Transfer Learning

In this lab, you will see how you can use a pre-trained model to achieve good results even with a small training dataset. This is called transfer learning and you do this by leveraging the trained layers of an existing model and adding your own layers to fit your application. For example, you can:

just get the convolution layers of one model attach some dense layers onto it train just the dense network evaluate the results Doing this will allow you to save time building your application because you will essentially skip weeks of training time of very deep networks. You will just use the features it has learned and tweak it for your dataset. Let's see how these are done in the next sections.

IMPORTANT NOTE: This notebook is designed to run as a Colab. Running the notebook on your local machine might result in some of the code blocks throwing errors.

Setup the pretrained model

You will need to prepare pretrained model and configure the layers that you need. For this exercise, you will use the convolution layers of the InceptionV3 architecture as your base model. To do that, you need to:

1. Set the input shape to fit your application. In this case. set it to 150x150x3 as you've been doing in the last few labs.
2. Pick and freeze the convolution layers to take advantage of the features it has learned already.
3. Add dense layers which you will train.

Let's see how to do these in the next cells.

First, in preparing the input to the model, you want to fetch the pretrained weights of the InceptionV3 model and remove the fully connected layer at the end because you will be replacing it later. You will also specify the input shape that your model will accept. Lastly, you want to freeze the weights of these layers because they have been trained already.

```
# Download the pre-trained weights. No top means it excludes the fully connected layer it use
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_ker
    -O /tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras import layers
```

```
# Set the weights file you downloaded into a variable
local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

# Initialize the base model.
# Set the input shape and remove the dense layers.
pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)

# Load the pre-trained weights you downloaded.
pre_trained_model.load_weights(local_weights_file)

# Freeze the weights of the layers.
for layer in pre_trained_model.layers:
  layer.trainable = False
```

You can see the summary of the model below. You can see that it is a very deep network. You can then select up to which point of the network you want to use. As Laurence showed in the exercise, you will use up to mixed_7 as your base model and add to that. This is because the original last layer might be too specialized in what it has learned so it might not translate well into your application. mixed_7 on the other hand will be more generalized and you can start with that for your application. After the exercise, feel free to modify and use other layers to see what the results you get.

```
pre_trained_model.summary()
```

```
conv2d_92 (Conv2D)              (None, 3, 3, 384)    442368      ['activation_90[0][0
average_pooling2d_8 (AveragePo  (None, 3, 3, 2048)  0           ['mixed9[0][0]']
oling2D)
conv2d_85 (Conv2D)              (None, 3, 3, 320)    655360      ['mixed9[0][0]']
batch_normalization_87 (BatchN  (None, 3, 3, 384)   1152        ['conv2d_87[0][0]']
ormalization)
batch_normalization_88 (BatchN  (None, 3, 3, 384)   1152        ['conv2d_88[0][0]']
ormalization)
batch_normalization_91 (BatchN  (None, 3, 3, 384)   1152        ['conv2d_91[0][0]']
ormalization)
batch_normalization_92 (BatchN  (None, 3, 3, 384)   1152        ['conv2d_92[0][0]']
ormalization)
conv2d_93 (Conv2D)              (None, 3, 3, 192)    393216      ['average_pooling2d_
```

```
batch_normalization_85 (BatchN  (None, 3, 3, 320)   960         ['conv2d_85[0][0]']
ormalization)

activation_87 (Activation)     (None, 3, 3, 384)    0           ['batch_normalizatic

activation_88 (Activation)     (None, 3, 3, 384)    0           ['batch_normalizatic

activation_91 (Activation)     (None, 3, 3, 384)    0           ['batch_normalizatic

activation_92 (Activation)     (None, 3, 3, 384)    0           ['batch_normalizatic

batch_normalization_93 (BatchN  (None, 3, 3, 192)   576         ['conv2d_93[0][0]']
ormalization)

activation_85 (Activation)     (None, 3, 3, 320)    0           ['batch_normalizatic

mixed9_1 (Concatenate)         (None, 3, 3, 768)    0           ['activation_87[0][0
                                                                 'activation_88[0][0

concatenate_1 (Concatenate)    (None, 3, 3, 768)    0           ['activation_91[0][0
                                                                 'activation_92[0][0

activation_93 (Activation)     (None, 3, 3, 192)    0           ['batch_normalizatic

mixed10 (Concatenate)          (None, 3, 3, 2048)   0           ['activation_85[0][0
                                                                 'mixed9_1[0][0]',
                                                                 'concatenate_1[0][0
                                                                 'activation_93[0][0

==================================================================================================
Total params: 21,802,784
Trainable params: 0
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

```
# Choose `mixed_7` as the last layer of your base model
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output
```

```
last layer output shape:  (None, 7, 7, 768)
```

Add dense layers for your classifier

Next, you will add dense layers to your model. These will be the layers that you will train and is tasked with recognizing cats and dogs. You will add a Dropout layer as well to regularize the output and avoid overfitting.

```
from tensorflow.keras.optimizers import RMSprop
```

```
from tensorflow.keras import Model

# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense  (1, activation='sigmoid')(x)

# Append the dense network to the base model
model = Model(pre_trained_model.input, x)

# Print the model summary. See your dense network connected at the end.
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 74, 74, 32)   864         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 74, 74, 32)  96          ['conv2d[0][0]']
 alization)

 activation (Activation)        (None, 74, 74, 32)   0           ['batch_normalizatio

 conv2d_1 (Conv2D)              (None, 72, 72, 32)   9216        ['activation[0][0]']

 batch_normalization_1 (BatchNo  (None, 72, 72, 32)  96          ['conv2d_1[0][0]']
 rmalization)

 activation_1 (Activation)      (None, 72, 72, 32)   0           ['batch_normalizatio

 conv2d_2 (Conv2D)              (None, 72, 72, 64)   18432       ['activation_1[0][0]

 batch_normalization_2 (BatchNo  (None, 72, 72, 64)  192         ['conv2d_2[0][0]']
 rmalization)

 activation_2 (Activation)      (None, 72, 72, 64)   0           ['batch_normalizatio

 max_pooling2d (MaxPooling2D)   (None, 35, 35, 64)   0           ['activation_2[0][0]

 conv2d_3 (Conv2D)              (None, 35, 35, 80)   5120        ['max_pooling2d[0][0

 batch_normalization_3 (BatchNo  (None, 35, 35, 80)  240         ['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 35, 35, 80)   0           ['batch_normalizatio
```

```
conv2d_4 (Conv2D)              (None, 33, 33, 192)  138240      ['activation_3[0][0]
batch_normalization_4 (BatchNo  (None, 33, 33, 192)  576        ['conv2d_4[0][0]']
rmalization)
activation_4 (Activation)      (None, 33, 33, 192)  0           ['batch_normalizatic
max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 192)  0          ['activation_4[0][0]
conv2d_8 (Conv2D)              (None, 16, 16, 64)   12288       ['max_pooling2d_1[0]
batch_normalization_8 (BatchNo  (None, 16, 16, 64)  192         ['conv2d_8[0][0]']
rmalization)
activation_8 (Activation)      (None, 16, 16, 64)   0           ['batch_normalizatic
conv2d_6 (Conv2D)              (None, 16, 16, 48)   9216        ['max_pooling2d_1[0]
conv2d_9 (Conv2D)              (None, 16, 16, 96)   55296       ['activation_8[0][0]
```

```
# Set the training parameters
model.compile(optimizer·=·RMSprop(learning_rate=0.0001),·
··············loss·=·'binary_crossentropy',·
··············metrics·=·['accuracy'])
```

Prepare the dataset

```
# Download the dataset
!wget https://storage.googleapis.com/tensorflow-1-public/course2/cats_and_dogs_filtered.zip
```

```
--2022-06-14 14:30:40--  https://storage.googleapis.com/tensorflow-1-public/course2/cats
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.197.128, 74.125.135
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.197.128|:443... cor
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: ‘cats_and_dogs_filtered.zip’

cats_and_dogs_filte 100%[===================>]  65.43M   142MB/s    in 0.5s

2022-06-14 14:30:41 (142 MB/s) - ‘cats_and_dogs_filtered.zip’ saved [68606236/68606236]
```

```
import os
import zipfile
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Extract the archive
zip_ref = zipfile.ZipFile("./cats_and_dogs_filtered.zip", 'r')
```

```
zip_ref.extractall("tmp/")
zip_ref.close()

# Define our example directories and files
base_dir = 'tmp/cats_and_dogs_filtered'

train_dir = os.path.join( base_dir, 'train')
validation_dir = os.path.join( base_dir, 'validation')

# Directory with training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')

# Directory with training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')

# Directory with validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255.,
                                   rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator( rescale = 1.0/255. )

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150))

# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory( validation_dir,
                                                          batch_size  = 20,
                                                          class_mode  = 'binary',
                                                          target_size = (150, 150))
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Train The Model

```
# Train the model.
history = model.fit(
            train_generator,
            validation_data = validation_generator,
            steps_per_epoch = 100,
            epochs = 20,
            validation_steps = 50,
            verbose = 2)
```

```
Epoch 1/20
100/100 - 164s - loss: 0.3459 - accuracy: 0.8670 - val_loss: 0.0968 - val_accuracy: 0.95
Epoch 2/20
100/100 - 156s - loss: 0.2478 - accuracy: 0.9045 - val_loss: 0.2051 - val_accuracy: 0.93
Epoch 3/20
100/100 - 154s - loss: 0.2017 - accuracy: 0.9220 - val_loss: 0.0942 - val_accuracy: 0.96
Epoch 4/20
100/100 - 155s - loss: 0.2187 - accuracy: 0.9255 - val_loss: 0.0851 - val_accuracy: 0.97
Epoch 5/20
100/100 - 153s - loss: 0.1805 - accuracy: 0.9340 - val_loss: 0.1225 - val_accuracy: 0.96
Epoch 6/20
100/100 - 154s - loss: 0.1975 - accuracy: 0.9335 - val_loss: 0.1413 - val_accuracy: 0.95
Epoch 7/20
100/100 - 153s - loss: 0.1822 - accuracy: 0.9325 - val_loss: 0.3446 - val_accuracy: 0.91
Epoch 8/20
100/100 - 153s - loss: 0.1778 - accuracy: 0.9380 - val_loss: 0.1330 - val_accuracy: 0.96
Epoch 9/20
100/100 - 154s - loss: 0.1624 - accuracy: 0.9465 - val_loss: 0.1309 - val_accuracy: 0.96
Epoch 10/20
100/100 - 157s - loss: 0.1472 - accuracy: 0.9500 - val_loss: 0.1073 - val_accuracy: 0.97
Epoch 11/20
```

Evaluate the results

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()
```

```
plt.show()
```