LABORATORIO 1

Semilleros
by DSRP

Ejercicios

Para comenzar, ejecute la celda de código a continuación para configurar todo.

```
# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.computer_vision.ex1 import *

# Imports
import os, warnings
import matplotlib.pyplot as plt
from matplotlib import gridspec

import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Reproducability
def set_seed(seed=31415):
    np.random.seed(seed)
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
set_seed()

# Set Matplotlib defaults
plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('image', cmap='magma')
warnings.filterwarnings("ignore") # to clean up output cells
```

```
# Load training and validation sets
ds_train_ = image_dataset_from_directory(
    '../input/car-or-truck/train',
    labels='inferred',
    label_mode='binary',
    image_size=[128, 128],
    interpolation='nearest',
    batch_size=64,
    shuffle=True,
)
ds_valid_ = image_dataset_from_directory(
    '../input/car-or-truck/valid',
    labels='inferred',
    label_mode='binary',
    image_size=[128, 128],
    interpolation='nearest',
    batch_size=64,
    shuffle=False,
)
```

```
# Data Pipeline
def convert_to_float(image, label):
    image = tf.image.convert_image_dtype(image, dtype=tf.float32)
    return image, label

AUTOTUNE = tf.data.experimental.AUTOTUNE
ds_train = (
    ds_train_
    .map(convert_to_float)
    .cache()
    .prefetch(buffer_size=AUTOTUNE)
)
ds_valid = (
    ds_valid_
    .map(convert_to_float)
    .cache()
    .prefetch(buffer_size=AUTOTUNE)
)
```

Semilleros by DSRP

El modelo InceptionV1 previamente entrenado en ImageNet está disponible en el repositorio de TensorFlow Hub, pero lo cargaremos desde una copia local.

Ejecute esta celda para cargar InceptionV1 para su base.

```
import tensorflow_hub as hub

pretrained_base = tf.keras.models.load_model(
    '../input/cv-course-models/cv-course-models/inceptionv1'
)
```

Ejercicios

Paso 1: Define Pretrained Base

Ahora que tiene una base previamente entrenada para realizar nuestra extracción de características, decida si esta base debe ser entrenable o no.

[12]:

```
# YOUR_CODE_HERE
pretrained_base.trainable = False

# Check your answer
q_1.check()
```

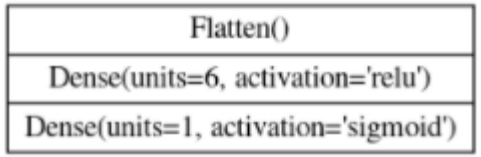
Correct: When doing transfer learning, it's generally not a good idea to retrain the entire base -- at least not without some care. The reason is that the random weights in the head will initially create large gradient updates, which propogate back into the base layers and destroy much of the pretraining. Using techniques known as **fine tuning** it's possible to further train the base on new data, but this requires some care to do well.

Semilleros by DSRP

Ejercicios

Paso 2: Attach Head

Ahora que la base está definida para realizar la extracción de características, cree un encabezado de Capas **Dense** (densas) para realizar la clasificación, siguiendo este diagrama:




```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    pretrained_base,
    layers.Flatten(),
    # YOUR CODE HERE. Attach a head of dense layers.
    layers.Dense(6, activation='relu'),
    layers.Dense(1, activation='sigmoid'),
])

# Check your answer
q_2.check()
```

Correct

Semilleros by DSRP

Ejercicios

Paso 3: Train

Antes de entrenar un modelo en Keras, debe especificar un optimizador para realizar el descenso de gradiente, una función de pérdida para minimizar y (opcionalmente) cualquier métrica de rendimiento. El algoritmo de optimización que usaremos para este curso se llama "**Adam**", que generalmente funciona bien independientemente del tipo de problema que esté tratando de resolver.

Sin embargo, la pérdida y las métricas deben coincidir con el tipo de problema que está tratando de resolver. Nuestro problema es un problema de clasificación binaria: el automóvil se codifica como 0 y el camión se codifica como 1. Elija una pérdida adecuada y una métrica de precisión adecuada para la clasificación binaria.

```
# YOUR CODE HERE: what loss function should you use for a binary
# classification problem? (Your answer for each should be a string.)
optimizer = tf.keras.optimizers.Adam(epsilon=0.01)
model.compile(
    optimizer=optimizer,
    loss = 'binary_crossentropy',
    metrics=['binary_accuracy'],
)

# Check your answer
q_3.check()
```

