This notebook is an exercise in the Computer Vision course. You can reference the tutorial at this link.

Introduction

In these exercises, you'll build a custom convnet with performance competitive to the VGG16 model from Lesson 1.

Get started by running the code cell below.

In [1]:

```
# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.computer_vision.ex5 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

# Reproducibility
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',
```

```python
    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)
)
```

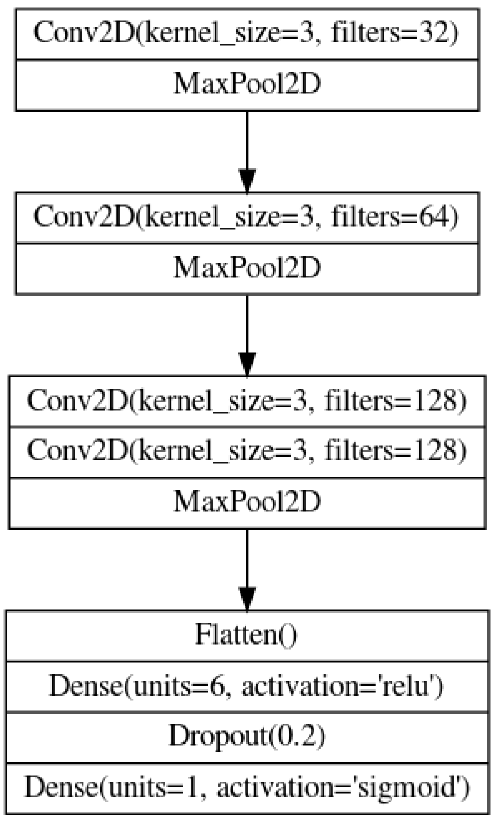
```
Found 5117 files belonging to 2 classes.
2022-12-19 13:31:10.593959: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:10.781235: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:10.782350: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:10.791951: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-19 13:31:10.792375: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:10.793593: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:10.794803: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:13.235380: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:13.236194: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:13.236846: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-19 13:31:13.237448: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Cre
```

```
ated device /job:localhost/replica:0/task:0/device:GPU:0 with 15401 MB memory:  -> devic
e: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
Found 5051 files belonging to 2 classes.
```

Design a Convnet

Let's design a convolutional network with a block architecture like we saw in the tutorial. The model from the example had three blocks, each with a single convolutional layer. Its performance on the "Car or Truck" problem was okay, but far from what the pretrained VGG16 could achieve. It might be that our simple network lacks the ability to extract sufficiently complex features. We could try improving the model either by adding more blocks or by adding convolutions to the blocks we have.

Let's go with the second approach. We'll keep the three block structure, but increase the number of `Conv2D` layer in the second block to two, and in the third block to three.



1) Define Model

Given the diagram above, complete the model by defining the layers of the third block.

In [2]:
```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    # Block One
```

```
    layers.Conv2D(filters=32, kernel_size=3, activation='relu', padding='same',
                  input_shape=[128, 128, 3]),
    layers.MaxPool2D(),

    # Block Two
    layers.Conv2D(filters=64, kernel_size=3, activation='relu', padding='same'),
    layers.MaxPool2D(),

    layers.Conv2D(kernel_size=3, filters=128, activation='relu', padding='same'),
    layers.Conv2D(kernel_size=3, filters=128, activation='relu', padding='same'),
    layers.MaxPool2D(),

    # Head
    layers.Flatten(),
    layers.Dense(6, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid'),
])

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

Correct

In [3]:

```
# Lines below will give you a hint or solution code
#q_1.hint()
#q_1.solution()
```

2) Compile

To prepare for training, compile the model with an appropriate loss and accuracy metric for the "Car or Truck" dataset.

In [4]:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
    loss='binary_crossentropy',
    metrics=['binary_accuracy']
)

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

Correct

In [5]:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
    loss='binary_crossentropy',
    metrics=['binary_accuracy'],
)
q_2.assert_check_passed()
```

Correct

In [6]:

```
# Lines below will give you a hint or solution code
#q_2.hint()
#q_2.solution()
```

Finally, let's test the performance of this new model. First run this cell to fit the model to the training set.

In [7]:

```
history = model.fit(
    ds_train,
    validation_data=ds_valid,
    epochs=50,
)
```

```
2022-12-19 13:31:16.284541: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:1
85] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/50
2022-12-19 13:31:19.025861: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cu
DNN version 8005
80/80 [==============================] - 30s 270ms/step - loss: 0.6816 - binary_accurac
y: 0.5744 - val_loss: 0.6691 - val_binary_accuracy: 0.5785
Epoch 2/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6667 - binary_accuracy:
0.5787 - val_loss: 0.6610 - val_binary_accuracy: 0.5785
Epoch 3/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6621 - binary_accuracy:
0.5787 - val_loss: 0.6507 - val_binary_accuracy: 0.5785
Epoch 4/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6534 - binary_accuracy:
0.5787 - val_loss: 0.6450 - val_binary_accuracy: 0.5785
Epoch 5/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6493 - binary_accuracy:
0.6002 - val_loss: 0.6381 - val_binary_accuracy: 0.5991
Epoch 6/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6390 - binary_accuracy:
0.6316 - val_loss: 0.6346 - val_binary_accuracy: 0.5955
Epoch 7/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6371 - binary_accuracy:
0.6344 - val_loss: 0.6367 - val_binary_accuracy: 0.6023
Epoch 8/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6276 - binary_accuracy:
0.6547 - val_loss: 0.6194 - val_binary_accuracy: 0.6452
Epoch 9/50
80/80 [==============================] - 3s 43ms/step - loss: 0.6236 - binary_accuracy:
0.6598 - val_loss: 0.6175 - val_binary_accuracy: 0.6440
Epoch 10/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6169 - binary_accuracy:
0.6691 - val_loss: 0.6091 - val_binary_accuracy: 0.6638
Epoch 11/50
80/80 [==============================] - 3s 42ms/step - loss: 0.6064 - binary_accuracy:
0.6826 - val_loss: 0.6039 - val_binary_accuracy: 0.6710
Epoch 12/50
80/80 [==============================] - 4s 45ms/step - loss: 0.6024 - binary_accuracy:
0.6873 - val_loss: 0.6019 - val_binary_accuracy: 0.6698
Epoch 13/50
```

