Training code

STEP1: WE ARE IMPLEMENTING YOLO IN KERAS TO TRY TO UNDERSTAND WHAT WE ARE DOING

Because our business case is quite unique, we might not need all the layers that the original yolo model offers

In [3]:

```
#Let's first load all the libraries we need

from keras.models import Sequential, Model
from keras.layers import Reshape, Activation, Conv2D, Input, MaxPooling2D,
BatchNormalization, Flatten, Dense, Lambda, Dropout
from keras.layers.advanced_activations import LeakyReLU
from keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
from keras.optimizers import SGD, Adam, RMSprop, Adamax
from keras.layers.merge import concatenate
import matplotlib.pyplot as plt
import keras.backend as K
import tensorflow as tf
#import imgaug as ia
from tqdm import tqdm
#from imgaug import augmenters as iaa
import numpy as np
import pandas as pd
import pickle
import os, cv2
#from preprocessing import parse_annotation, BatchGenerator
```

```
/anaconda/envs/py35/lib/python3.5/site-packages/h5py/__init__.py:36: Future
Warning: Conversion of the second argument of issubdtype from `float` to `n
p.floating` is deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [4]:

```
#custom to us are the labels and the image size
LABELS = ['melanoma', 'notmelanoma']

    #I originally tried 200 * 200 and gave me an error - this is because
the input layer is a 32 neuron so we need multiples of 32
#I'm going to use 32*7 = 224

IMAGE_H, IMAGE_W = 192, 192
#Grids are used when you are facing problems with more than one object to
detect and the fact they alow 2 (in the original)
#overlapping bouding boxes. In our case, we have only 1 very well defined
object to detect so we don't need more than 1 grid

#GRID_H,  GRID_W  = 7 , 7

#Let's leave the rest as is

BOX              = 5
CLASS            = len(LABELS)
CLASS_WEIGHTS    = np.ones(CLASS, dtype='float32')
```

```
OBJ_THRESHOLD    = 0.3#0.5
NMS_THRESHOLD    = 0.3#0.45
ANCHORS          = [0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434,
7.88282, 3.52778, 9.77052, 9.16828]

NO_OBJECT_SCALE  = 1.0
OBJECT_SCALE     = 5.0
COORD_SCALE      = 1.0
CLASS_SCALE      = 1.0

BATCH_SIZE       = 16
WARM_UP_BATCHES  = 0
TRUE_BOX_BUFFER  = 50
```

STEP 1.1: LET'S BUILD THE NETWORK

In [6]:

```
# the function to implement the orgnization layer (thanks to
github.com/allanzelener/YAD2K)
def space_to_depth_x2(x):
    return tf.space_to_depth(x, block_size=2)
```

In [8]:

```
input_image = Input(shape=(IMAGE_H, IMAGE_W, 3))
#true_boxes  = Input(shape=(1, 1, 1, TRUE_BOX_BUFFER , 4))

#NOTE ON THE SINTAX = This isn't using Sequential(), it's building a
pipeline of layers applied to
#the input_image. So what this is doing x = fn...(f1(x)). Nested functions.
This is very useful when you need to do skip
#connections or you need to do something a bit more complex to the output,
it isn't a clear sequence

# Layer 1
x = Conv2D(32, (3,3), strides=(1,1), padding='same', name='conv_1',
use_bias=False)(input_image)
x = BatchNormalization(name='norm_1')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Layer 2
x = Conv2D(64, (3,3), strides=(1,1), padding='same', name='conv_2',
use_bias=False)(x)
x = BatchNormalization(name='norm_2')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Layer 3
x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_3',
use_bias=False)(x)
x = BatchNormalization(name='norm_3')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 4
x = Conv2D(64, (1,1), strides=(1,1), padding='same', name='conv_4',
use_bias=False)(x)
x = BatchNormalization(name='norm_4')(x)
x = LeakyReLU(alpha=0.1)(x)
```

