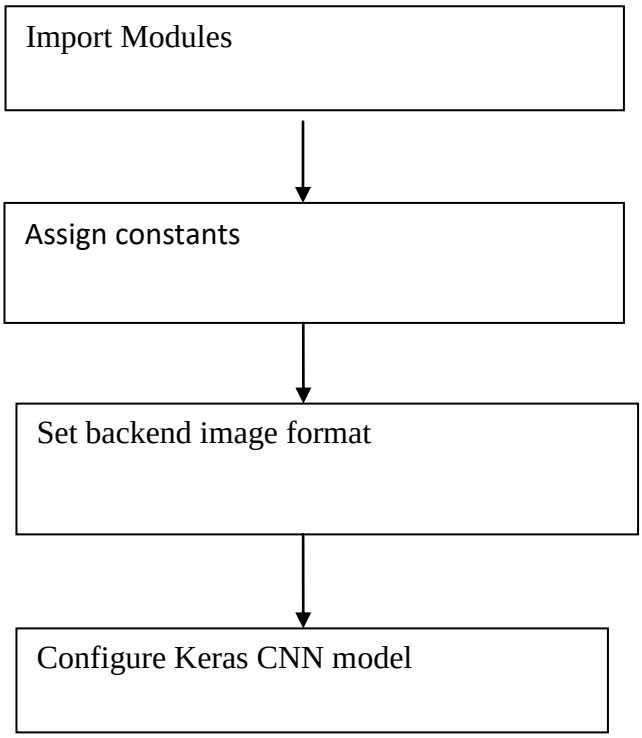
LESSON 25 Image Classification2 **last update June 3, 2021**

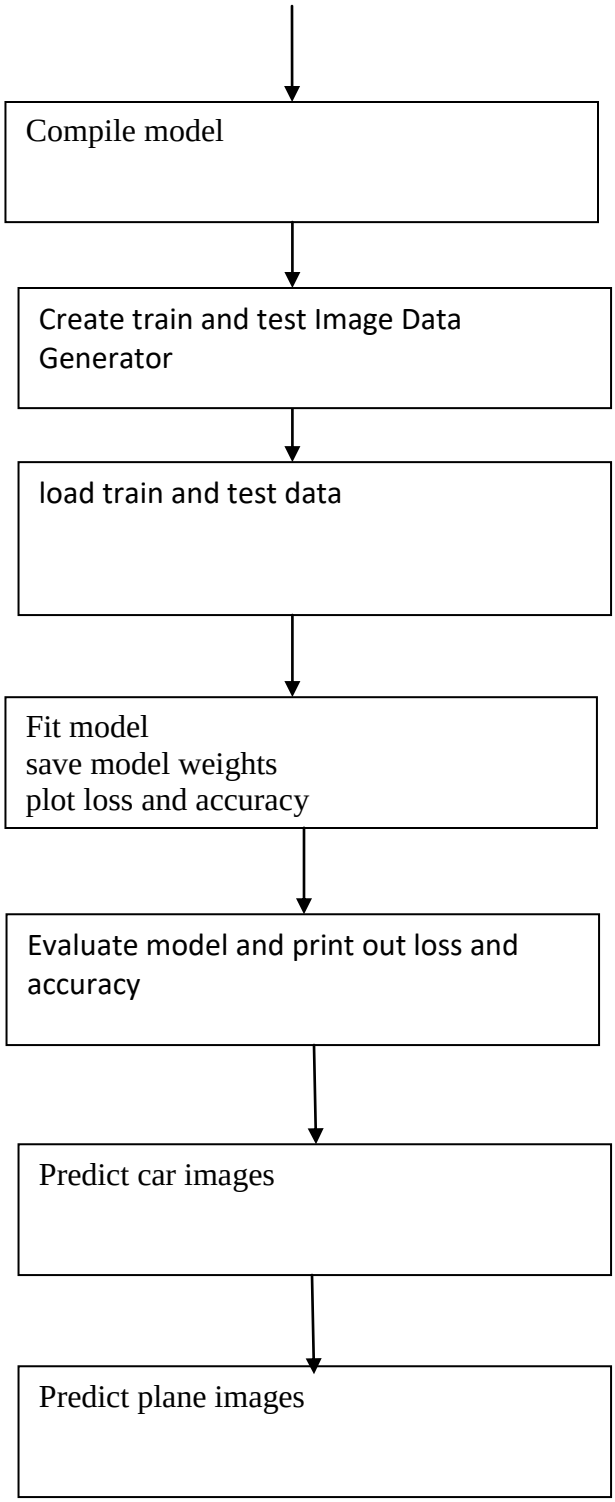
Conventions used in these lessons:

bold - headings, keywords, code
italics - code syntax
underline - important words

