Transfer Learning with Inception Network

One of the most powerful ideas in deep learning is that sometimes we can take knowledge the neural network has learned from one task and apply that knowledge to a separate task. So for example, maybe we could have the neural network learn to recognize objects like cats and then use that knowledge or use part of that knowledge to help us do a better job reading x-ray scans. This is called **transfer learning**.

Let's say we've trained our neural network on some task. If we want to take this neural network and adapt what is learned to a different task, what we can do is delete the last one or more layers of the neural network, and create several new layers. Depending on how much data we have, we might just retrain the new layers of the network or maybe we could retrain even more layers of this neural network.

Transfer learning makes sense when we have a lot of data for the problem we're transferring from and usually relatively less data for the problem we're transferring to.

The motivation of **Inception Network** is, instead of picking a filter size or thinking about if we want to use a conv layer or pooling layer, we can do them all and just concatenate all the outputs, and let the network learn whatever parameters it wants to use. To save the computation time, we can use 1x1 convolution to shrink the channel number.

In this notebook, we will use a pre-trained Inception Network for cat-dog classification task.

In [1]:
```
%matplotlib inline
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False
import matplotlib.pyplot as plt
import os
import zipfile
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, Adam
from keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

```
Using TensorFlow backend.
```

1. Data Pre-Processing

The following python code will use the OS library to use Operating System libraries, giving us access to the file system, and the zipfile library allowing us to unzip the data.

1.1 File Preprocessing

In [2]:
```
# training and development dataset

zipfile_path = 'data/cats_dogs_train.zip'
zip_file = zipfile.ZipFile(zipfile_path, 'r') # Open a ZIP file
zip_file.extractall(path = 'data/') # Extract all members from the archive
zip_file.close() # Close the archive file

zipfile_path = 'data/cats_dogs_validation.zip'
zip_file = zipfile.ZipFile(zipfile_path, 'r') # Open a ZIP file
zip_file.extractall(path = 'data/') # Extract all members from the archive
zip_file.close() # Close the archive file
```

The contents of the .zip are extracted to the base directory `data/cats_and_dogs_filtered/train` (for training) and `data/cats_and_dogs_filtered/validation` (for development), which in turn each contains `cats` and `dogs` subdirectories.

We do not explicitly label the images as cats or dogs. We'll use Image Generator -- and this is coded to read images from subdirectories, and automatically label them from the name of that subdirectory.

In [3]:
```
# Directory with our training cat pictures
train_cat_dir = os.path.join('data/cats_dogs_train/train/cats')
train_cat_names = os.listdir(train_cat_dir)
# Directory with our training dog pictures
train_dog_dir = os.path.join('data/cats_dogs_train/train/dogs')
train_dog_names = os.listdir(train_dog_dir)
# Directory with our dev cat pictures
dev_cat_dir = os.path.join('data/cats_dogs_validation/validation/cats')
dev_cat_names = os.listdir(dev_cat_dir)
# Directory with our dev dog pictures
dev_dog_dir = os.path.join('data/cats_dogs_validation/validation/dogs')
dev_dog_names = os.listdir(dev_dog_dir)
```

Let's find out the total number of cat and dog images in the directories.

In [4]:
```
print('total training cat images:', len(train_cat_names))
print('total training dog images:', len(train_dog_names))
print('total dev cat images:', len(dev_cat_names))
print('total dev dog images:', len(dev_dog_names))
```

```
total training cat images: 1000
total training dog images: 1000
total dev cat images: 500
total dev dog images: 500
```

1.2 Image Data Examples

Now let's take a look at a few pictures to get a better sense of what they look like.

In [5]:
```
# we'll output images in a nrows x ncols configuration

nrows = 4
ncols = 4

train_cat_paths = [os.path.join(train_cat_dir, fname) for fname in train_cat_names]
train_dog_paths = [os.path.join(train_dog_dir, fname) for fname in train_dog_names]
dev_cat_paths = [os.path.join(dev_cat_dir, fname) for fname in dev_cat_names]
dev_dog_paths = [os.path.join(dev_dog_dir, fname) for fname in dev_dog_names]
```

Display a batch of 8 cat and 8 dog pictures.

In [6]:

```
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 3)

# iterating over 16 images
for i, img_path in enumerate(train_cat_paths[10:14] + dev_cat_paths[10:14] + train_dog_paths[10:14] + dev_dog_paths[10:14]):
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img)
```

1.3. Image Data Augmentation and Generator

In [7]:
```
# create image data

# training set with image data augmentation
train_gen = 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,
        fill_mode = 'nearest')
train_generator = train_gen.flow_from_directory(
        'data/cats_dogs_train/train/',
        target_size = (150, 150),  # All images will be resized to this dimension
        batch_size = 20,
        class_mode = 'binary')

# dev set
dev_gen = ImageDataGenerator(rescale = 1/255)
dev_generator = dev_gen.flow_from_directory(
        'data/cats_dogs_validation/validation/',
        target_size = (150, 150),  # All images will be resized to this dimension
        batch_size = 20,
        class_mode = 'binary')