```
# Layer 5
x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_5',
use_bias=False)(x)
x = BatchNormalization(name='norm_5')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Layer 6
x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_6',
use_bias=False)(x)
x = BatchNormalization(name='norm_6')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 7
x = Conv2D(128, (1,1), strides=(1,1), padding='same', name='conv_7',
use_bias=False)(x)
x = BatchNormalization(name='norm_7')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 8
x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_8',
use_bias=False)(x)
x = BatchNormalization(name='norm_8')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Layer 9
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_9',
use_bias=False)(x)
x = BatchNormalization(name='norm_9')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 10
x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_10',
use_bias=False)(x)
x = BatchNormalization(name='norm_10')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 11
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_11',
use_bias=False)(x)
x = BatchNormalization(name='norm_11')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 12
x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_12',
use_bias=False)(x)
x = BatchNormalization(name='norm_12')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 13
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_13',
use_bias=False)(x)
x = BatchNormalization(name='norm_13')(x)
x = LeakyReLU(alpha=0.1)(x)
```

```
skip_connection = x

x = MaxPooling2D(pool_size=(2, 2))(x)

# Layer 14
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_14',
use_bias=False)(x)
x = BatchNormalization(name='norm_14')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 15
x = Conv2D(512, (1,1), strides=(1,1), padding='same', name='conv_15',
use_bias=False)(x)
x = BatchNormalization(name='norm_15')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 16
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_16',
use_bias=False)(x)
x = BatchNormalization(name='norm_16')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 17
x = Conv2D(512, (1,1), strides=(1,1), padding='same', name='conv_17',
use_bias=False)(x)
x = BatchNormalization(name='norm_17')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 18
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_18',
use_bias=False)(x)
x = BatchNormalization(name='norm_18')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 19
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_19',
use_bias=False)(x)
x = BatchNormalization(name='norm_19')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 20
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_20',
use_bias=False)(x)
x = BatchNormalization(name='norm_20')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 21
skip_connection = Conv2D(64, (1,1), strides=(1,1), padding='same',
name='conv_21', use_bias=False)(skip_connection)
skip_connection = BatchNormalization(name='norm_21')(skip_connection)
skip_connection = LeakyReLU(alpha=0.1)(skip_connection)
skip_connection = Lambda(space_to_depth_x2)(skip_connection)

x = concatenate([skip_connection, x])

# Layer 22
```

```
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_22',
use_bias=False)(x)
x = BatchNormalization(name='norm_22')(x)
x = LeakyReLU(alpha=0.1)(x)
# As per the comment below, we only need 4 outputs so we are adding a dense
layer as an output with one neuron per dimension
#Important - the output of layer22 is a 6*6 feature map with 1024 feature
maps so 6*6*1024 - This is not really what we need
#Therefore we need to flatten the output so we can apply the final
transforamtion in the dense and get the output we nee
#output.shape(none, 4)
#none = the number of batches if we were splitting our dataset in batches


x = Flatten()(x)
x = Dropout(0.5)(x)
output = Dense(4, activation='linear')(x)

# Layer 23
#What they are doing here is to recreante a dense layer using a convolusion
but they are just fixing the weights for each feature
#map but reading each pixel at a time with those same weights in each
feature map
#What we want is to adapt this layer to the number of outputs we want. In
our case - xmin, ymin, xmax, ymax

#x = Conv2D(BOX * (4 + 1 + CLASS), (1,1), strides=(1,1), padding='same',
name='conv_23')(x)
#output = Reshape((GRID_H, GRID_W, BOX, 4 + 1 + CLASS))(x)

# small hack to allow true_boxes to be registered when Keras build the
model
# for more information: https://github.com/fchollet/keras/issues/2790
#output = Lambda(lambda args: args[0])([output, true_boxes])

model = Model(input_image, output)
```

Freeze layers

Freezing layers - this are needed because when we printed the original summary we have 50million+ trainable parameters and even if we have prelearnt weights, it will optimise them again

Total params: 50,695,396 Trainable params: 50,674,724 Non-trainable params: 20,672

In [ ]:

```
#for i in range(1,23):
    #model.layers[i].trainable = False
```

In [9]:

```
model.summary()
```

```
_________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #
Connected
to
=================================================================
=====================
input_3 (InputLayer)         (None, 192, 192, 3)       0
```

conv_1 (Conv2D)	(None, 192, 192, 32)	864	input_3[0][0]
norm_1 (BatchNormalization)	(None, 192, 192, 32)	128	conv_1[0][0]
leaky_re_lu_44 (LeakyReLU)	(None, 192, 192, 32)	0	norm_1[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 96, 96, 32)	0	leaky_re_lu_44[0][0]
conv_2 (Conv2D)	(None, 96, 96, 64)	18432	max_pooling2d_11[0][0]
norm_2 (BatchNormalization)	(None, 96, 96, 64)	256	conv_2[0][0]
leaky_re_lu_45 (LeakyReLU)	(None, 96, 96, 64)	0	norm_2[0][0]
max_pooling2d_12 (MaxPooling2D)	(None, 48, 48, 64)	0	leaky_re_lu_45[0][0]
conv_3 (Conv2D)	(None, 48, 48, 128)	73728	max_pooling2d_12[0][0]
norm_3 (BatchNormalization)	(None, 48, 48, 128)	512	conv_3[0][0]
leaky_re_lu_46 (LeakyReLU)	(None, 48, 48, 128)	0	norm_3[0][0]
conv_4 (Conv2D)	(None, 48, 48, 64)	8192	leaky_re_lu_46[0][0]
norm_4 (BatchNormalization)	(None, 48, 48, 64)	256	conv_4[0][0]
leaky_re_lu_47 (LeakyReLU)	(None, 48, 48, 64)	0	norm_4[0][0]

