Open in Colab

T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- **Part 9.2: Keras Transfer Learning for Computer Vision** [Video] [Notebook]
- Part 9.3: Transfer Learning for NLP with Keras [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

Part 9.2: Keras Transfer Learning for Computer Vision

We will take a look at several popular pretrained neural networks for Keras. The following two sites, among others, can be great starting points to find pretrained models for use in your projects:

- TensorFlow Model Zoo
- Papers with Code

Keras contains built-in support for several pretrained models. In the Keras documentation, you can find the complete list.

Transfering Computer Vision

There are many pretrained models for computer vision. This section will show you how to obtain a pretrained model for computer vision and train just the output layer. Additionally, once we train the output layer, we will fine-tune the entire network by training all weights using by applying a low learning rate.

The Kaggle Cats vs. Dogs Dataset

We will train a neural network to recognize cats and dogs for this example. The [cats and dogs dataset] comes from a classic Kaggle competition. We can achieve a very high score on this data set through modern training techniques and ensemble learning. I based this module on a tutorial provided by Francois Chollet, one of the creators of Keras. I made some changes to his example to fit with this course.

We begin by downloading this dataset from Keras. We do not need the entire dataset to achieve high accuracy. Using a portion also speeds up training. We will use 40% of the original training data (25,000 images) for training and 10% for validation.

The dogs and cats dataset is relatively large and will not easily fit into a less than 12GB system, such as Colab. Because of this memory size, you must take additional steps to handle the data. Rather than loading the dataset as a Numpy array, as done previously in this book, we will load it as a prefetched dataset so that only the portions of the dataset currently needed are in RAM. If you wish to load the dataset, in its entirety as a Numpy array, add the batch_size=-1 option to the load command below.

In [ ]:
```
import tensorflow_datasets as tfds
import tensorflow as tf

tfds.disable_progress_bar()

train_ds, validation_ds = tfds.load(
    "cats_vs_dogs",
    split=["train[:40%]", "train[40%:50%]"],
    as_supervised=True,  # Include labels
)

num_train = tf.data.experimental.cardinality(train_ds)
num_test = tf.data.experimental.cardinality(validation_ds)

print(f"Number of training samples: {num_train}")
print(f"Number of validation samples: {num_test}")
```

```
Number of training samples: 9305
Number of validation samples: 2326
```

Looking at the Data and Augmentations

We begin by displaying several of the images from this dataset. The labels are above each image. As can be seen from the images below, 1 indicates a dog, and 0 indicates a cat.

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(train_ds.take(9)):
```

```
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(int(label))
    plt.axis("off")
```

Upon examining the above images, another problem becomes evident. The images are of various sizes. We will standardize all images to 190x190 with the following code.

In [ ]:
```
size = (150, 150)

train_ds = train_ds.map(lambda x, y: (tf.image.resize(x, size), y))
validation_ds = validation_ds.map(lambda x, y: \
                                  (tf.image.resize(x, size), y))
```

We will batch the data and use caching and prefetching to optimize loading speed.

In [ ]:
```
batch_size = 32