# print class indices
print("Class index: "  + str(train_generator.class_indices))
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Class index: {'cats': 0, 'dogs': 1}
```

2. Transfer Learning

We use an **Inception Model** pre-trained with a large data set. We will create new layers on top of the 'mixed7' layer.

Note that because we are facing a two-class classification problem, i.e. a **binary classification problem**, we will end our network with a **sigmoid** activation, so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0).

In [8]:
```
model_pretrained = InceptionV3(include_top = False, # do not include the fully-connected layer
                               input_shape = (150, 150, 3),
                               weights = None)

# load the weights
model_pretrained.load_weights('model/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5')

# To freeze keras layers, set the trainable property of a layer to True or False after instantiation.
for layer in model_pretrained.layers:
    layer.trainable = False

# get the output from a specific layer, we will start from this layer
model_pretrained_out = model_pretrained.get_layer('mixed7')
print('Pre-trained model output shape:', model_pretrained_out.output_shape)
```

```
Pre-trained model output shape: (None, 7, 7, 768)
```

In [9]:
```
# complete model

# output from pre-trained model
x = model_pretrained_out.output
# flatten the results to feed into a DNN
x = layers.Flatten()(x)
# 1024 neuron hidden layer
x = layers.Dense(1024, activation='relu')(x)
# add a drop out layer
x = layers.Dropout(rate = 0.2)(x)
# Only 1 output neuron. It will contain a value from 0-1 where 0 for 'cats' and 1 for 'dogs'
x = layers.Dense(1, activation='sigmoid')(x)

model = Model(inputs = model_pretrained.input, outputs = x)
```

In [10]:
```
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 (BatchNorma	(None, 74, 74, 32)	96	conv2d[0][0]
activation (Activation)	(None, 74, 74, 32)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 72, 72, 32)	9216	activation[0][0]
batch_normalization_1 (BatchNor	(None, 72, 72, 32)	96	conv2d_1[0][0]
activation_1 (Activation)	(None, 72, 72, 32)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18432	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 72, 72, 64)	192	conv2d_2[0][0]
activation_2 (Activation)	(None, 72, 72, 64)	0	batch_normalization_2[0][0]
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 (BatchNor	(None, 35, 35, 80)	240	conv2d_3[0][0]
activation_3 (Activation)	(None, 35, 35, 80)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 33, 33, 192)	138240	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 33, 33, 192)	576	conv2d_4[0][0]
activation_4 (Activation)	(None, 33, 33, 192)	0	batch_normalization_4[0][0]
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][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 64)	192	conv2d_8[0][0]
activation_8 (Activation)	(None, 16, 16, 64)	0	batch_normalization_8[0][0]

conv2d_6 (Conv2D)	(None, 16, 16, 48)	9216	max_pooling2d_1[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 96)	55296	activation_8[0][0]
batch_normalization_6 (BatchNor	(None, 16, 16, 48)	144	conv2d_6[0][0]
batch_normalization_9 (BatchNor	(None, 16, 16, 96)	288	conv2d_9[0][0]
activation_6 (Activation)	(None, 16, 16, 48)	0	batch_normalization_6[0][0]
activation_9 (Activation)	(None, 16, 16, 96)	0	batch_normalization_9[0][0]
average_pooling2d (AveragePooli	(None, 16, 16, 192)	0	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 16, 16, 64)	12288	max_pooling2d_1[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 64)	76800	activation_6[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 96)	82944	activation_9[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 32)	6144	average_pooling2d[0][0]
batch_normalization_5 (BatchNor	(None, 16, 16, 64)	192	conv2d_5[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 64)	192	conv2d_7[0][0]
batch_normalization_10 (BatchNo	(None, 16, 16, 96)	288	conv2d_10[0][0]
batch_normalization_11 (BatchNo	(None, 16, 16, 32)	96	conv2d_11[0][0]
activation_5 (Activation)	(None, 16, 16, 64)	0	batch_normalization_5[0][0]
activation_7 (Activation)	(None, 16, 16, 64)	0	batch_normalization_7[0][0]
activation_10 (Activation)	(None, 16, 16, 96)	0	batch_normalization_10[0][0]
activation_11 (Activation)	(None, 16, 16, 32)	0	batch_normalization_11[0][0]
mixed0 (Concatenate)	(None, 16, 16, 256)	0	activation_5[0][0] activation_7[0][0] activation_10[0][0] activation_11[0][0]
conv2d_15 (Conv2D)	(None, 16, 16, 64)	16384	mixed0[0][0]