In this lesson we will continue our image classification from previous lesson. We will now venture out and classify more image classes. We will classify between cars and planes and between cats and dogs. We have better success classifying cars and planes then cats and dogs because cats and dogs are more similar and more difficult to distinguish between them. Fortunately for us we have many pictures of cars and planes and cats and dogs that we will send to you when you start this lesson. The image classification is very similar to the previous image classification code.

Our Program flow is as follows:





(1) Import Modules

```
# import modules
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
from os import path
```

(2) assign constants

```
img_width, img_height = 224, 224
train_data_dir = 'cars_and_planes/train'
validation_data_dir = 'cars_and_planes/test'
nb_train_samples = 400
nb_validation_samples = 100
epochs = 10
batch_size = 16
```

(3) Set backend Image data format

```
# set backend image format
if K.image_data_format() == 'channels_first':
   input_shape = (3, img_width, img_height)
else:
   input_shape = (img_width, img_height, 3)
```

Keras is a high level implementation relying on backends modules to do all the calculations. For now Keras relies on the following backend modules: **TensorFlow** from Google, **Theano** fromUniversité de Montréal and the **CNTK** from Microsoft.

An image can be stored as a three-dimensional array in memory.

Typically, the image format has one dimension for rows (height), one for columns (width) and one for channels.

If the image is black and white (grayscale), the channels dimension may not be explicitly present, e.g. there is one unsigned integer pixel value for each (row, column) coordinate in the image.

Colored images typically have three channels, for the pixel value at the (row, column) coordinate for the red, green, and blue components.

Deep learning neural networks require that image data be provided as three-dimensional arrays.

This applies even if your image is grayscale. In this case, the additional dimension for the single color channel must be added.

There are two ways to represent the image data as a three dimensional array. The first involves having the channels as the last or third dimension in the array. This is called "**channels last*".*** The second involves having the channels as the first dimension in the array, called "**channels first*".***

- **Channels Last**. Image data is represented in a three-dimensional array where the last channel represents the color channels, e.g. [rows][cols][channels].

- **Channels First**. Image data is represented in a three-dimensional array where the first channel represents the color channels, e.g. [channels][rows][cols].

(4) Configure Keras CNN model

```
model = Sequential()
model.add(Conv2D(32, (2, 2), input_shape = input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Conv2D(32, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))
```

```
model.add(Conv2D(64, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

We have 1 input layer with 32 filters of size 2*2

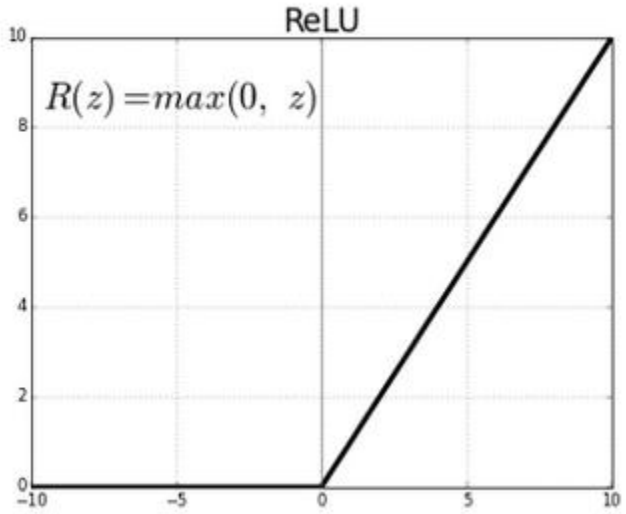
We have a first Conv2D hidden layer of 32 filters of size 2 x2 with activation “relu”

And another Conv2D hidden layer of 32 filters of size 2 x2 with activation “relu”

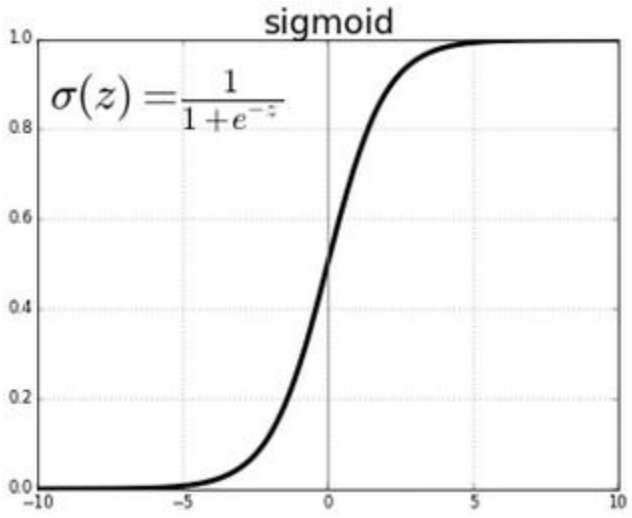
And a third Conv2D hidden layer of 64 filters of size 2 x2 with activation “relu”

After flattening we have a Dense layer of 64 neurons with activation “relu”,

The relu activation function is max(x,0) so no output value goes below , otherwise the output follows the input.



Finally we have a out pt layer of 1 neuron with activation “sigmoid”.



We have lots of filtering for our image classification. Many later will help us the identify the features of each image.

(5) Compile Model