train_ds = train_ds.cache().batch(batch_size).prefetch(buffer_size=10)
```

```
validation_ds = validation_ds.cache() \
    .batch(batch_size).prefetch(buffer_size=10)
```

Augmentation is a powerful computer vision technique that increases the amount of training data available to your model by altering the images in the training data. To use augmentation, we will allow horizontal flips of the images. A horizontal flip makes much more sense for cats and dogs in the real world than a vertical flip. How often do you see upside-down dogs or cats? We also include a limited degree of rotation.

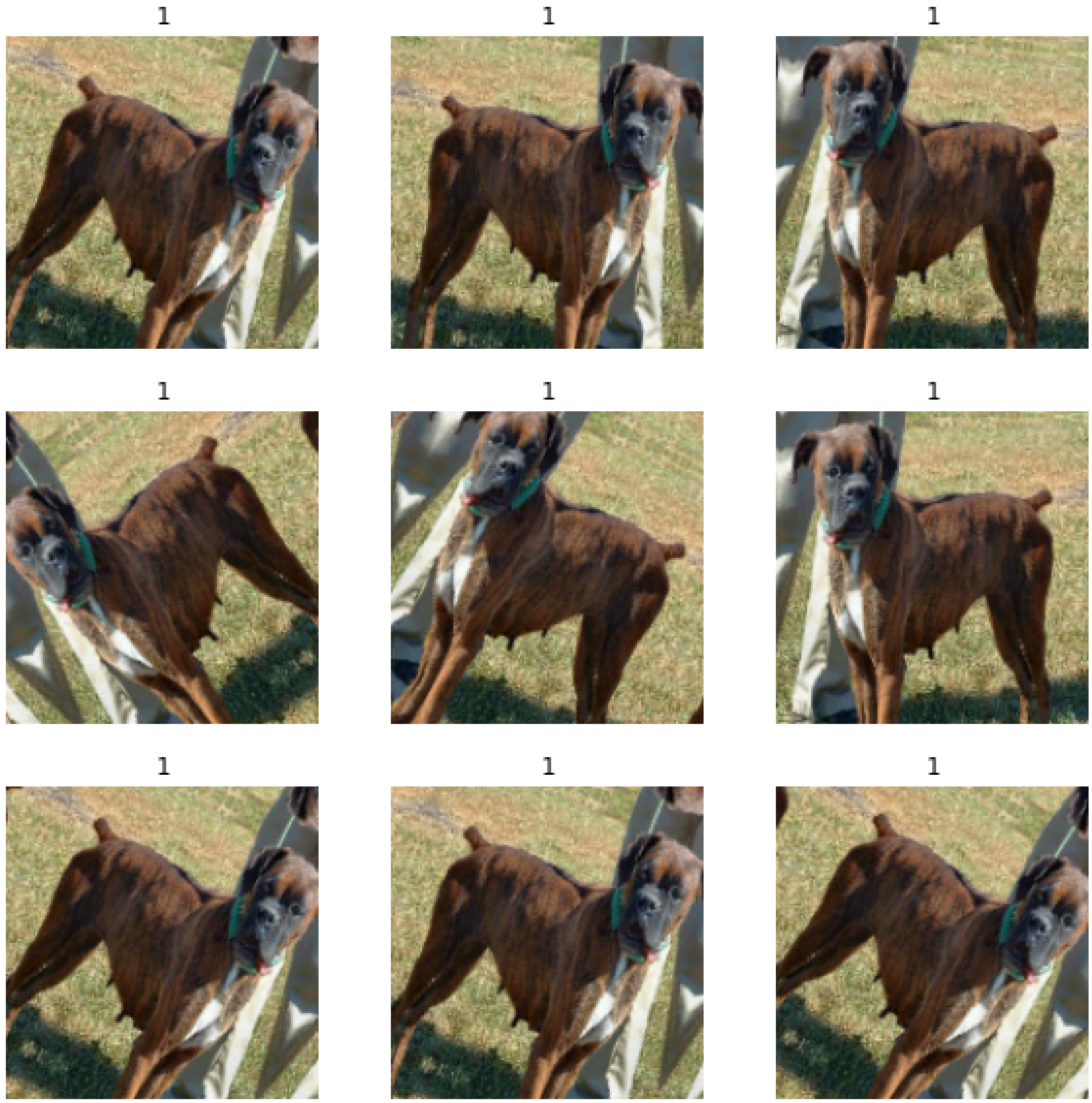
In [ ]:
```
from tensorflow import keras
from tensorflow.keras import layers

data_augmentation = keras.Sequential(
    [layers.RandomFlip("horizontal"), layers.RandomRotation(0.1),]
)
```

The following code allows us to visualize the augmentation.

In [ ]:
```
import numpy as np

for images, labels in train_ds.take(1):
    plt.figure(figsize=(10, 10))
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0].numpy().astype("int32"))
        plt.title(int(labels[0]))
        plt.axis("off")
```



Create a Network and Transfer Weights

We are now ready to create our new neural network with transferred weights. We will transfer the weights from an Xception neural network that contains weights trained for imagenet. We load the existing Xception neural network with **keras.applications**. There is quite a bit going on with the loading of the **base_model**, so we will examine this call piece by piece.

The base Xception neural network accepts an image of 299x299. However, we would like to use 150x150. It turns out that it is relatively easy to overcome this difference. Convolutional neural networks move a kernel across an image tensor as they scan. Keras defines the number of weights by the size of the layer's kernel, not the image that the kernel scans. As a result, we can discard the old input layer and recreate an input layer consistent with our desired image size. We specify **include_top** as false and specify our input shape.

We freeze the base model so that the model will not update existing weights as training occurs. We create the new input layer that consists of 150x150 by 3 RGB color components. These RGB components are integer numbers between 0 and 255. Neural networks deal better with floating-point numbers when you distribute them around zero. To accomplish this neural network advantage, we normalize each RGB component to between -1 and 1.

The batch normalization layers do require special consideration. We need to keep these layers in inference mode when we unfreeze the base model for fine-tuning. To do this, we make sure that the base model is running in inference mode here.

In [ ]:
```
base_model = keras.applications.Xception(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False,
)  # Do not include the ImageNet classifier at the top.

# Freeze the base_model
base_model.trainable = False

# Create new model on top
inputs = keras.Input(shape=(150, 150, 3))
x = data_augmentation(inputs)  # Apply random data augmentation

# Pre-trained Xception weights requires that input be scaled
# from (0, 255) to a range of (-1., +1.), the rescaling layer
# outputs: `(inputs * scale) + offset`
scale_layer = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)
x = scale_layer(x)

# The base model contains batchnorm layers.
# We want to keep them in inference mode
# when we unfreeze the base model for fine-tuning,
# so we make sure that the
# base_model is running in inference mode here.
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 150, 150, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 150, 150, 3)       0         
                                                                 
 xception (Functional)       (None, 5, 5, 2048)        20861480  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout (Dropout)           (None, 2048)              0         
                                                                 
 dense (Dense)               (None, 1)                 2049      
                                                                 
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

Next, we compile and fit the model. The fitting will use the Adam optimizer; because we are performing binary classification, we use the binary cross-entropy loss function, as we have done before.

In [ ]:
```
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)

epochs = 20
model.fit(train_ds, epochs=epochs, validation_data=validation_ds)
```

```
Epoch 1/20
291/291 [==============================] - 31s 70ms/step - loss: 0.1735 - bi
nary_accuracy: 0.9240 - val_loss: 0.0831 - val_binary_accuracy: 0.9678
Epoch 2/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1256 - bi
nary_accuracy: 0.9463 - val_loss: 0.0773 - val_binary_accuracy: 0.9703
Epoch 3/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1140 - bi
nary_accuracy: 0.9536 - val_loss: 0.0750 - val_binary_accuracy: 0.9708
Epoch 4/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1097 - bi
nary_accuracy: 0.9556 - val_loss: 0.0729 - val_binary_accuracy: 0.9729
Epoch 5/20
291/291 [==============================] - 12s 41ms/step - loss: 0.0996 - bi
nary_accuracy: 0.9589 - val_loss: 0.0717 - val_binary_accuracy: 0.9746
Epoch 6/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0979 - bi
nary_accuracy: 0.9587 - val_loss: 0.0734 - val_binary_accuracy: 0.9690
Epoch 7/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1007 - bi
nary_accuracy: 0.9601 - val_loss: 0.0738 - val_binary_accuracy: 0.9703
Epoch 8/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0968 - bi
nary_accuracy: 0.9591 - val_loss: 0.0709 - val_binary_accuracy: 0.9729
Epoch 9/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0940 - bi
nary_accuracy: 0.9607 - val_loss: 0.0713 - val_binary_accuracy: 0.9733
Epoch 10/20
291/291 [==============================] - 11s 37ms/step - loss: 0.1019 - bi
nary_accuracy: 0.9599 - val_loss: 0.0731 - val_binary_accuracy: 0.9690
Epoch 11/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0904 - bi
nary_accuracy: 0.9640 - val_loss: 0.0694 - val_binary_accuracy: 0.9738
Epoch 12/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0953 - bi
nary_accuracy: 0.9629 - val_loss: 0.0777 - val_binary_accuracy: 0.9712
Epoch 13/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0922 - bi
nary_accuracy: 0.9635 - val_loss: 0.0735 - val_binary_accuracy: 0.9738
Epoch 14/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0975 - bi
nary_accuracy: 0.9610 - val_loss: 0.0714 - val_binary_accuracy: 0.9733
Epoch 15/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0916 - bi
nary_accuracy: 0.9634 - val_loss: 0.0770 - val_binary_accuracy: 0.9699
Epoch 16/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0957 - bi
nary_accuracy: 0.9605 - val_loss: 0.0713 - val_binary_accuracy: 0.9721
Epoch 17/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0899 - bi
nary_accuracy: 0.9638 - val_loss: 0.0731 - val_binary_accuracy: 0.9725
Epoch 18/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0937 - bi
nary_accuracy: 0.9620 - val_loss: 0.0755 - val_binary_accuracy: 0.9712
Epoch 19/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0907 - bi
```

```
nary_accuracy: 0.9627 - val_loss: 0.0718 - val_binary_accuracy: 0.9729
Epoch 20/20
291/291 [==============================] - 11s 37ms/step - loss: 0.0899 - bi
nary_accuracy: 0.9652 - val_loss: 0.0694 - val_binary_accuracy: 0.9746
```

