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 [29]:

```
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 [30]:

```
GRADED FUNCTION: train_happy_sad_model
f 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

 # YOUR CODE STARTS HERE
  class myCallback(tf.keras.callbacks.Callback):
      def on_epoch_end(self, epoch, logs={}):
          if(logs.get('acc')>0.999):
              print("\nReached 99.9% accuracy so cancelling training!")
              self.model.stop_training = True
  # YOUR CODE ENDS HERE

  callbacks = myCallback()

  # This Code Block should Define and Compile the Model. Please assume the images a
  model = tf.keras.models.Sequential([
  # Note the input shape is the desired size of the image 300x300 with 3 bytes colo
  # This is the first convolution
      tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)
      tf.keras.layers.MaxPooling2D(2, 2),
  # The second convolution
      tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),
  # The third convolution
      tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),
  # Flatten the results to feed into a DNN
      tf.keras.layers.Flatten(),
  # 512 neuron hidden layer
      tf.keras.layers.Dense(512, activation='relu'),
  # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('ho
      tf.keras.layers.Dense(1, activation='sigmoid')
  ])

  from tensorflow.keras.optimizers import RMSprop

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


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

  from tensorflow.keras.preprocessing.image import ImageDataGenerator

  train_datagen = ImageDataGenerator(rescale=1/255)# Your Code Here

  # Please use a target_size of 150 X 150.

  train_generator = train_datagen.flow_from_directory(
      "/tmp/h-or-s",  # This is the source directory for training images
      target_size=(300, 300),  # All images will be resized to 150x150
      batch_size=128,
      # Since we use binary_crossentropy loss, we need binary labels
      class_mode='binary')
```

```
    # Your Code Here)
# 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=8,
  epochs=15,
  verbose=1)
      # Your Code Here)
# model fitting
return history.history['acc'][-1]
```

In [31]:

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

```
Found 80 images belonging to 2 classes.
Epoch 1/15
8/8 [==============================] - 9s 1s/step - loss: 4.1431 - ac
c: 0.5312
Epoch 2/15
8/8 [==============================] - 5s 675ms/step - loss: 0.3291 -
acc: 0.8375
Epoch 3/15
8/8 [==============================] - 5s 674ms/step - loss: 0.0786 -
acc: 0.9719
Epoch 4/15
8/8 [==============================] - 5s 674ms/step - loss: 0.0431 -
acc: 0.9891
Epoch 5/15
8/8 [==============================] - 5s 662ms/step - loss: 0.0143 -
acc: 1.0000
Epoch 6/15
8/8 [==============================] - 6s 699ms/step - loss: 0.0084 -
acc: 1.0000
Epoch 7/15
8/8 [==============================] - 5s 687ms/step - loss: 0.0026 -
acc: 1.0000
Epoch 8/15
8/8 [==============================] - 5s 687ms/step - loss: 0.0013 -
acc: 1.0000
Epoch 9/15
8/8 [==============================] - 5s 687ms/step - loss: 7.0013e-0
4 - acc: 1.0000
Epoch 10/15
8/8 [==============================] - 5s 687ms/step - loss: 3.4169e-0
4 - acc: 1.0000
Epoch 11/15
8/8 [==============================] - 5s 687ms/step - loss: 1.7692e-0
4 - acc: 1.0000
Epoch 12/15
8/8 [==============================] - 5s 674ms/step - loss: 9.7270e-0
5 - acc: 1.0000
Epoch 13/15
8/8 [==============================] - 5s 674ms/step - loss: 5.3621e-0
5 - acc: 1.0000
Epoch 14/15
8/8 [==============================] - 6s 699ms/step - loss: 2.9916e-0
5 - acc: 1.0000
Epoch 15/15
8/8 [==============================] - 5s 675ms/step - loss: 1.6872e-0
5 - acc: 1.0000
```

Out[31]:

```
1.0
```

In [4]:

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

In [ ]:

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

In [ ]:

```
%%javascript
<!-- Shutdown and close the notebook -->
window.onbeforeunload = null
window.close();
IPython.notebook.session.delete();
```