batch_normalization_15 (BatchNo	(None, 16, 16, 64)	192	conv2d_15[0][0]
activation_15 (Activation)	(None, 16, 16, 64)	0	batch_normalization_15[0][0]
conv2d_13 (Conv2D)	(None, 16, 16, 48)	12288	mixed0[0][0]
conv2d_16 (Conv2D)	(None, 16, 16, 96)	55296	activation_15[0][0]
batch_normalization_13 (BatchNo	(None, 16, 16, 48)	144	conv2d_13[0][0]
batch_normalization_16 (BatchNo	(None, 16, 16, 96)	288	conv2d_16[0][0]
activation_13 (Activation)	(None, 16, 16, 48)	0	batch_normalization_13[0][0]
activation_16 (Activation)	(None, 16, 16, 96)	0	batch_normalization_16[0][0]
average_pooling2d_1 (AveragePoo	(None, 16, 16, 256)	0	mixed0[0][0]
conv2d_12 (Conv2D)	(None, 16, 16, 64)	16384	mixed0[0][0]
conv2d_14 (Conv2D)	(None, 16, 16, 64)	76800	activation_13[0][0]
conv2d_17 (Conv2D)	(None, 16, 16, 96)	82944	activation_16[0][0]
conv2d_18 (Conv2D)	(None, 16, 16, 64)	16384	average_pooling2d_1[0][0]
batch_normalization_12 (BatchNo	(None, 16, 16, 64)	192	conv2d_12[0][0]
batch_normalization_14 (BatchNo	(None, 16, 16, 64)	192	conv2d_14[0][0]
batch_normalization_17 (BatchNo	(None, 16, 16, 96)	288	conv2d_17[0][0]
batch_normalization_18 (BatchNo	(None, 16, 16, 64)	192	conv2d_18[0][0]
activation_12 (Activation)	(None, 16, 16, 64)	0	batch_normalization_12[0][0]
activation_14 (Activation)	(None, 16, 16, 64)	0	batch_normalization_14[0][0]
activation_17 (Activation)	(None, 16, 16, 96)	0	batch_normalization_17[0][0]
activation_18 (Activation)	(None, 16, 16, 64)	0	batch_normalization_18[0][0]
mixed1 (Concatenate)	(None, 16, 16, 288)	0	activation_12[0][0] activation_14[0][0]

			activation_17[0][0] activation_18[0][0]
conv2d_22 (Conv2D)	(None, 16, 16, 64)	18432	mixed1[0][0]
batch_normalization_22 (BatchNo	(None, 16, 16, 64)	192	conv2d_22[0][0]
activation_22 (Activation)	(None, 16, 16, 64)	0	batch_normalization_22[0][0]
conv2d_20 (Conv2D)	(None, 16, 16, 48)	13824	mixed1[0][0]
conv2d_23 (Conv2D)	(None, 16, 16, 96)	55296	activation_22[0][0]
batch_normalization_20 (BatchNo	(None, 16, 16, 48)	144	conv2d_20[0][0]
batch_normalization_23 (BatchNo	(None, 16, 16, 96)	288	conv2d_23[0][0]
activation_20 (Activation)	(None, 16, 16, 48)	0	batch_normalization_20[0][0]
activation_23 (Activation)	(None, 16, 16, 96)	0	batch_normalization_23[0][0]
average_pooling2d_2 (AveragePoo	(None, 16, 16, 288)	0	mixed1[0][0]
conv2d_19 (Conv2D)	(None, 16, 16, 64)	18432	mixed1[0][0]
conv2d_21 (Conv2D)	(None, 16, 16, 64)	76800	activation_20[0][0]
conv2d_24 (Conv2D)	(None, 16, 16, 96)	82944	activation_23[0][0]
conv2d_25 (Conv2D)	(None, 16, 16, 64)	18432	average_pooling2d_2[0][0]
batch_normalization_19 (BatchNo	(None, 16, 16, 64)	192	conv2d_19[0][0]
batch_normalization_21 (BatchNo	(None, 16, 16, 64)	192	conv2d_21[0][0]
batch_normalization_24 (BatchNo	(None, 16, 16, 96)	288	conv2d_24[0][0]
batch_normalization_25 (BatchNo	(None, 16, 16, 64)	192	conv2d_25[0][0]
activation_19 (Activation)	(None, 16, 16, 64)	0	batch_normalization_19[0][0]
activation_21 (Activation)	(None, 16, 16, 64)	0	batch_normalization_21[0][0]
activation_24 (Activation)	(None, 16, 16, 96)	0	batch_normalization_24[0][0]

activation_25 (Activation)	(None, 16, 16, 64)	0	batch_normalization_25[0][0]
mixed2 (Concatenate)	(None, 16, 16, 288)	0	activation_19[0][0] activation_21[0][0] activation_24[0][0] activation_25[0][0]
conv2d_27 (Conv2D)	(None, 16, 16, 64)	18432	mixed2[0][0]
batch_normalization_27 (BatchNo	(None, 16, 16, 64)	192	conv2d_27[0][0]
activation_27 (Activation)	(None, 16, 16, 64)	0	batch_normalization_27[0][0]
conv2d_28 (Conv2D)	(None, 16, 16, 96)	55296	activation_27[0][0]
batch_normalization_28 (BatchNo	(None, 16, 16, 96)	288	conv2d_28[0][0]
activation_28 (Activation)	(None, 16, 16, 96)	0	batch_normalization_28[0][0]
conv2d_26 (Conv2D)	(None, 7, 7, 384)	995328	mixed2[0][0]
conv2d_29 (Conv2D)	(None, 7, 7, 96)	82944	activation_28[0][0]
batch_normalization_26 (BatchNo	(None, 7, 7, 384)	1152	conv2d_26[0][0]
batch_normalization_29 (BatchNo	(None, 7, 7, 96)	288	conv2d_29[0][0]
activation_26 (Activation)	(None, 7, 7, 384)	0	batch_normalization_26[0][0]
activation_29 (Activation)	(None, 7, 7, 96)	0	batch_normalization_29[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 288)	0	mixed2[0][0]
mixed3 (Concatenate)	(None, 7, 7, 768)	0	activation_26[0][0] activation_29[0][0] max_pooling2d_2[0][0]
conv2d_34 (Conv2D)	(None, 7, 7, 128)	98304	mixed3[0][0]
batch_normalization_34 (BatchNo	(None, 7, 7, 128)	384	conv2d_34[0][0]
activation_34 (Activation)	(None, 7, 7, 128)	0	batch_normalization_34[0][0]
conv2d_35 (Conv2D)	(None, 7, 7, 128)	114688	activation_34[0][0]

batch_normalization_35 (BatchNo	(None, 7, 7, 128)	384	conv2d_35[0][0]
activation_35 (Activation)	(None, 7, 7, 128)	0	batch_normalization_35[0][0]
conv2d_31 (Conv2D)	(None, 7, 7, 128)	98304	mixed3[0][0]
conv2d_36 (Conv2D)	(None, 7, 7, 128)	114688	activation_35[0][0]
batch_normalization_31 (BatchNo	(None, 7, 7, 128)	384	conv2d_31[0][0]
batch_normalization_36 (BatchNo	(None, 7, 7, 128)	384	conv2d_36[0][0]
activation_31 (Activation)	(None, 7, 7, 128)	0	batch_normalization_31[0][0]
activation_36 (Activation)	(None, 7, 7, 128)	0	batch_normalization_36[0][0]
conv2d_32 (Conv2D)	(None, 7, 7, 128)	114688	activation_31[0][0]
conv2d_37 (Conv2D)	(None, 7, 7, 128)	114688	activation_36[0][0]
batch_normalization_32 (BatchNo	(None, 7, 7, 128)	384	conv2d_32[0][0]
batch_normalization_37 (BatchNo	(None, 7, 7, 128)	384	conv2d_37[0][0]
activation_32 (Activation)	(None, 7, 7, 128)	0	batch_normalization_32[0][0]
activation_37 (Activation)	(None, 7, 7, 128)	0	batch_normalization_37[0][0]
average_pooling2d_3 (AveragePoo	(None, 7, 7, 768)	0	mixed3[0][0]
conv2d_30 (Conv2D)	(None, 7, 7, 192)	147456	mixed3[0][0]
conv2d_33 (Conv2D)	(None, 7, 7, 192)	172032	activation_32[0][0]
conv2d_38 (Conv2D)	(None, 7, 7, 192)	172032	activation_37[0][0]
conv2d_39 (Conv2D)	(None, 7, 7, 192)	147456	average_pooling2d_3[0][0]
batch_normalization_30 (BatchNo	(None, 7, 7, 192)	576	conv2d_30[0][0]
batch_normalization_33 (BatchNo	(None, 7, 7, 192)	576	conv2d_33[0][0]
batch_normalization_38 (BatchNo	(None, 7, 7, 192)	576	conv2d_38[0][0]
batch_normalization_39 (BatchNo	(None, 7, 7, 192)	576	conv2d_39[0][0]

activation_30 (Activation)	(None, 7, 7, 192)	0	batch_normalization_30[0][0]
activation_33 (Activation)	(None, 7, 7, 192)	0	batch_normalization_33[0][0]
activation_38 (Activation)	(None, 7, 7, 192)	0	batch_normalization_38[0][0]
activation_39 (Activation)	(None, 7, 7, 192)	0	batch_normalization_39[0][0]
mixed4 (Concatenate)	(None, 7, 7, 768)	0	activation_30[0][0] activation_33[0][0] activation_38[0][0] activation_39[0][0]
conv2d_44 (Conv2D)	(None, 7, 7, 160)	122880	mixed4[0][0]
batch_normalization_44 (BatchNo	(None, 7, 7, 160)	480	conv2d_44[0][0]
activation_44 (Activation)	(None, 7, 7, 160)	0	batch_normalization_44[0][0]
conv2d_45 (Conv2D)	(None, 7, 7, 160)	179200	activation_44[0][0]
batch_normalization_45 (BatchNo	(None, 7, 7, 160)	480	conv2d_45[0][0]
activation_45 (Activation)	(None, 7, 7, 160)	0	batch_normalization_45[0][0]
conv2d_41 (Conv2D)	(None, 7, 7, 160)	122880	mixed4[0][0]
conv2d_46 (Conv2D)	(None, 7, 7, 160)	179200	activation_45[0][0]
batch_normalization_41 (BatchNo	(None, 7, 7, 160)	480	conv2d_41[0][0]
batch_normalization_46 (BatchNo	(None, 7, 7, 160)	480	conv2d_46[0][0]
activation_41 (Activation)	(None, 7, 7, 160)	0	batch_normalization_41[0][0]
activation_46 (Activation)	(None, 7, 7, 160)	0	batch_normalization_46[0][0]
conv2d_42 (Conv2D)	(None, 7, 7, 160)	179200	activation_41[0][0]
conv2d_47 (Conv2D)	(None, 7, 7, 160)	179200	activation_46[0][0]
batch_normalization_42 (BatchNo	(None, 7, 7, 160)	480	conv2d_42[0][0]
batch_normalization_47 (BatchNo	(None, 7, 7, 160)	480	conv2d_47[0][0]

activation_42 (Activation)	(None, 7, 7, 160)	0	batch_normalization_42[0][0]
activation_47 (Activation)	(None, 7, 7, 160)	0	batch_normalization_47[0][0]
average_pooling2d_4 (AveragePoo	(None, 7, 7, 768)	0	mixed4[0][0]
conv2d_40 (Conv2D)	(None, 7, 7, 192)	147456	mixed4[0][0]
conv2d_43 (Conv2D)	(None, 7, 7, 192)	215040	activation_42[0][0]
conv2d_48 (Conv2D)	(None, 7, 7, 192)	215040	activation_47[0][0]
conv2d_49 (Conv2D)	(None, 7, 7, 192)	147456	average_pooling2d_4[0][0]
batch_normalization_40 (BatchNo	(None, 7, 7, 192)	576	conv2d_40[0][0]
batch_normalization_43 (BatchNo	(None, 7, 7, 192)	576	conv2d_43[0][0]
batch_normalization_48 (BatchNo	(None, 7, 7, 192)	576	conv2d_48[0][0]
batch_normalization_49 (BatchNo	(None, 7, 7, 192)	576	conv2d_49[0][0]
activation_40 (Activation)	(None, 7, 7, 192)	0	batch_normalization_40[0][0]
activation_43 (Activation)	(None, 7, 7, 192)	0	batch_normalization_43[0][0]
activation_48 (Activation)	(None, 7, 7, 192)	0	batch_normalization_48[0][0]
activation_49 (Activation)	(None, 7, 7, 192)	0	batch_normalization_49[0][0]
mixed5 (Concatenate)	(None, 7, 7, 768)	0	activation_40[0][0] activation_43[0][0] activation_48[0][0] activation_49[0][0]
conv2d_54 (Conv2D)	(None, 7, 7, 160)	122880	mixed5[0][0]
batch_normalization_54 (BatchNo	(None, 7, 7, 160)	480	conv2d_54[0][0]
activation_54 (Activation)	(None, 7, 7, 160)	0	batch_normalization_54[0][0]
conv2d_55 (Conv2D)	(None, 7, 7, 160)	179200	activation_54[0][0]
batch_normalization_55 (BatchNo	(None, 7, 7, 160)	480	conv2d_55[0][0]

activation_55 (Activation)	(None, 7, 7, 160)	0	batch_normalization_55[0][0]
conv2d_51 (Conv2D)	(None, 7, 7, 160)	122880	mixed5[0][0]
conv2d_56 (Conv2D)	(None, 7, 7, 160)	179200	activation_55[0][0]
batch_normalization_51 (BatchNo	(None, 7, 7, 160)	480	conv2d_51[0][0]
batch_normalization_56 (BatchNo	(None, 7, 7, 160)	480	conv2d_56[0][0]
activation_51 (Activation)	(None, 7, 7, 160)	0	batch_normalization_51[0][0]
activation_56 (Activation)	(None, 7, 7, 160)	0	batch_normalization_56[0][0]
conv2d_52 (Conv2D)	(None, 7, 7, 160)	179200	activation_51[0][0]
conv2d_57 (Conv2D)	(None, 7, 7, 160)	179200	activation_56[0][0]
batch_normalization_52 (BatchNo	(None, 7, 7, 160)	480	conv2d_52[0][0]
batch_normalization_57 (BatchNo	(None, 7, 7, 160)	480	conv2d_57[0][0]
activation_52 (Activation)	(None, 7, 7, 160)	0	batch_normalization_52[0][0]
activation_57 (Activation)	(None, 7, 7, 160)	0	batch_normalization_57[0][0]
average_pooling2d_5 (AveragePoo	(None, 7, 7, 768)	0	mixed5[0][0]
conv2d_50 (Conv2D)	(None, 7, 7, 192)	147456	mixed5[0][0]
conv2d_53 (Conv2D)	(None, 7, 7, 192)	215040	activation_52[0][0]
conv2d_58 (Conv2D)	(None, 7, 7, 192)	215040	activation_57[0][0]
conv2d_59 (Conv2D)	(None, 7, 7, 192)	147456	average_pooling2d_5[0][0]
batch_normalization_50 (BatchNo	(None, 7, 7, 192)	576	conv2d_50[0][0]
batch_normalization_53 (BatchNo	(None, 7, 7, 192)	576	conv2d_53[0][0]
batch_normalization_58 (BatchNo	(None, 7, 7, 192)	576	conv2d_58[0][0]
batch_normalization_59 (BatchNo	(None, 7, 7, 192)	576	conv2d_59[0][0]

activation_50 (Activation)	(None, 7, 7, 192)	0	batch_normalization_50[0][0]
activation_53 (Activation)	(None, 7, 7, 192)	0	batch_normalization_53[0][0]
activation_58 (Activation)	(None, 7, 7, 192)	0	batch_normalization_58[0][0]
activation_59 (Activation)	(None, 7, 7, 192)	0	batch_normalization_59[0][0]
mixed6 (Concatenate)	(None, 7, 7, 768)	0	activation_50[0][0] activation_53[0][0] activation_58[0][0] activation_59[0][0]
conv2d_64 (Conv2D)	(None, 7, 7, 192)	147456	mixed6[0][0]
batch_normalization_64 (BatchNo	(None, 7, 7, 192)	576	conv2d_64[0][0]
activation_64 (Activation)	(None, 7, 7, 192)	0	batch_normalization_64[0][0]
conv2d_65 (Conv2D)	(None, 7, 7, 192)	258048	activation_64[0][0]
batch_normalization_65 (BatchNo	(None, 7, 7, 192)	576	conv2d_65[0][0]
activation_65 (Activation)	(None, 7, 7, 192)	0	batch_normalization_65[0][0]
conv2d_61 (Conv2D)	(None, 7, 7, 192)	147456	mixed6[0][0]
conv2d_66 (Conv2D)	(None, 7, 7, 192)	258048	activation_65[0][0]
batch_normalization_61 (BatchNo	(None, 7, 7, 192)	576	conv2d_61[0][0]
batch_normalization_66 (BatchNo	(None, 7, 7, 192)	576	conv2d_66[0][0]
activation_61 (Activation)	(None, 7, 7, 192)	0	batch_normalization_61[0][0]
activation_66 (Activation)	(None, 7, 7, 192)	0	batch_normalization_66[0][0]
conv2d_62 (Conv2D)	(None, 7, 7, 192)	258048	activation_61[0][0]
conv2d_67 (Conv2D)	(None, 7, 7, 192)	258048	activation_66[0][0]
batch_normalization_62 (BatchNo	(None, 7, 7, 192)	576	conv2d_62[0][0]
batch_normalization_67 (BatchNo	(None, 7, 7, 192)	576	conv2d_67[0][0]

activation_62 (Activation)	(None, 7, 7, 192)	0	batch_normalization_62[0][0]
activation_67 (Activation)	(None, 7, 7, 192)	0	batch_normalization_67[0][0]
average_pooling2d_6 (AveragePoo	(None, 7, 7, 768)	0	mixed6[0][0]
conv2d_60 (Conv2D)	(None, 7, 7, 192)	147456	mixed6[0][0]
conv2d_63 (Conv2D)	(None, 7, 7, 192)	258048	activation_62[0][0]
conv2d_68 (Conv2D)	(None, 7, 7, 192)	258048	activation_67[0][0]
conv2d_69 (Conv2D)	(None, 7, 7, 192)	147456	average_pooling2d_6[0][0]
batch_normalization_60 (BatchNo	(None, 7, 7, 192)	576	conv2d_60[0][0]
batch_normalization_63 (BatchNo	(None, 7, 7, 192)	576	conv2d_63[0][0]
batch_normalization_68 (BatchNo	(None, 7, 7, 192)	576	conv2d_68[0][0]
batch_normalization_69 (BatchNo	(None, 7, 7, 192)	576	conv2d_69[0][0]
activation_60 (Activation)	(None, 7, 7, 192)	0	batch_normalization_60[0][0]
activation_63 (Activation)	(None, 7, 7, 192)	0	batch_normalization_63[0][0]
activation_68 (Activation)	(None, 7, 7, 192)	0	batch_normalization_68[0][0]
activation_69 (Activation)	(None, 7, 7, 192)	0	batch_normalization_69[0][0]
mixed7 (Concatenate)	(None, 7, 7, 768)	0	activation_60[0][0] activation_63[0][0] activation_68[0][0] activation_69[0][0]
flatten (Flatten)	(None, 37632)	0	mixed7[0][0]
dense (Dense)	(None, 1024)	38536192	flatten[0][0]
dropout (Dropout)	(None, 1024)	0	dense[0][0]
dense_1 (Dense)	(None, 1)	1025	dropout[0][0]

Total params: 47,512,481
Trainable params: 38,537,217

```
Non-trainable params: 8,975,264
_________________________________________________________________
```

We will train our model with the `binary_crossentropy` loss. We will use the `RMSprop` optimizer with a learning rate of `0.0001`. During training, we will want to monitor classification accuracy.

In [11]:
```
model.compile(loss = 'binary_crossentropy',
              optimizer = RMSprop(lr = 1e-4),
              metrics = ['acc'])
```

3. Model Training

In [12]:
```
# A callback is a set of functions to be applied at given stages of the training procedure.
# We can use callbacks to get a view on internal states and statistics of the model during training.
# We can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential
# or Model classes.
# The relevant methods of the callbacks will then be called at each stage of the training.

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc') > 0.98):
            print("\nReached target accuracy so cancelling training!")
            self.model.stop_training = True

callbacks = myCallback()
```

In [13]:

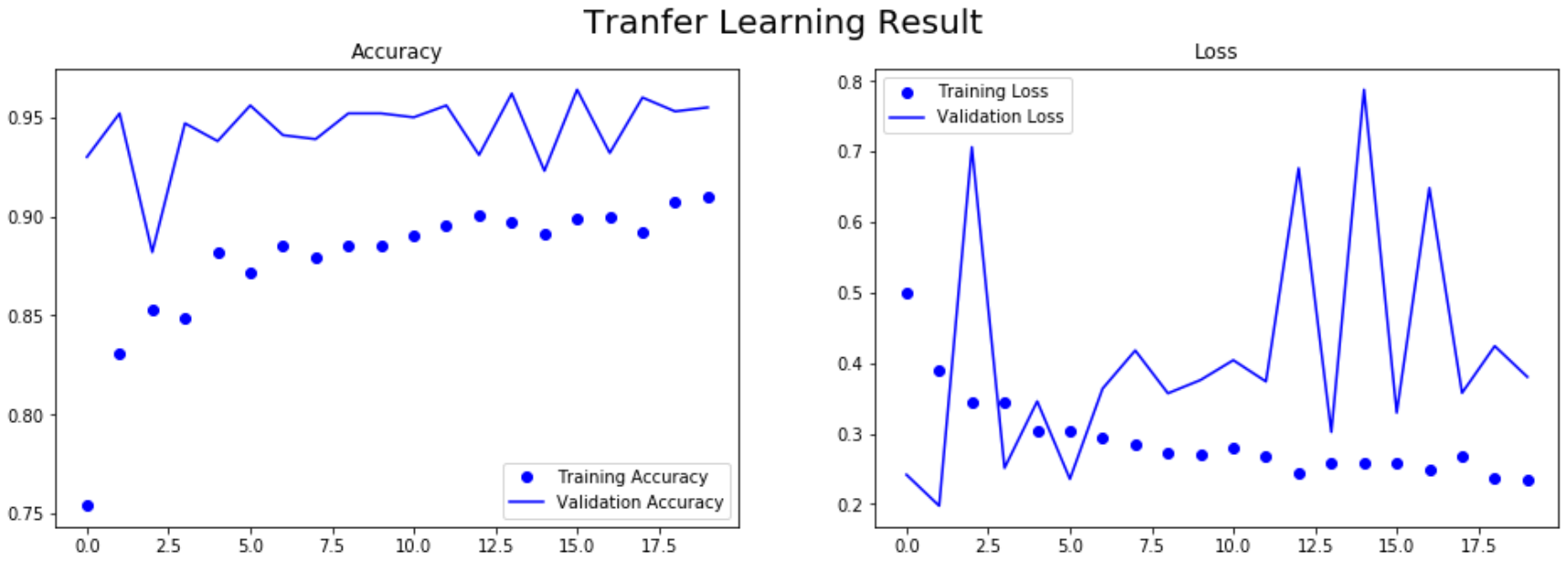
```
# create function to plot the training progress

def plot_progress(history, title):
    """
    Plot the loss and accuracy of training and validation data, as functions of epochs

    Arguments:
    history: the History object returned by model.fit_generator()
    title: the title of the graph
    """

    train_loss = history.history['loss']
    train_acc = history.history['acc']
    val_loss = history.history['val_loss']
    val_acc = history.history['val_acc']
    epochs = range(len(train_acc))

    fig, axes = plt.subplots(1, 2)
    fig.suptitle(title, fontsize = 20)
    fig.set_size_inches(16, 5)
    axes[0].plot(epochs, train_acc, 'bo', label = 'Training Accuracy')
    axes[0].plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
    axes[0].set_title('Accuracy')
    axes[0].legend()
    axes[1].plot(epochs, train_loss, 'bo', label = 'Training Loss')
    axes[1].plot(epochs, val_loss, 'b', label = 'Validation Loss')
    axes[1].set_title('Loss')
    axes[1].legend()
    plt.show()
```