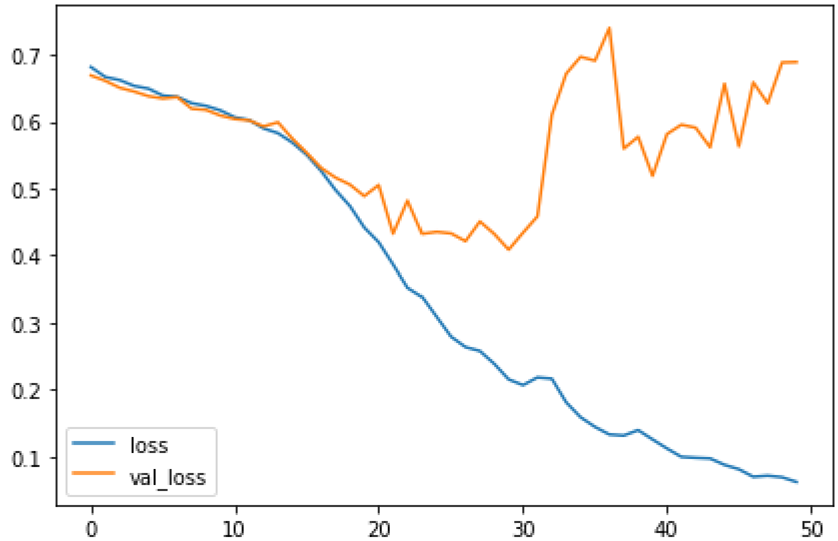
```
80/80 [==============================] - 3s 42ms/step - loss: 0.5899 - binary_accuracy:
0.6959 - val_loss: 0.5929 - val_binary_accuracy: 0.6840
Epoch 14/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5828 - binary_accuracy:
0.7059 - val_loss: 0.5990 - val_binary_accuracy: 0.6789
Epoch 15/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5687 - binary_accuracy:
0.7286 - val_loss: 0.5747 - val_binary_accuracy: 0.7113
Epoch 16/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5503 - binary_accuracy:
0.7370 - val_loss: 0.5535 - val_binary_accuracy: 0.7161
Epoch 17/50
80/80 [==============================] - 3s 42ms/step - loss: 0.5263 - binary_accuracy:
0.7452 - val_loss: 0.5308 - val_binary_accuracy: 0.7434
Epoch 18/50
80/80 [==============================] - 3s 42ms/step - loss: 0.4979 - binary_accuracy:
0.7608 - val_loss: 0.5163 - val_binary_accuracy: 0.7408
Epoch 19/50
80/80 [==============================] - 3s 43ms/step - loss: 0.4739 - binary_accuracy:
0.7813 - val_loss: 0.5060 - val_binary_accuracy: 0.7531
Epoch 20/50
80/80 [==============================] - 3s 42ms/step - loss: 0.4418 - binary_accuracy:
0.8036 - val_loss: 0.4887 - val_binary_accuracy: 0.7753
Epoch 21/50
80/80 [==============================] - 3s 42ms/step - loss: 0.4199 - binary_accuracy:
0.8163 - val_loss: 0.5053 - val_binary_accuracy: 0.7410
Epoch 22/50
80/80 [==============================] - 3s 42ms/step - loss: 0.3867 - binary_accuracy:
0.8319 - val_loss: 0.4326 - val_binary_accuracy: 0.8052
Epoch 23/50
80/80 [==============================] - 3s 43ms/step - loss: 0.3524 - binary_accuracy:
0.8507 - val_loss: 0.4820 - val_binary_accuracy: 0.7612
Epoch 24/50
80/80 [==============================] - 3s 42ms/step - loss: 0.3386 - binary_accuracy:
0.8554 - val_loss: 0.4324 - val_binary_accuracy: 0.7957
Epoch 25/50
80/80 [==============================] - 3s 42ms/step - loss: 0.3089 - binary_accuracy:
0.8697 - val_loss: 0.4353 - val_binary_accuracy: 0.7886
Epoch 26/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2793 - binary_accuracy:
0.8865 - val_loss: 0.4330 - val_binary_accuracy: 0.8101
Epoch 27/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2638 - binary_accuracy:
0.8988 - val_loss: 0.4213 - val_binary_accuracy: 0.8236
Epoch 28/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2580 - binary_accuracy:
0.8943 - val_loss: 0.4508 - val_binary_accuracy: 0.8240
Epoch 29/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2387 - binary_accuracy:
0.9021 - val_loss: 0.4321 - val_binary_accuracy: 0.8157
Epoch 30/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2155 - binary_accuracy:
0.9162 - val_loss: 0.4086 - val_binary_accuracy: 0.8379
Epoch 31/50
80/80 [==============================] - 3s 42ms/step - loss: 0.2069 - binary_accuracy:
0.9214 - val_loss: 0.4341 - val_binary_accuracy: 0.8266
Epoch 32/50
80/80 [==============================] - 3s 43ms/step - loss: 0.2185 - binary_accuracy:
0.9111 - val_loss: 0.4584 - val_binary_accuracy: 0.8293
Epoch 33/50
```