Out[ ]:
```
<keras.callbacks.History at 0x7f0c2cfbf410>
```

The training above shows that the validation accuracy reaches the mid 90% range. This accuracy is good; however, we can do better.

Fine-Tune the Model

Finally, we will fine-tune the model. First, we set all weights to trainable and then train the neural network with a low learning rate (1e-5). This fine-tuning results in an accuracy in the upper 90% range. The fine-tuning allows all weights in the neural network to adjust slightly to optimize for the dogs/cats data.

In [ ]:
```
# Unfreeze the base_model. Note that it keeps running in inference mode
# since we passed `training=False` when calling it. This means that
# the batchnorm layers will not update their batch statistics.
# This prevents the batchnorm layers from undoing all the training
# we've done so far.
base_model.trainable = True
model.summary()

model.compile(
    optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)

epochs = 10
model.fit(train_ds, epochs=epochs, validation_data=validation_ds)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 150, 150, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 150, 150, 3)       0         
                                                                 
 xception (Functional)       (None, 5, 5, 2048)        20861480  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout (Dropout)           (None, 2048)              0         
                                                                 
 dense (Dense)               (None, 1)                 2049      
                                                                 
=================================================================
Total params: 20,863,529
Trainable params: 20,809,001
Non-trainable params: 54,528
_________________________________________________________________
Epoch 1/10
291/291 [==============================] - 48s 144ms/step - loss: 0.0771 - b
inary_accuracy: 0.9712 - val_loss: 0.0533 - val_binary_accuracy: 0.9772
Epoch 2/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0571 - b
inary_accuracy: 0.9774 - val_loss: 0.0484 - val_binary_accuracy: 0.9811
Epoch 3/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0412 - b
inary_accuracy: 0.9830 - val_loss: 0.0467 - val_binary_accuracy: 0.9819
Epoch 4/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0354 - b
inary_accuracy: 0.9861 - val_loss: 0.0543 - val_binary_accuracy: 0.9785
Epoch 5/10
291/291 [==============================] - 41s 141ms/step - loss: 0.0313 - b
inary_accuracy: 0.9876 - val_loss: 0.0490 - val_binary_accuracy: 0.9807
Epoch 6/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0231 - b
inary_accuracy: 0.9911 - val_loss: 0.0625 - val_binary_accuracy: 0.9776
Epoch 7/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0227 - b
inary_accuracy: 0.9909 - val_loss: 0.0579 - val_binary_accuracy: 0.9798
Epoch 8/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0195 - b
inary_accuracy: 0.9937 - val_loss: 0.0493 - val_binary_accuracy: 0.9837
Epoch 9/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0136 - b
inary_accuracy: 0.9952 - val_loss: 0.0447 - val_binary_accuracy: 0.9837
Epoch 10/10
291/291 [==============================] - 41s 140ms/step - loss: 0.0162 - b
inary_accuracy: 0.9944 - val_loss: 0.0548 - val_binary_accuracy: 0.9819
```

Out[ ]:
```
<keras.callbacks.History at 0x7f0c2ceaafd0>
```