```
model.compile(loss ='binary_crossentropy',
              optimizer ='rmsprop',
             metrics =['accuracy'])
```

We are using 'binary_crossentropy' loss and the rmsprop optimizer and accuracy metrics,

Binary cross entropy is a loss function that are tasks that answer a question with only two choices (yes or no, A or B, 0 or 1, left or right).

The binary cross entropy loss function calculates the loss of an example by computing the following average:

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

where ȳi is the i-th scalar value in the model output, yi is the corresponding target value, and output size is the number of scalar values in the model output.

RMSprop algorithm uses a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.

RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. This means that the learning rate changes over time.

(6) Create train and test Image Data Generator

```
# create train ImageDataGaenerator
train_datagen = ImageDataGenerator(
            rescale = 1. / 255,
             shear_range = 0.2,
              zoom_range = 0.2,
         horizontal_flip = True)

# create TEST ImageDataGaenerator
test_datagen = ImageDataGenerator(rescale = 1. / 255)
```

The **Image Data Generator** is used to generate batches of tensor image data with real-time data augmentation.

Rescale: multiplies data by the value provided after applying all other transformations.
`shear_range`: Shear Intensity (Shear angle in counter-clockwise direction in degrees)
zoom_range: Float or [lower, upper]. Range for random zoom. If a float, `[lower, upper] = [1-zoom_range, 1+zoom_range]`.
horizontal flip: Randomly flip inputs horizontally.

(7) load train and test data

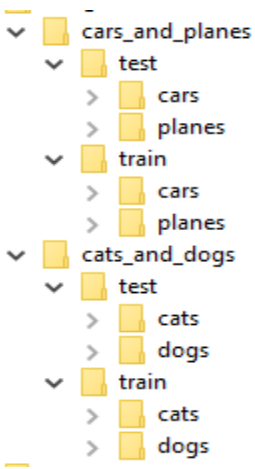
```
# load train image data
train_generator = train_datagen.flow_from_directory(train_data_dir,
                    target_size =(img_width, img_height),
                batch_size = batch_size, class_mode ='binary')

# load train image data
validation_generator = test_datagen.flow_from_directory(
                        validation_data_dir,
                target_size =(img_width, img_height),
        batch_size = batch_size, class_mode ='binary')
```

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

Generates a `tf.data.Dataset` from image files in a directory having the following structure format:

```
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
```

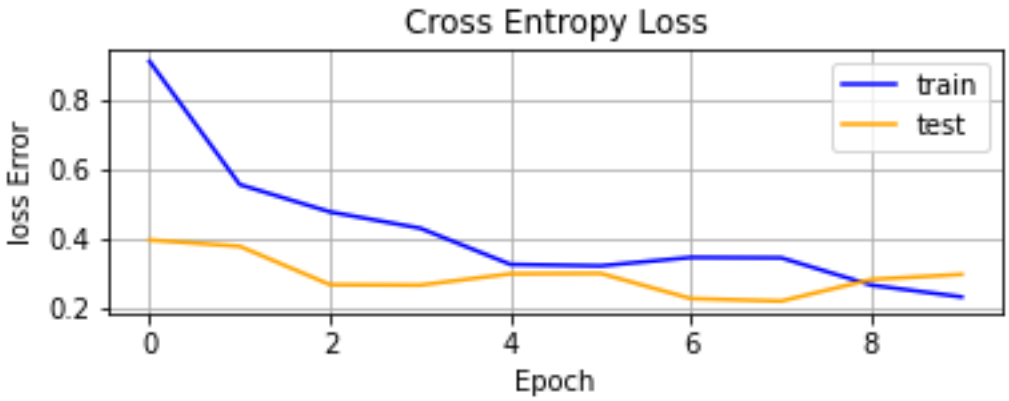
Our file for cars and planes is as follows:

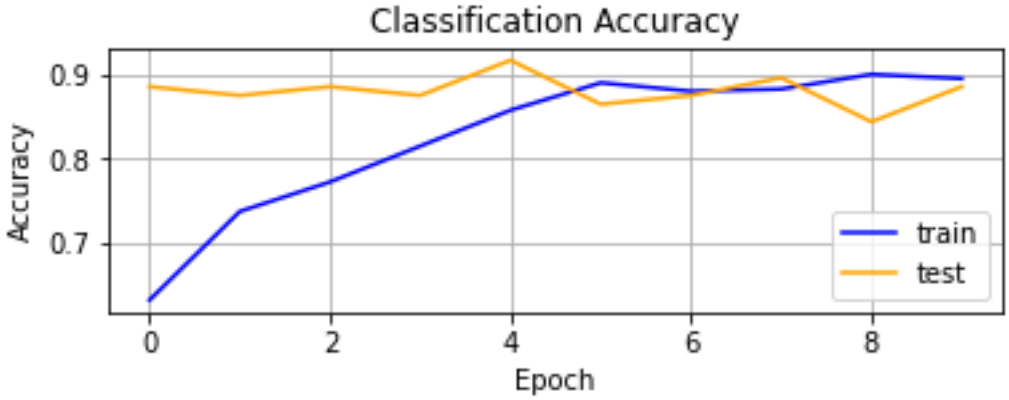


(8) fit model save model results and plot history loss and accuracy

```
# check for weights file
if not path.exists("cars_planes.h5"):
   history = model.fit(train_generator,
      steps_per_epoch = nb_train_samples // batch_size,
      epochs = epochs, validation_data = validation_generator,
      validation_steps = nb_validation_samples // batch_size)

   model.save_weights('cars_planes.h5')

   # plot loss
   plt.subplot(2, 1, 1)
   plt.title('Cross Entropy Loss')
   plt.plot(history.history['loss'], color='blue', label='train')
   plt.plot(history.history['val_loss'], color='orange', label='test')
   plt.xlabel('Epoch')
   plt.ylabel('loss Error')
   plt.legend()
   plt.grid(True)
   plt.show()


   # plot accuracy
   plt.subplot(2, 1, 2)
   plt.title('Classification Accuracy')
   plt.plot(history.history['accuracy'], color='blue', label='train')
   plt.plot(history.history['val_accuracy'], color='orange', label='test')
   plt.xlabel('Epoch')
   plt.ylabel('Accuracy')
   plt.legend()
   plt.grid(True)

   plt.show()

else:
   model.load_weights('cars_planes.h5')
```

In the first round we fit (train) the model using the train generator that generates train and test sets from the image file structure. We then store the module weights if a h5 format file, we also plot **'Cross Entropy Loss'** and **'Classification Accuracy'.** The second time around we just load the module weights from the h5 file.





(9) evaluate and print out loss and accuracy

```
# evaluate
score = model.evaluate(train_generator, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.15304824709892273
Test accuracy: 0.9524999856948853
```

(10) predict car images

```
# predict cars images
fig=plt.figure(figsize=(8, 8))

for i in range(1,21):

   test_image_load = image.load_img \
          ('cars_and_planes/test/cars/'+str(i)+'.jpg', target_size = (img_width, img_height))
    test_image = image.img_to_array(test_image_load)
    test_image = np.expand_dims(test_image, axis = 0)
    test_image /= 255
    result = model.predict(test_image)

    ax=fig.add_subplot(4,5,i)

