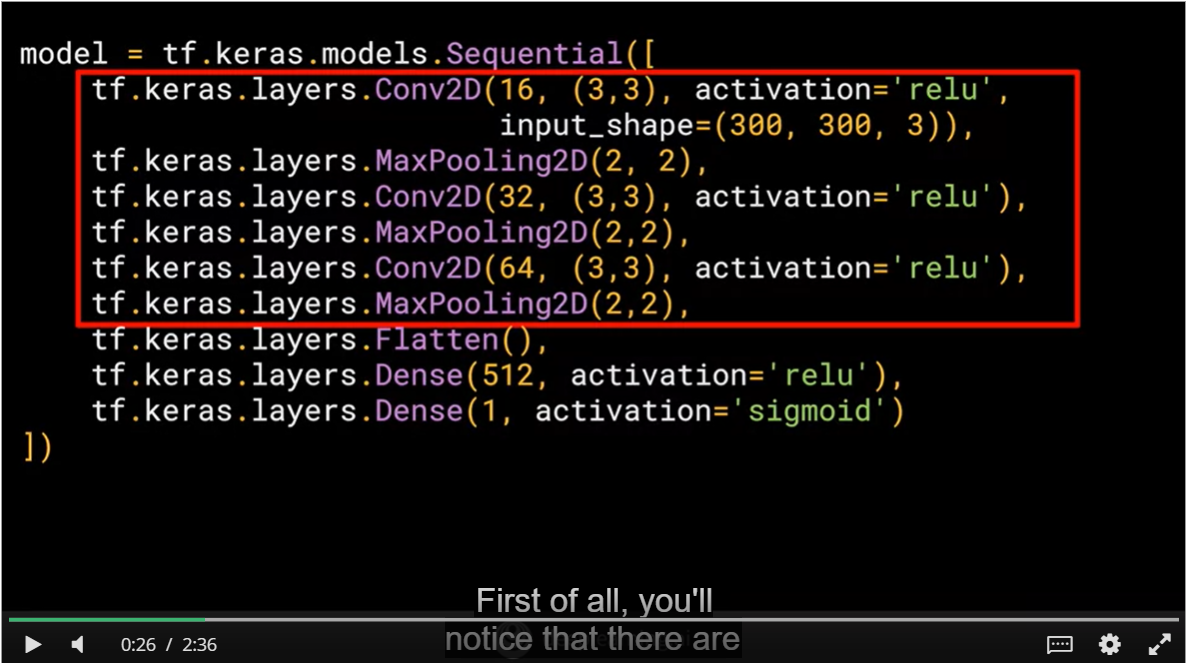
Dealing with real world images using Image generator for a binary classification

Dealing with real world images is not easy as dealing with these simple 26x26 grey scal images due to many reasons: The number of channels ( colors ) , the Object to classify may not be in the center , the image quality , the object may be carried or some part of it is hidden , We may have more than a one object to detect and classify , the orientation

And to make our model flexible with these changes , we have to use Image Generator API In order to cover all these cases and more by injecting our training and validation data and let it handle the generation of new images