Correct

Semilleros by DSRP

Ejercicios

Paso 3: Train

```
history = model.fit(
    ds_train,
    validation_data=ds_valid,
    epochs=30,
)
```

```
Epoch 1/30
80/80 [==============================] - 107s 1s/step - loss: 0.6996 - binary_accuracy: 0.5382 - val_loss: 0.6895 - val_binary_accuracy: 0.5963
Epoch 2/30
80/80 [==============================] - 101s 1s/step - loss: 0.6834 - binary_accuracy: 0.6324 - val_loss: 0.6675 - val_binary_accuracy: 0.7345
Epoch 3/30
80/80 [==============================] - 101s 1s/step - loss: 0.6423 - binary_accuracy: 0.7342 - val_loss: 0.6093 - val_binary_accuracy: 0.7830
Epoch 4/30
80/80 [==============================] - 101s 1s/step - loss: 0.5882 - binary_accuracy: 0.7794 - val_loss: 0.5632 - val_binary_accuracy: 0.8056
Epoch 5/30
80/80 [==============================] - 101s 1s/step - loss: 0.5514 - binary_accuracy: 0.8024 - val_loss: 0.5337 - val_binary_accuracy: 0.8210
Epoch 6/30
80/80 [==============================] - 101s 1s/step - loss: 0.5260 - binary_accuracy: 0.8169 - val_loss: 0.5121 - val_binary_accuracy: 0.8295
Epoch 7/30
80/80 [==============================] - 101s 1s/step - loss: 0.5058 - binary_accuracy: 0.8243 - val_loss: 0.4949 - val_binary_accuracy: 0.8347
Epoch 8/30
80/80 [==============================] - 100s 1s/step - loss: 0.4889 - binary_accuracy: 0.8312 - val_loss: 0.4808 - val_binary_accuracy: 0.8390
Epoch 9/30
80/80 [==============================] - 101s 1s/step - loss: 0.4741 - binary_accuracy: 0.8364 - val_loss: 0.4685 - val_binary_accuracy: 0.8392
Epoch 10/30
80/80 [==============================] - 101s 1s/step - loss: 0.4613 - binary_accuracy: 0.8397 - val_loss: 0.4577 - val_binary_accuracy: 0.8414
Epoch 11/30
80/80 [==============================] - 101s 1s/step - loss: 0.4496 - binary_accuracy: 0.8454 - val_loss: 0.4481 - val_binary_accuracy: 0.8440
Epoch 12/30
80/80 [==============================] - 101s 1s/step - loss: 0.4392 - binary_accuracy: 0.8491 - val_loss: 0.4394 - val_binary_accuracy: 0.8464
Epoch 13/30
80/80 [==============================] - 101s 1s/step - loss: 0.4295 - binary_accuracy: 0.8525 - val_loss: 0.4317 - val_binary_accuracy: 0.8466
```

Semilleros by DSRP

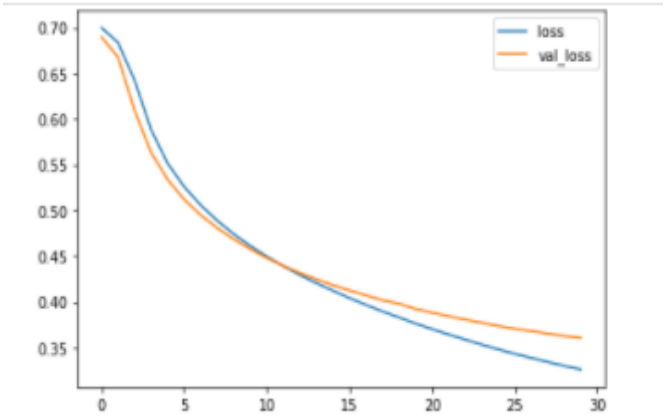
Ejercicios

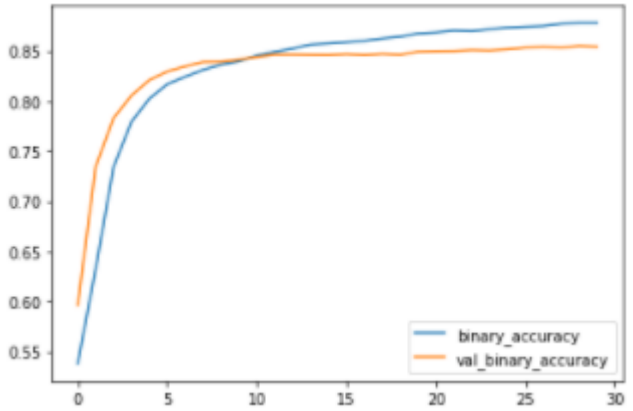
Paso 4: Examine Loss and Accuracy

Ejecute la celda a continuación para trazar las curvas métricas y de pérdida para esta ejecución de entrenamiento.

```
import pandas as pd
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
history_frame.loc[:, ['binary_accuracy', 'val_binary_accuracy']].plot();
```

¿Notas alguna diferencia entre estas curvas de aprendizaje y las curvas para VGG16 del tutorial? ¿Qué le dice esta diferencia sobre lo que aprendió este modelo (InceptionV2) en comparación con VGG16? ¿Hay formas en las que uno es mejor que el otro? ¿Peor?





Nota que la pérdida de entrenamiento y la pérdida de validación permanezcan bastante cerca, es evidencia de que el modelo no solo memoriza los datos de entrenamiento, sino que aprende las propiedades generales de las dos clases. Pero, debido a que este modelo converge con una pérdida mayor que el modelo VGG16, es probable que no se ajuste a algunos y podría beneficiarse de una capacidad adicional.

Semilleros by DSRP

LABORATORIO 2

Semilleros
by DSRP

Ejercicios

Para comenzar, ejecute la celda de código a continuación para comenzar.

```
# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.computer_vision.ex2 import *

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('image', cmap='magma')

tf.config.run_functions_eagerly(True)
```

Semilleros by DSRP

Apply transformations

Ejecute la siguiente celda para cargar una imagen que usaremos para los próximos ejercicios

```
image_path = '../input/computer-vision-resources/car_illus.jpg'
image = tf.io.read_file(image_path)
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[400, 400])

img = tf.squeeze(image).numpy()
plt.figure(figsize=(6, 6))
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show();
```

Semilleros by DSRP

Apply transformations

Puede ejecutar esta celda para ver algunos kernels estándar utilizados en el procesamiento de imágenes.

```
import learntools.computer_vision.visiontools as visiontools
from learntools.computer_vision.visiontools import edge, bottom_sobel, emboss, sharpen

kernels = [edge, bottom_sobel, emboss, sharpen]
names = ["Edge Detect", "Bottom Sobel", "Emboss", "Sharpen"]

plt.figure(figsize=(12, 12))
for i, (kernel, name) in enumerate(zip(kernels, names)):
    plt.subplot(1, 4, i+1)
    visiontools.show_kernel(kernel)
    plt.title(name)
plt.tight_layout()
```

Semilleros by DSRP

Ejercicios

Paso 1: Define kernel

Utilice la siguiente celda de código para definir un kernel. Puede elegir qué tipo de kernel aplicar. Una cosa a tener en cuenta es que la suma de los números en el kernel determina qué tan brillante es la imagen final. Por lo general, debe intentar mantener la suma de los números entre 0 y 1 (aunque no es necesario para una respuesta correcta).En general, un kernel puede tener cualquier número de filas y columnas.

Para este ejercicio, usemos un kernel de 3 × 3, que a menudo da los mejores resultados. Defina un kernel con tf.constant.

```
# YOUR CODE HERE: Define a kernel with 3 rows and 3 columns.
kernel = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])
# Uncomment to view kernel
# visiontools.show_kernel(kernel)

# Check your answer
q_1.check()
```

Correct

Ahora haremos el primer paso de feature extraction, el paso de filtrado. Primero ejecute esta celda para reformatear un poco para TensorFlow.

```
# Reformat for batch compatibility.
image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)
```

Semilleros by DSRP

Ejercicios

Paso 2: Apply Convolution

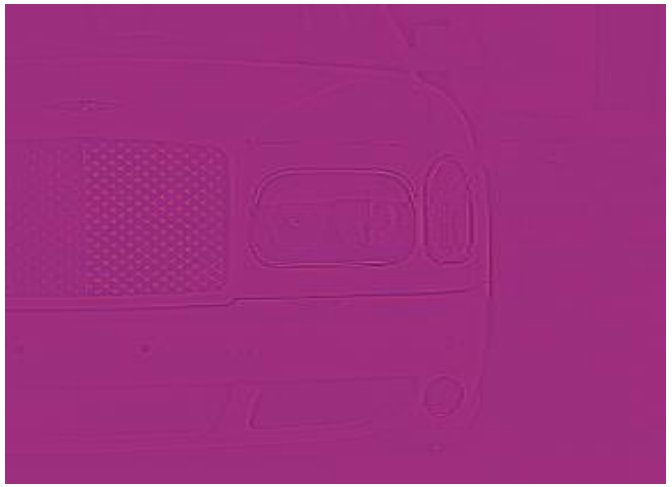
Ahora aplicaremos el kernel a la imagen mediante una convolución. La capa en Keras que hace esto son **layers.Conv2D**. ¿Cuál es la función de backend en TensorFlow que realiza la misma operación?

