Below is code with a link to a happy or sad dataset which contains 80 images, 40 happy and 40 sad. Create a convolutional neural network that trains to 100% accuracy on these images, which cancels training upon hitting training accuracy of >.999

Hint -- it will work best with 3 convolutional layers.

In [1]:

```
import tensorflow as tf
import os
import zipfile
from os import path, getcwd, chdir

# DO NOT CHANGE THE LINE BELOW. If you are developing in a local
# environment, then grab happy-or-sad.zip from the Coursera Jupyter Notebook
# and place it inside a local folder and edit the path to that location
path = f"{getcwd()}/../tmp2/happy-or-sad.zip"

zip_ref = zipfile.ZipFile(path, 'r')
zip_ref.extractall("/tmp/h-or-s")
zip_ref.close()
```

In [2]:

```
# GRADED FUNCTION: train_happy_sad_model
def train_happy_sad_model():
    # Please write your code only where you are indicated.
    # please do not remove # model fitting inline comments.

    DESIRED_ACCURACY = 0.999

    class myCallback(tf.keras.callbacks.Callback):
         def on_epoch_end(self, epoch, logs={}):
                if(logs.get('acc') > DESIRED_ACCURACY):
                    print("Reached 100% Accuracy!")
                    self.model.stop_training=True

    callbacks = myCallback()

    # This Code Block should Define and Compile the Model. Please assume the images are 150 X 150
in your implementation.
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    from tensorflow.keras.optimizers import RMSprop

    model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy', metrics=['acc'])


    # This code block should create an instance of an ImageDataGenerator called train_datagen
    # And a train_generator by calling train_datagen.flow_from_directory

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    train_datagen = ImageDataGenerator(rescale=1./255)

    # Please use a target_size of 150 X 150.
    train_generator = train_datagen.flow_from_directory(
        "/tmp/h-or-s",
        target_size=(150, 150),
        batch_size=10,
        class_mode='binary'

    )
    # Expected output: 'Found 80 images belonging to 2 classes'

    # This code block should call model.fit_generator and train for
    # a number of epochs.
    # model fitting
    history = model.fit_generator(
        train_generator,
        steps_per_epoch=2,
        epochs=15,
        callbacks=[callbacks]

    )
    # model fitting
    return history.history['acc'][-1]
```

In [3]:

```
# The Expected output: "Reached 99.9% accuracy so cancelling training!""
train_happy_sad_model()
```

```
WARNING: Logging before flag parsing goes to stderr.
W1006 00:06:36.611587 140509657356096 deprecation.py:506] From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/init_ops.py:1251: calling VarianceScaling.__init__ (from
tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
W1006 00:06:36.960245 140509657356096 deprecation.py:323] From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Found 80 images belonging to 2 classes.
Epoch 1/15
2/2 [==============================] - 4s 2s/step - loss: 2.1541 - acc: 0.5500
Epoch 2/15
2/2 [==============================] - 0s 8ms/step - loss: 1.4260 - acc: 0.5000
Epoch 3/15
2/2 [==============================] - 0s 37ms/step - loss: 0.7200 - acc: 0.5000
Epoch 4/15
2/2 [==============================] - 0s 8ms/step - loss: 0.6782 - acc: 0.5500
Epoch 5/15
2/2 [==============================] - 0s 52ms/step - loss: 0.6548 - acc: 0.7500
Epoch 6/15
2/2 [==============================] - 0s 48ms/step - loss: 0.7028 - acc: 0.4500
Epoch 7/15
2/2 [==============================] - 0s 51ms/step - loss: 0.6002 - acc: 0.7500
Epoch 8/15
2/2 [==============================] - 0s 47ms/step - loss: 0.5264 - acc: 0.7500
Epoch 9/15
2/2 [==============================] - 0s 88ms/step - loss: 0.7714 - acc: 0.5500
Epoch 10/15
2/2 [==============================] - 0s 50ms/step - loss: 0.4681 - acc: 0.7000
Epoch 11/15
2/2 [==============================] - 0s 48ms/step - loss: 0.4347 - acc: 0.8500
Epoch 12/15
2/2 [==============================] - 0s 12ms/step - loss: 0.5111 - acc: 0.7000
Epoch 13/15
2/2 [==============================] - 0s 86ms/step - loss: 0.2029 - acc: 0.8500
Epoch 14/15
2/2 [==============================] - 0s 49ms/step - loss: 0.2018 - acc: 0.9000
Epoch 15/15
1/2 [==============>...............] - ETA: 0s - loss: 0.0731 - acc: 1.0000Reached 100% Accuracy!
2/2 [==============================] - 0s 13ms/step - loss: 0.0460 - acc: 1.0000
```

Out[3]:

```
1.0
```

In [ ]:

```
# Now click the 'Submit Assignment' button above.
# Once that is complete, please run the following two cells to save your work and close the notebo
ok
```

In [ ]:

```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();
```

In [ ]:

```
%%javascript
IPython.notebook.session.delete();
window.onbeforeunload = null
setTimeout(function() { window.close(); }, 1000);
```