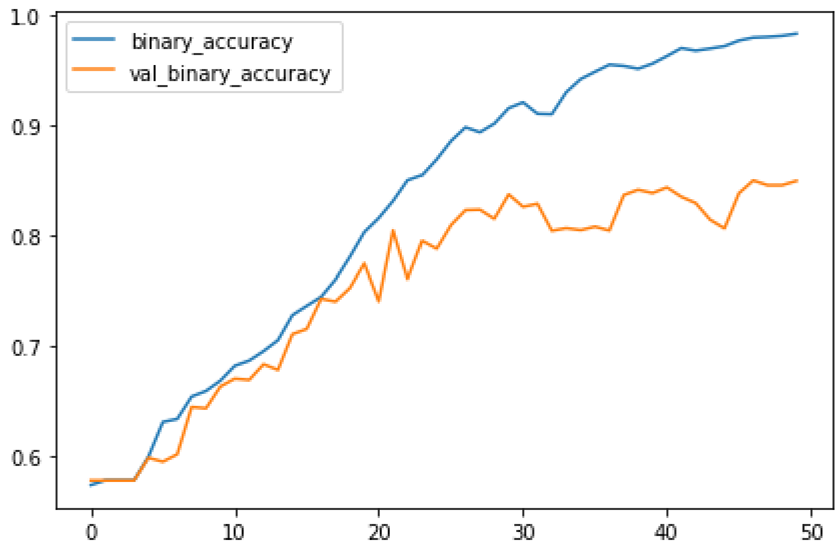
```
80/80 [==============================] - 3s 43ms/step - loss: 0.2165 - binary_accuracy: 0.9107 - val_loss: 0.6101 - val_binary_accuracy: 0.8048
Epoch 34/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1811 - binary_accuracy: 0.9306 - val_loss: 0.6718 - val_binary_accuracy: 0.8072
Epoch 35/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1589 - binary_accuracy: 0.9425 - val_loss: 0.6969 - val_binary_accuracy: 0.8054
Epoch 36/50
80/80 [==============================] - 3s 43ms/step - loss: 0.1445 - binary_accuracy: 0.9492 - val_loss: 0.6911 - val_binary_accuracy: 0.8086
Epoch 37/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1331 - binary_accuracy: 0.9556 - val_loss: 0.7400 - val_binary_accuracy: 0.8050
Epoch 38/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1316 - binary_accuracy: 0.9545 - val_loss: 0.5597 - val_binary_accuracy: 0.8373
Epoch 39/50
80/80 [==============================] - 3s 43ms/step - loss: 0.1397 - binary_accuracy: 0.9519 - val_loss: 0.5774 - val_binary_accuracy: 0.8420
Epoch 40/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1258 - binary_accuracy: 0.9566 - val_loss: 0.5190 - val_binary_accuracy: 0.8390
Epoch 41/50
80/80 [==============================] - 3s 42ms/step - loss: 0.1123 - binary_accuracy: 0.9635 - val_loss: 0.5814 - val_binary_accuracy: 0.8442
Epoch 42/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0997 - binary_accuracy: 0.9707 - val_loss: 0.5954 - val_binary_accuracy: 0.8357
Epoch 43/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0983 - binary_accuracy: 0.9683 - val_loss: 0.5909 - val_binary_accuracy: 0.8299
Epoch 44/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0973 - binary_accuracy: 0.9703 - val_loss: 0.5616 - val_binary_accuracy: 0.8149
Epoch 45/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0878 - binary_accuracy: 0.9724 - val_loss: 0.6567 - val_binary_accuracy: 0.8070
Epoch 46/50
80/80 [==============================] - 3s 43ms/step - loss: 0.0812 - binary_accuracy: 0.9775 - val_loss: 0.5631 - val_binary_accuracy: 0.8388
Epoch 47/50
80/80 [==============================] - 3s 43ms/step - loss: 0.0699 - binary_accuracy: 0.9803 - val_loss: 0.6587 - val_binary_accuracy: 0.8505
Epoch 48/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0716 - binary_accuracy: 0.9808 - val_loss: 0.6274 - val_binary_accuracy: 0.8462
Epoch 49/50
80/80 [==============================] - 3s 43ms/step - loss: 0.0693 - binary_accuracy: 0.9818 - val_loss: 0.6884 - val_binary_accuracy: 0.8462
Epoch 50/50
80/80 [==============================] - 3s 42ms/step - loss: 0.0622 - binary_accuracy: 0.9838 - val_loss: 0.6890 - val_binary_accuracy: 0.8501
```

And now run the cell below to plot the loss and metric curves for this training run.

In [8]:

```python
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();
```





3) Train the Model

How would you interpret these training curves? Did this model improve upon the model from the tutorial?

In [9]:

```
# View the solution (Run this code cell to receive credit!)
q_3.check()
```

Correct:

The learning curves for the model from the tutorial diverged fairly rapidly. This would indicate that it was prone to overfitting and in need of some regularization. The additional layer in our new model would make it even more prone to overfitting. However, adding some regularization with the `Dropout` layer helped prevent this. These changes improved the validation accuracy of the model by several points.

Conclusion

These exercises showed you how to design a custom convolutional network to solve a specific classification problem. Though most models these days will be built on top of a pretrained base, it certain circumstances a smaller custom convnet might still be preferable -- such as with a smaller or unusual dataset or when computing resources are very limited. As you saw here, for certain problems they can perform just as well as a pretrained model.

Keep Going

Continue on to **Lesson 6**, where you'll learn a widely-used technique that can give a boost to your training data: **data augmentation**.

Have questions or comments? Visit the course discussion forum to chat with other learners.