In [14]:
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch = 100, # number of images = batch_size * steps
    epochs = 20,
    verbose = 1,
    validation_data = dev_generator,
    validation_steps = 50, # number of images = batch_size * steps
    callbacks = [callbacks])
```

```
Epoch 1/20
100/100 [==============================] - 333s 3s/step - loss: 0.4997 - acc: 0.7540 - val_loss: 0.2418 - val_acc: 0.9300
Epoch 2/20
100/100 [==============================] - 336s 3s/step - loss: 0.3901 - acc: 0.8310 - val_loss: 0.1977 - val_acc: 0.9520
Epoch 3/20
100/100 [==============================] - 337s 3s/step - loss: 0.3452 - acc: 0.8530 - val_loss: 0.7065 - val_acc: 0.8820
Epoch 4/20
100/100 [==============================] - 334s 3s/step - loss: 0.3446 - acc: 0.8485 - val_loss: 0.2513 - val_acc: 0.9470
Epoch 5/20
100/100 [==============================] - 343s 3s/step - loss: 0.3040 - acc: 0.8815 - val_loss: 0.3460 - val_acc: 0.9380
Epoch 6/20
100/100 [==============================] - 343s 3s/step - loss: 0.3044 - acc: 0.8720 - val_loss: 0.2357 - val_acc: 0.9560
Epoch 7/20
100/100 [==============================] - 338s 3s/step - loss: 0.2954 - acc: 0.8855 - val_loss: 0.3639 - val_acc: 0.9410
Epoch 8/20
100/100 [==============================] - 337s 3s/step - loss: 0.2840 - acc: 0.8790 - val_loss: 0.4181 - val_acc: 0.9390
Epoch 9/20
100/100 [==============================] - 344s 3s/step - loss: 0.2737 - acc: 0.8855 - val_loss: 0.3573 - val_acc: 0.9520
Epoch 10/20
100/100 [==============================] - 347s 3s/step - loss: 0.2711 - acc: 0.8855 - val_loss: 0.3762 - val_acc: 0.9520
Epoch 11/20
100/100 [==============================] - 345s 3s/step - loss: 0.2789 - acc: 0.8905 - val_loss: 0.4043 - val_acc: 0.9500
Epoch 12/20
100/100 [==============================] - 352s 4s/step - loss: 0.2689 - acc: 0.8955 - val_loss: 0.3740 - val_acc: 0.9560
Epoch 13/20
100/100 [==============================] - 352s 4s/step - loss: 0.2439 - acc: 0.9005 - val_loss: 0.6766 - val_acc: 0.9310
Epoch 14/20
100/100 [==============================] - 354s 4s/step - loss: 0.2576 - acc: 0.8970 - val_loss: 0.3023 - val_acc: 0.9620
Epoch 15/20
100/100 [==============================] - 350s 3s/step - loss: 0.2581 - acc: 0.8910 - val_loss: 0.7881 - val_acc: 0.9230
Epoch 16/20
100/100 [==============================] - 343s 3s/step - loss: 0.2584 - acc: 0.8985 - val_loss: 0.3297 - val_acc: 0.9640
Epoch 17/20
100/100 [==============================] - 340s 3s/step - loss: 0.2488 - acc: 0.8995 - val_loss: 0.6486 - val_acc: 0.9320
Epoch 18/20
100/100 [==============================] - 328s 3s/step - loss: 0.2673 - acc: 0.8920 - val_loss: 0.3578 - val_acc: 0.9600
Epoch 19/20
100/100 [==============================] - 327s 3s/step - loss: 0.2365 - acc: 0.9075 - val_loss: 0.4243 - val_acc: 0.9530
Epoch 20/20
100/100 [==============================] - 327s 3s/step - loss: 0.2356 - acc: 0.9100 - val_loss: 0.3805 - val_acc: 0.9550
```

Plot the training progress

In [15]:
```
plot_progress(history, 'Tranfer Learning Result')
```



4. Model Testing

Let's now take a look at actually running a prediction using the model. This code will allow us to choose 1 or more files from our file system, it will then upload them, and run them through the model, giving an indication of whether the object is a cat or a dog.

Our images are downloaded from [https://pixabay.com/ (https://pixabay.com/)](https://pixabay.com/)

In [16]:

```
test_path = 'test/' # the folder that saves the test images
test_files = [os.path.join(test_path, fname) for fname in os.listdir(test_path)]

for file in test_files:
    img = image.load_img(file, target_size = (150, 150))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis = 0)
    x /= 255

    # predict
    classes = model.predict(x)

    # show the figure with prediction
    image_cur = plt.imread(file)
    plt.figure(figsize = (5, 5))
    plt.imshow(image_cur)
    plt.axis('off')
    if classes[0] > 0.5:
        plt.title(file.split('/')[-1] + ": dog")
    else:
        plt.title(file.split('/')[-1] + ": cat")
```

cat-1.jpg: cat

cat-2.jpg: cat

dog-1.jpg: dog

dog-2.jpg: dog