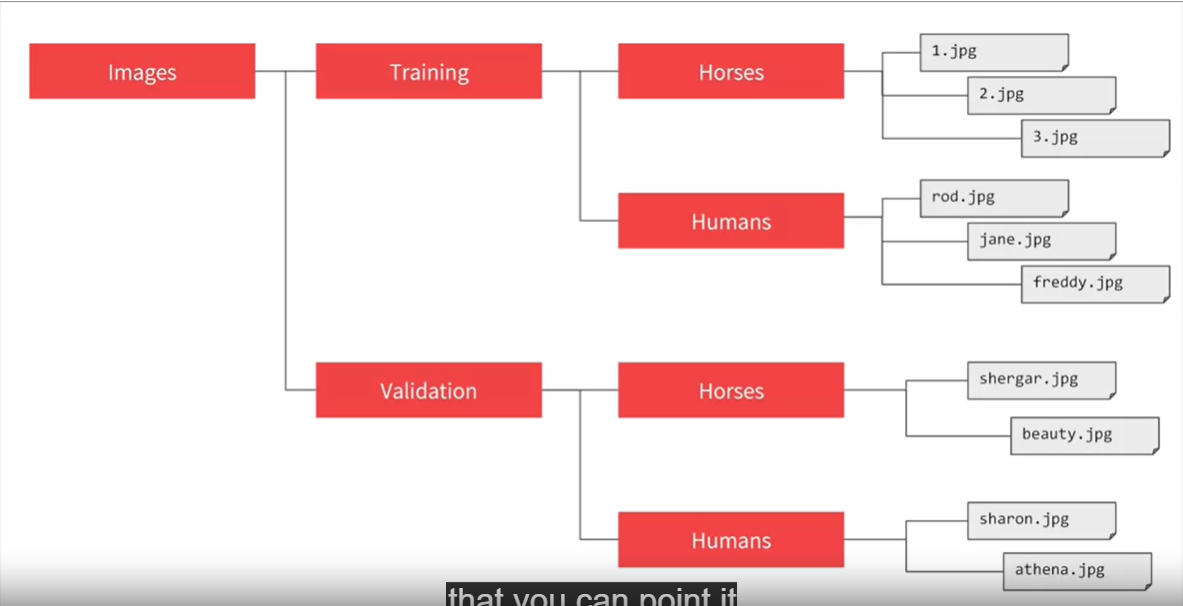
## The model architecture (Horse/Human? classifier):



* + Our images are in shape (300,300,3) which means a RGB images with 300x300 as a dimension
  + Note that we used 3 Convolutions layers (each one is associated with Pooling layer obviously), this is because we are dealing with an images with higher resolutions, So that would be beneficial to reduce the image sizes before doing the flattening and pass the images through the Dense layers
  + Note that the final layer is a dense with a single output using a Sigmoid as an activation function:
    - * Sigmoid is great when it comes to deal with binary classification (Human , Horse ) Since it returns a value between [0 , 1 ] so we can know the predicted class by looking if it’s close to 1 or to 0
      * We can replace the last layer by a Dense with two outputs using SoftMax function as we used to do for classification problem with more than two classes But the first approach is better
* It’s important to note that the filter will be applied only to the intern pixels ( who have a neighbor pixels in all the sides ) ,so the pixels in the border will be removed ( so from 26\*26 we will go to 24\*24 )

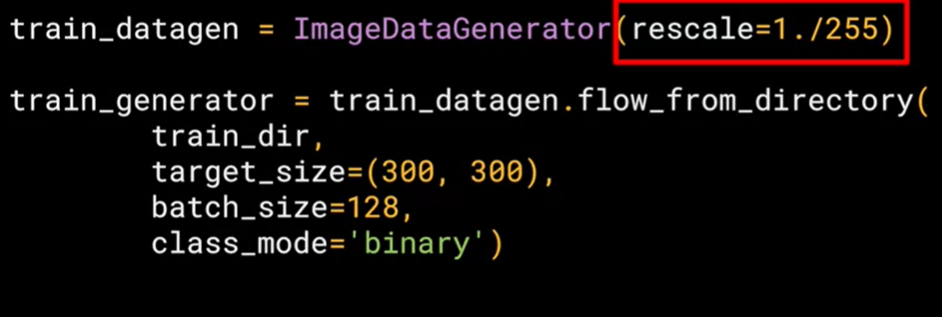
## Understanding Image Generator:

First of all , our data should be set in a sub directories in a very specific way :



* + This is important because Image generator API label will need to know the subdirectory name to label correctly the generated images (Horse or Human )

## Using Image Generator for training:

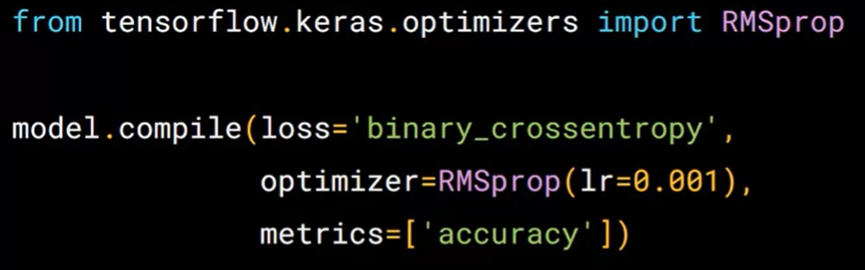


* + **Rescale=1. /255 :**  in order to normalize the data ( having a n array of values between 0 and 1 )
  + **Train\_dir :** the directory which contains the training data
  + **Target\_size = (300,300) :** we will resize all the images to be 300\*300 pixels
  + **Batch\_size=128 :** the number of images will be passed in the same time
  + **Class\_mode=’binary’ :** because we are dealing with binary classification
    - **Note :** 
      * binary will work as “Human vs No Human”
      * we can replace it with ‘sparse’ and it will work as “Human vs Horse”

## Using Image Generator for validation:

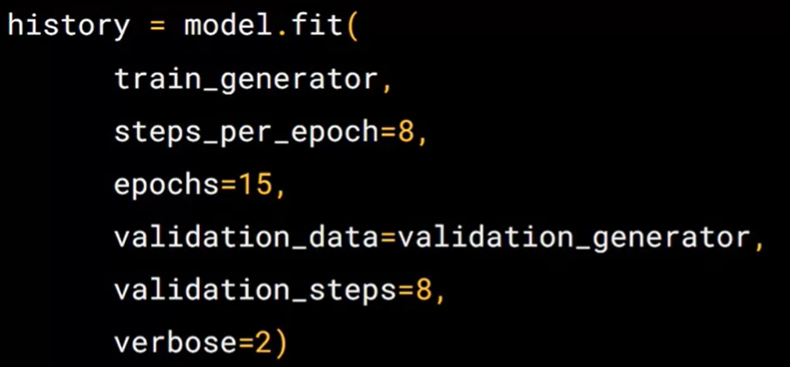
We used exactly the same args , we will just replace **train\_dir** by **validation\_dir**

## Compiling the model



* + **Loss=’binary\_crossentropy’ :** we used this loss function instead of categorical\_crossentropy because we arer dealing now with a binary classification
  + **Optimizer=RMSprop(lr=0.001) :** rmsprop is an extension of the classical gradient descent , lr argument stand for the learning rate

## Fitting the model



* + **train\_generator :** the generated images from imageGenerator
  + **steps\_per\_epoch=8 :** since we have 1024 images and steps per epoch is 8 so batch size = 1024/8 = 128
    - Advantages of using a batch size < number of all samples:
      * It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory.
      * Typically networks train faster with mini-batches. That's because we update the weights after each propagation. In our example we've propagated 11 batches (10 of them had 100 samples and 1 had 50 samples) and after each of them we've updated our network's parameters. If we used all samples during propagation we would make only 1 update for the network's parameter.
    - Disadvantages of using a batch size < number of all samples:
      * The smaller the batch the less accurate the estimate of the gradient will be
  + **Validation\_data :**  the generated images from the image generator we use it to validate our model and test its accuracy
  + **Validation\_steps :** similar to steps\_per\_epoch but this is related to to the validation data
  + **Verbose =2 :** verbose arg specify the amount of the animation to display during the training in the epochs

## Compacting the images ( reducing their resolutions )

* Why?
  + To accelerate the fitting process ( reducing the parameters )
* How ?
  + We set the target\_size=(150, 150) for the ImageGenerator object for testing and validating images
  + For the first Coonv2D layer we specify the arg input\_shape=(150, 150, 3) instead of ( 300,300,3) because we will feed it with the generated image by The ImageGenerator which are 150\*150 RGB pixels
* Conclusion :
  + For our case , The reduction of the dimension size didin’t affect negatively the val accuracy of the testing data and it gave us a quick training which is quite good thing