[62]:
```
# YOUR CODE HERE: Give the TensorFlow convolution function (without arguments)

conv_fn = tf.nn.conv2d

# Check your answer
q_2.check()
```

Correct

Una vez que tenga la respuesta correcta, ejecute la siguiente celda para ejecutar la convolución y ver el resultado.

```
image_filter = conv_fn(
    input=image,
    filters=kernel,
    strides=1, # or (1, 1)
    padding='SAME',
)

plt.imshow(
    # Reformat for plotting
    tf.squeeze(image_filter)
)
plt.axis('off')
plt.show();
```



Semilleros by DSRP

Ejercicios

Paso 3: Apply ReLU

Ahora detecte la característica con la función ReLU. En Keras, normalmente usará esto como la función de activación en una capa **Conv2D**. ¿Cuál es la función de backend en TensorFlow que hace lo mismo?

[110]:
```
# YOUR CODE HERE: Give the TensorFlow ReLU function (without arguments)
relu_fn = tf.nn.relu

# Check your answer
q_3.check()
```

Correct

Una vez que tenga la solución, ejecute esta celda para detectar la función con ReLU y vea el resultado.

La imagen que ve a continuación es el **feature map** producido por el kernel que eligió. Si lo desea, experimente con algunos de los otros **kernels** sugeridos anteriormente, o intente inventar uno que extraiga cierto tipo de característica.

```
image_detect = relu_fn(image_filter)

plt.imshow(
    # Reformat for plotting
    tf.squeeze(image_detect)
)
plt.axis('off')
plt.show();
```

Semilleros by DSRP

Ejercicios

En el tutorial, nuestra discusión sobre **kernels** y **feature maps** fue principalmente visual. Vimos el efecto de Conv2D y ReLU al observar cómo transformaron algunas imágenes de ejemplo.

Pero las operaciones en una red convolucional (como en todas las redes neuronales) suelen definirse mediante funciones matemáticas, mediante un cálculo de números. En el próximo ejercicio, nos tomaremos un momento para explorar este punto de vista.

Comencemos por definir una matriz simple para que actúe como imagen y otra matriz para que actúe como núcleo. Ejecute la siguiente celda para ver estas matrices.

```python
# Sympy is a python library for symbolic mathematics. It has a nice
# pretty printer for matrices, which is all we'll use it for.
import sympy
sympy.init_printing()
from IPython.display import display

image = np.array([
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 1, 1, 1],
    [0, 1, 0, 0, 0, 0],
])

kernel = np.array([
    [1, -1],
    [1, -1],
])

display(sympy.Matrix(image))
display(sympy.Matrix(kernel))
# Reformat for Tensorflow
image = tf.cast(image, dtype=tf.float32)
image = tf.reshape(image, [1, *image.shape, 1])
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

Semilleros by DSRP

Ejercicios

Paso 4: Observe Convolution on a Numerical Matrix

¿Que ves? La imagen es simplemente una línea vertical larga a la izquierda y una línea horizontal corta en la parte inferior derecha. ¿Y el kernel? ¿Qué efecto crees que tendrá en esta imagen? Una vez que lo haya pensado, ejecute la siguiente celda para obtener la respuesta.

```
# View the solution (Run this code cell to receive credit!)
q_4.check()
```

Correct:

En el tutorial, hablamos sobre cómo el patrón de números positivos le dirá el tipo de características que extraerá el kernel. Este núcleo tiene una columna vertical de unos, por lo que esperaríamos que devolviera características de líneas verticales.

Ahora probémoslo. Ejecute la siguiente celda para aplicar convolución y ReLU a la imagen y mostrar el resultado.

```
image_filter = tf.nn.conv2d(
    input=image,
    filters=kernel,
    strides=1,
    padding='VALID',
)
image_detect = tf.nn.relu(image_filter)

# The first matrix is the image after convolution, and the second is
# the image after ReLU.
display(sympy.Matrix(tf.squeeze(image_filter).numpy()))
display(sympy.Matrix(tf.squeeze(image_detect).numpy()))
```

$$\begin{bmatrix} -2.0 & 2.0 & 0 & 0 & 0 \\ -2.0 & 2.0 & 0 & 0 & 0 \\ -2.0 & 2.0 & 0 & 0 & 0 \\ -2.0 & 2.0 & -1.0 & 0 & 0 \\ -2.0 & 2.0 & -1.0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2.0 & 0 & 0 & 0 \\ 0 & 2.0 & 0 & 0 & 0 \\ 0 & 2.0 & 0 & 0 & 0 \\ 0 & 2.0 & 0 & 0 & 0 \\ 0 & 2.0 & 0 & 0 & 0 \end{bmatrix}$$

Semilleros by DSRP