100%|          | 5226/5226 [01:20<00:00, 65.09it/s]
bird - augmented: 100%|          | 6316/6316 [00:52<00:00, 119.78it/s]
```

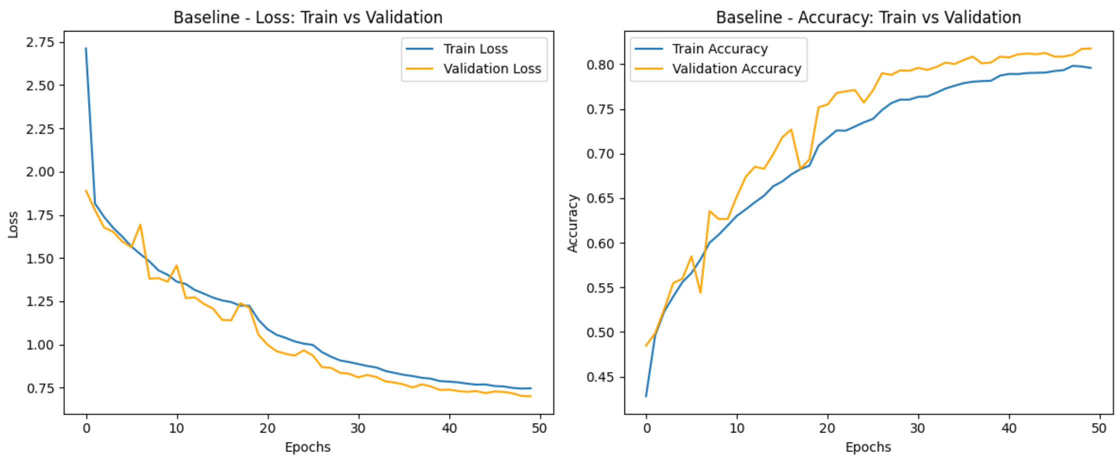
[81]:
```python
counts = count_files_per_class(TRAIN)
counts
```

[81]:
```
{'opossum': 8978,
 'bird': 7501,
 'dog': 7263,
 'car': 7186,
 'squirrel': 6818,
 'raccoon': 5307,
 'rabbit': 3747,
 'coyote': 3525,
 'bobcat': 3240,
 'cat': 3228}
```

[ ]:
```python
from vision.config import CHECKPOINTS_DIR
from vision.plots import plot_training_history_from_dict
```

[2]:
```python
import json
with open(CHECKPOINTS_DIR / "baseline_model_history.json", "r") as f:
    history_data = json.load(f)
```

[3]:
```python
plot_training_history_from_dict(history_data, title_prefix="Baseline - ")
```



modelo baseline

- **Loss:** Disminuye de forma estable en entrenamiento y validación, sin señales de sobreajuste.

- **Accuracy:** La precisión en validación supera a la de entrenamiento desde la época 10, lo que sugiere buena generalización.
- **Regularización:** El uso de Dropout y L2 fue efectivo para controlar el sobreentrenamiento.
- **Callbacks:** `EarlyStopping` y `ReduceLROnPlateau` funcionaron correctamente, estabilizando el entrenamiento.
- **Arquitectura:** La red CNN secuencial es adecuada como línea base para clasificación multiclase de fauna.

Conclusión: El modelo presentó un desempeño estable y generalizable, siendo un punto de partida sólido para futuras mejoras.

0.1.4 Desventajas del modelo actual

- Parte de cero → aprende bordes, texturas y patrones sin conocimiento previo.
- Capacidad limitada → no captura la complejidad de fauna salvaje (variaciones de especies, fondos, luz).
- Requiere muchos más datos y epochs para alcanzar niveles competitivos.

Recomendación - Cambiar a un modelo preentrenado (EfficientNetB0 o ResNet50) con fine-tuning. - Incorporar class_weights para compensar el desbalance entre clases. - Considerar optimizaciones como learning rate scheduler.

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: