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

> **Accelerate Training with a Kaggle GPU!**
> Did you know Kaggle offers free time with a GPU accelerator? You can speed up training neural networks in this course by switching to **GPU** in the *Accelerator* option on the right. (It may already be turned on.) Two things to be aware of:
>
> - Changing the *Accelerator* option will cause the notebook session to restart. You'll need to rerun any setup code.
> - You can have only one GPU session at a time, so be sure to shut the notebook down after you've finished the exercise.

Introduction

In the tutorial, we saw how to build an image classifier by attaching a head of dense layers to a pretrained base. The base we used was from a model called **VGG16**. We saw that the VGG16 architecture was prone to overfitting this dataset. Over this course, you'll learn a number of ways you can improve upon this initial attempt.

The first way you'll see is to use a base more appropriate to the dataset. The base this model comes from is called **InceptionV1** (also known as GoogLeNet). InceptionV1 was one of the early winners of the ImageNet competition. One of its successors, InceptionV4, is among the state of the art today.

To get started, run the code cell below to set everything up.

In [1]:
```
# 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)
)
```

```
Found 5117 files belonging to 2 classes.
```

```
2022-12-18 05:57:12.229625: 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
2022-12-18 05:57:12.358152: 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
2022-12-18 05:57:12.358894: 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
```

```
2022-12-18 05:57:12.371869: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Te
nsorFlow 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 fla
gs.
2022-12-18 05:57:12.372173: 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
2022-12-18 05:57:12.372920: 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
2022-12-18 05:57:12.373697: 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
2022-12-18 05:57:14.379394: 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
2022-12-18 05:57:14.380306: 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
2022-12-18 05:57:14.381026: 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
2022-12-18 05:57:14.381623: 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.
```

The **InceptionV1** model pretrained on ImageNet is available in the TensorFlow Hub repository, but we'll load it from a local copy. Run this cell to load InceptionV1 for your base.

In [2]:

```
import tensorflow_hub as hub

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

1) Define Pretrained Base

Now that you have a pretrained base to do our feature extraction, decide whether this base should be trainable or not.

In [3]:

```
# YOUR_CODE_HERE
pretrained_base.trainable = False

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

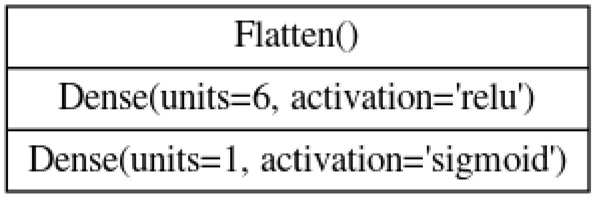
Correct: When doing transfer learning, it's generally not a good idea to retrain the entire base -- at least not without some care. The reason is that the random weights in the head will initially create large gradient updates, which propogate back into the base layers and destroy much of the pretraining. 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.

In [4]:

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

2) Attach Head

Now that the base is defined to do the feature extraction, create a head of `Dense` layers to perform the classification, following this diagram:



Flatten()

Dense(units=6, activation='relu')

Dense(units=1, activation='sigmoid')

In [5]:

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

model = keras.Sequential([
    pretrained_base,
    layers.Flatten(),
    layers.Dense(units=6, activation='relu'),
    layers.Dense(units=1, activation='sigmoid')
    # ____
])

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

Correct

Out[5]:

```
(None,)
```

In [6]:

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

3) Train

Before training a model in Keras, you need to specify an *optimizer* to perform the gradient descent, a *loss function* to be minimized, and (optionally) any *performance metrics*. The optimization algorithm we'll use for this course is called "Adam", which generally performs well regardless of what kind of problem you're trying to solve.

The loss and the metrics, however, need to match the kind of problem you're trying to solve. Our problem is a **binary classification** problem: `Car` coded as 0, and `Truck` coded as 1. Choose an appropriate loss and an appropriate accuracy metric for binary classification.