    #plt.imshow(test_image , cmap ='gray')
    plt.imshow(test_image_load)

    print(result[0][0])
    #print(training_set.class_indices)
    if result[0][0] > .5:
       prediction = 'plane'
    else:
       prediction = 'car'
    print(prediction)
    ax.title.set_text(prediction)

plt.show()
```

(10) predict plane images

```
# predict plane images
fig=plt.figure(figsize=(8, 8))

for i in range(1,21):

    test_image_load = image.load_img \
          ('cars_and_planes/test/planes/'+str(i)+'.jpg', target_size = (img_width, img_height))
    test_image = image.img_to_array(test_image_load)
    test_image = np.expand_dims(test_image, axis = 0)
    test_image /= 255
    result = model.predict(test_image)

    ax=fig.add_subplot(4,5,i)

    #plt.imshow(test_image , cmap ='gray')
    plt.imshow(test_image_load)

    print(result[0][0])
    #print(training_set.class_indices)
    if result[0][0] > .5:
       prediction = 'plane'
    else:
       prediction = 'car'
    print(prediction)
    ax.title.set_text(prediction)
plt.show()
```

Here is the compete program:

```
# carsandplanes.py
# import modules
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K

import numpy as np
import matplotlib.pyplot as plt
#import matplotlib.image as mpimg
#import os.path
from os import path

# assign constants
img_width, img_height = 224, 224
train_data_dir = 'cars_and_planes/train'
validation_data_dir = 'cars_and_planes/test'
nb_train_samples = 400
nb_validation_samples = 100
epochs = 10
batch_size = 16

# set backend image format
if K.image_data_format() == 'channels_first':
   input_shape = (3, img_width, img_height)
else:
   input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (2, 2), input_shape = input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Conv2D(32, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Conv2D(64, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))
```

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))


# compile model
model.compile(loss ='binary_crossentropy',
                optimizer ='rmsprop',
              metrics =['accuracy'])

# create train ImageDataGaenerator
train_datagen = ImageDataGenerator(
            rescale = 1. / 255,
             shear_range = 0.2,
              zoom_range = 0.2,
         horizontal_flip = True)

# create Test ImageDataGaenerator
test_datagen = ImageDataGenerator(rescale = 1. / 255)


# load train image data
train_generator = train_datagen.flow_from_directory(train_data_dir,
                    target_size =(img_width, img_height),
                batch_size = batch_size, class_mode ='binary')

# load train image data
validation_generator = test_datagen.flow_from_directory(
                           validation_data_dir,
                target_size =(img_width, img_height),
        batch_size = batch_size, class_mode ='binary')


# check for weights file
if not path.exists("cars_planes.h5"):
   history = model.fit(train_generator,
      steps_per_epoch = nb_train_samples // batch_size,
      epochs = epochs, validation_data = validation_generator,
      validation_steps = nb_validation_samples // batch_size)
```

```
    model.save_weights('cars_planes.h5')

    # plot loss
    plt.subplot(2, 1, 1)
    plt.title('Cross Entropy Loss')
    plt.plot(history.history['loss'], color='blue', label='train')
    plt.plot(history.history['val_loss'], color='orange', label='test')
    plt.xlabel('Epoch')
    plt.ylabel('loss Error')
    plt.legend()
    plt.grid(True)
    plt.show()

    # plot accuracy
    plt.subplot(2, 1, 2)
    plt.title('Classification Accuracy')
    plt.plot(history.history['accuracy'], color='blue', label='train')
    plt.plot(history.history['val_accuracy'], color='orange', label='test')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.show()

else:
    model.load_weights('cars_planes.h5')

# evaluate
score = model.evaluate(train_generator, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

# predict images
predictions = model.predict(validation_generator)

print(predictions)

i=0
for file in validation_generator.filenames:
    print(file,predictions[i])
    i+=1
```

```
#print images
"""
w=10
h=10
fig=plt.figure(figsize=(8, 8))
columns = 4
rows = 5
for i in range(1, columns*rows +1):
   img = np.random.randint(10, size=(h,w))
   fig.add_subplot(rows, columns, i)
   plt.imshow(img)
plt.show()
"""
# predict cars images
fig=plt.figure(figsize=(8, 8))

for i in range(1,21):

   test_image_load = image.load_img('cars_and_planes/test/cars/'+str(i)+'.jpg',
            target_size = (img_width, img_height))
   test_image = image.img_to_array(test_image_load)
   test_image = np.expand_dims(test_image, axis = 0)
   test_image /= 255
   result = model.predict(test_image)

   ax=fig.add_subplot(4,5,i)

   #plt.imshow(test_image , cmap ='gray')
   plt.imshow(test_image_load)

   print(result[0][0])
   #print(training_set.class_indices)
   if result[0][0] > .5:
      prediction = 'plane'
   else:
      prediction = 'car'
   print(prediction)
   ax.title.set_text(prediction)

plt.show()
```