In [7]:
```
# 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

In [8]:
```
# Lines below will give you a hint or solution code
#q_3.hint()
#q_3.solution()
```

In [9]:
```
history = model.fit(
    ds_train,
    validation_data=ds_valid,
    epochs=30,
)
```

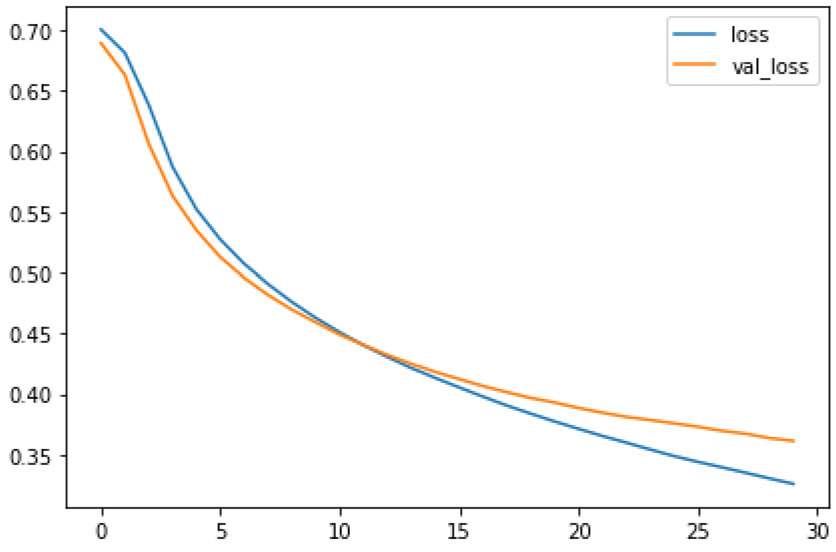
```
2022-12-18 05:57:22.194895: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:1
85] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/30
2022-12-18 05:57:29.410274: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cu
DNN version 8005
80/80 [==============================] - 43s 389ms/step - loss: 0.7006 - binary_accurac
y: 0.5419 - val_loss: 0.6893 - val_binary_accuracy: 0.6003
Epoch 2/30
80/80 [==============================] - 3s 41ms/step - loss: 0.6814 - binary_accuracy:
0.6490 - val_loss: 0.6632 - val_binary_accuracy: 0.7468
Epoch 3/30
80/80 [==============================] - 3s 41ms/step - loss: 0.6384 - binary_accuracy:
0.7340 - val_loss: 0.6062 - val_binary_accuracy: 0.7854
Epoch 4/30
80/80 [==============================] - 3s 42ms/step - loss: 0.5872 - binary_accuracy:
0.7799 - val_loss: 0.5632 - val_binary_accuracy: 0.8125
Epoch 5/30
80/80 [==============================] - 3s 41ms/step - loss: 0.5522 - binary_accuracy:
0.8013 - val_loss: 0.5348 - val_binary_accuracy: 0.8284
Epoch 6/30
80/80 [==============================] - 3s 43ms/step - loss: 0.5273 - binary_accuracy:
0.8165 - val_loss: 0.5126 - val_binary_accuracy: 0.8313
Epoch 7/30
80/80 [==============================] - 3s 41ms/step - loss: 0.5072 - binary_accuracy:
0.8247 - val_loss: 0.4958 - val_binary_accuracy: 0.8363
Epoch 8/30
80/80 [==============================] - 3s 40ms/step - loss: 0.4908 - binary_accuracy:
0.8306 - val_loss: 0.4818 - val_binary_accuracy: 0.8386
Epoch 9/30
```

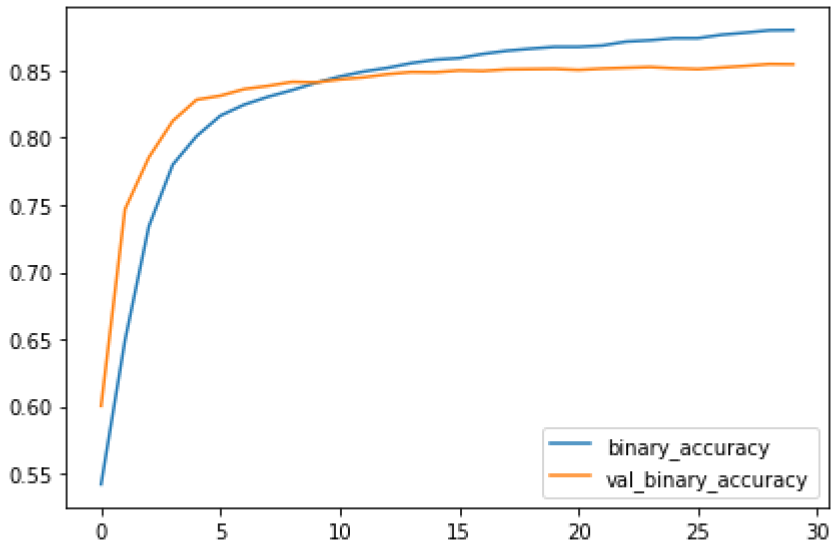
```
80/80 [==============================] - 3s 42ms/step - loss: 0.4760 - binary_accuracy:
0.8355 - val_loss: 0.4697 - val_binary_accuracy: 0.8416
Epoch 10/30
80/80 [==============================] - 3s 40ms/step - loss: 0.4629 - binary_accuracy:
0.8409 - val_loss: 0.4592 - val_binary_accuracy: 0.8412
Epoch 11/30
80/80 [==============================] - 3s 41ms/step - loss: 0.4511 - binary_accuracy:
0.8456 - val_loss: 0.4491 - val_binary_accuracy: 0.8434
Epoch 12/30
80/80 [==============================] - 3s 41ms/step - loss: 0.4404 - binary_accuracy:
0.8493 - val_loss: 0.4406 - val_binary_accuracy: 0.8450
Epoch 13/30
80/80 [==============================] - 3s 41ms/step - loss: 0.4307 - binary_accuracy:
0.8521 - val_loss: 0.4323 - val_binary_accuracy: 0.8474
Epoch 14/30
80/80 [==============================] - 3s 41ms/step - loss: 0.4215 - binary_accuracy:
0.8556 - val_loss: 0.4250 - val_binary_accuracy: 0.8489
Epoch 15/30
80/80 [==============================] - 4s 45ms/step - loss: 0.4132 - binary_accuracy:
0.8581 - val_loss: 0.4182 - val_binary_accuracy: 0.8487
Epoch 16/30
80/80 [==============================] - 3s 41ms/step - loss: 0.4054 - binary_accuracy:
0.8593 - val_loss: 0.4123 - val_binary_accuracy: 0.8501
Epoch 17/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3978 - binary_accuracy:
0.8624 - val_loss: 0.4065 - val_binary_accuracy: 0.8497
Epoch 18/30
80/80 [==============================] - 3s 42ms/step - loss: 0.3904 - binary_accuracy:
0.8648 - val_loss: 0.4015 - val_binary_accuracy: 0.8509
Epoch 19/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3838 - binary_accuracy:
0.8663 - val_loss: 0.3968 - val_binary_accuracy: 0.8511
Epoch 20/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3773 - binary_accuracy:
0.8677 - val_loss: 0.3930 - val_binary_accuracy: 0.8513
Epoch 21/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3713 - binary_accuracy:
0.8677 - val_loss: 0.3885 - val_binary_accuracy: 0.8505
Epoch 22/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3654 - binary_accuracy:
0.8687 - val_loss: 0.3846 - val_binary_accuracy: 0.8515
Epoch 23/30
80/80 [==============================] - 3s 43ms/step - loss: 0.3599 - binary_accuracy:
0.8716 - val_loss: 0.3811 - val_binary_accuracy: 0.8521
Epoch 24/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3542 - binary_accuracy:
0.8726 - val_loss: 0.3786 - val_binary_accuracy: 0.8527
Epoch 25/30
80/80 [==============================] - 3s 40ms/step - loss: 0.3488 - binary_accuracy:
0.8741 - val_loss: 0.3758 - val_binary_accuracy: 0.8517
Epoch 26/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3440 - binary_accuracy:
0.8741 - val_loss: 0.3730 - val_binary_accuracy: 0.8511
Epoch 27/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3396 - binary_accuracy:
0.8767 - val_loss: 0.3697 - val_binary_accuracy: 0.8523
Epoch 28/30
80/80 [==============================] - 3s 41ms/step - loss: 0.3350 - binary_accuracy:
0.8782 - val_loss: 0.3672 - val_binary_accuracy: 0.8535
Epoch 29/30
```

```
80/80 [==============================] - 3s 41ms/step - loss: 0.3304 - binary_accuracy:
0.8800 - val_loss: 0.3636 - val_binary_accuracy: 0.8549
Epoch 30/30
80/80 [==============================] - 3s 40ms/step - loss: 0.3258 - binary_accuracy:
0.8802 - val_loss: 0.3613 - val_binary_accuracy: 0.8547
```

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

In [10]:
```
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();
```





4) Examine Loss and Accuracy

Do you notice a difference between these learning curves and the curves for VGG16 from the tutorial? What does this difference tell you about what this model (InceptionV2) learned compared to VGG16? Are there ways in which one is better than the other? Worse?

After you've thought about it, run the cell below to see the answer.

In [11]:
```
# View the solution (Run this code cell to receive credit!)
q_4.check()
```

Correct:

That the training loss and validation loss stay fairly close is evidence that the model isn't just memorizing the training data, but rather learning general properties of the two classes. But, because this model converges at a loss greater than the VGG16 model, it's likely that it is underfitting some, and could benefit from some extra capacity.

Conclusion

In this first lesson, you learned the basics of **convolutional image classifiers**, that they consist of a **base** for extracting features from images, and a **head** which uses the features to decide the image's class. You also saw how to build a classifier with **transfer learning** on pretrained base.

Keep Going

Move on to **Lesson 2** for a detailed look at how the base does this feature extraction. (It's really cool!)

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