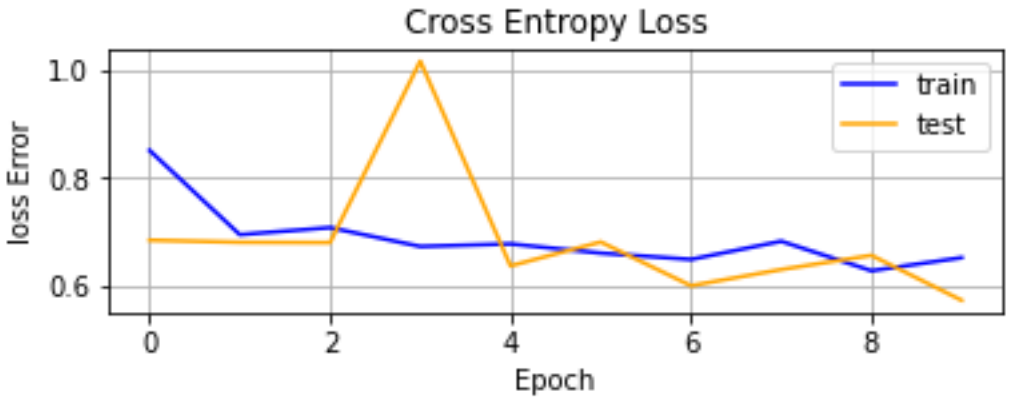
```
# predic plane images
fig=plt.figure(figsize=(8, 8))

for i in range(1,21):

   #test_image_load = image.load_img('v_data/test/cars/12.jpg', target_size =
         (img_width, img_height))
   test_image_load = image.load_img('cars_and_planes/test/planes/'+str(i)+'.jpg',
          target_size = (img_width, img_height))
   test_image = image.img_to_array(test_image_load)
   test_image = np.expand_dims(test_image, axis = 0)
   test_image /= 255
   result = model.predict(test_image)

   ax=fig.add_subplot(4,5,i)

   #plt.imshow(test_image , cmap ='gray')
   plt.imshow(test_image_load)

   print(result[0][0])
   #print(training_set.class_indices)
   if result[0][0] > .5:
      prediction = 'plane'
   else:
      prediction = 'car'
   print(prediction)
   ax.title.set_text(prediction)
plt.show()
```

todo: Type in or copy and paste in to a file called carsandplanes.py and run it.
Note: This program may not work with a theano back end completely.

IMAGE CLASIFICATION2 HOMEWORK Question 1

Use the cats and dogs image files instead of the cars and planes image files,, you should get something like this:

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



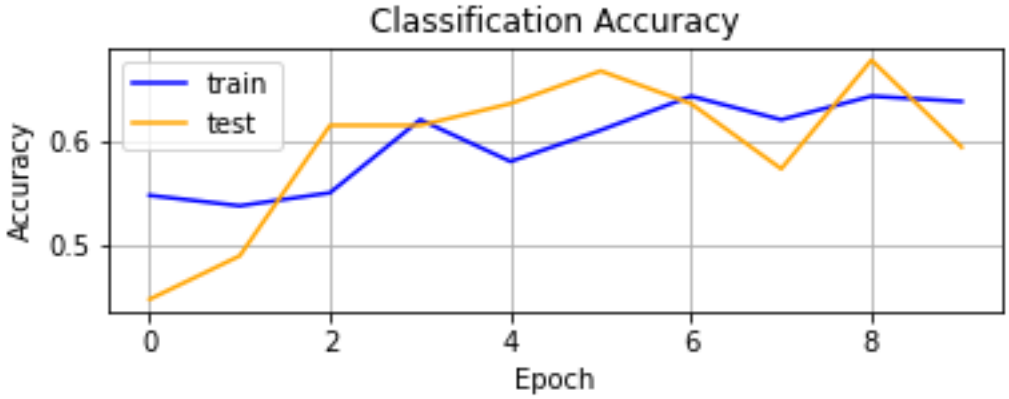


IMAGE CLASIFICATION2 Homework Question 2

Finds some images off the internet like male and female faces and see if you can classify images for male and female faces.

END