```
_________________________________________________________________
conv_5 (Conv2D)              (None, 48, 48, 128)  73728       leaky_re_l
u_47[0][0]
_________________________________________________________________
norm_5 (BatchNormalization)  (None, 48, 48, 128)  512         conv_5[0][
0]
_________________________________________________________________
leaky_re_lu_48 (LeakyReLU)   (None, 48, 48, 128)  0           norm_5[0][
0]
_________________________________________________________________
max_pooling2d_13 (MaxPooling2D) (None, 24, 24, 128)  0        leaky_re_l
u_48[0][0]
_________________________________________________________________
conv_6 (Conv2D)              (None, 24, 24, 256)  294912      max_poolin
g2d_13[0][0]
_________________________________________________________________
norm_6 (BatchNormalization)  (None, 24, 24, 256)  1024        conv_6[0][
0]
_________________________________________________________________
leaky_re_lu_49 (LeakyReLU)   (None, 24, 24, 256)  0           norm_6[0][
0]
_________________________________________________________________
conv_7 (Conv2D)              (None, 24, 24, 128)  32768       leaky_re_l
u_49[0][0]
_________________________________________________________________
norm_7 (BatchNormalization)  (None, 24, 24, 128)  512         conv_7[0][
0]
_________________________________________________________________
leaky_re_lu_50 (LeakyReLU)   (None, 24, 24, 128)  0           norm_7[0][
0]
_________________________________________________________________
conv_8 (Conv2D)              (None, 24, 24, 256)  294912      leaky_re_l
u_50[0][0]
_________________________________________________________________
norm_8 (BatchNormalization)  (None, 24, 24, 256)  1024        conv_8[0][
0]
_________________________________________________________________
leaky_re_lu_51 (LeakyReLU)   (None, 24, 24, 256)  0           norm_8[0][
0]
_________________________________________________________________
max_pooling2d_14 (MaxPooling2D) (None, 12, 12, 256)  0        leaky_re_l
u_51[0][0]
```

conv_9 (Conv2D)	(None, 12, 12, 512)	1179648	max_poolin g2d_14[0][0]
norm_9 (BatchNormalization)	(None, 12, 12, 512)	2048	conv_9[0][0]
leaky_re_lu_52 (LeakyReLU)	(None, 12, 12, 512)	0	norm_9[0][0]
conv_10 (Conv2D)	(None, 12, 12, 256)	131072	leaky_re_l u_52[0][0]
norm_10 (BatchNormalization)	(None, 12, 12, 256)	1024	conv_10[0] [0]
leaky_re_lu_53 (LeakyReLU)	(None, 12, 12, 256)	0	norm_10[0] [0]
conv_11 (Conv2D)	(None, 12, 12, 512)	1179648	leaky_re_l u_53[0][0]
norm_11 (BatchNormalization)	(None, 12, 12, 512)	2048	conv_11[0] [0]
leaky_re_lu_54 (LeakyReLU)	(None, 12, 12, 512)	0	norm_11[0] [0]
conv_12 (Conv2D)	(None, 12, 12, 256)	131072	leaky_re_l u_54[0][0]
norm_12 (BatchNormalization)	(None, 12, 12, 256)	1024	conv_12[0] [0]
leaky_re_lu_55 (LeakyReLU)	(None, 12, 12, 256)	0	norm_12[0] [0]
conv_13 (Conv2D)	(None, 12, 12, 512)	1179648	leaky_re_l u_55[0][0]
norm_13 (BatchNormalization)	(None, 12, 12, 512)	2048	conv_13[0] [0]

leaky_re_lu_56 (LeakyReLU) [0]	(None, 12, 12, 512)	0	norm_13[0]
max_pooling2d_15 (MaxPooling2D) u_56[0][0]	(None, 6, 6, 512)	0	leaky_re_l
conv_14 (Conv2D) g2d_15[0][0]	(None, 6, 6, 1024)	4718592	max_poolin
norm_14 (BatchNormalization) [0]	(None, 6, 6, 1024)	4096	conv_14[0]
leaky_re_lu_57 (LeakyReLU) [0]	(None, 6, 6, 1024)	0	norm_14[0]
conv_15 (Conv2D) u_57[0][0]	(None, 6, 6, 512)	524288	leaky_re_l
norm_15 (BatchNormalization) [0]	(None, 6, 6, 512)	2048	conv_15[0]
leaky_re_lu_58 (LeakyReLU) [0]	(None, 6, 6, 512)	0	norm_15[0]
conv_16 (Conv2D) u_58[0][0]	(None, 6, 6, 1024)	4718592	leaky_re_l
norm_16 (BatchNormalization) [0]	(None, 6, 6, 1024)	4096	conv_16[0]
leaky_re_lu_59 (LeakyReLU) [0]	(None, 6, 6, 1024)	0	norm_16[0]
conv_17 (Conv2D) u_59[0][0]	(None, 6, 6, 512)	524288	leaky_re_l
norm_17 (BatchNormalization) [0]	(None, 6, 6, 512)	2048	conv_17[0]
leaky_re_lu_60 (LeakyReLU) [0]	(None, 6, 6, 512)	0	norm_17[0]

conv_18 (Conv2D)	(None, 6, 6, 1024)	4718592	leaky_relu_60[0][0]
norm_18 (BatchNormalization)	(None, 6, 6, 1024)	4096	conv_18[0][0]
leaky_re_lu_61 (LeakyReLU)	(None, 6, 6, 1024)	0	norm_18[0][0]
conv_19 (Conv2D)	(None, 6, 6, 1024)	9437184	leaky_relu_61[0][0]
norm_19 (BatchNormalization)	(None, 6, 6, 1024)	4096	conv_19[0][0]
conv_21 (Conv2D)	(None, 12, 12, 64)	32768	leaky_relu_56[0][0]
leaky_re_lu_62 (LeakyReLU)	(None, 6, 6, 1024)	0	norm_19[0][0]
norm_21 (BatchNormalization)	(None, 12, 12, 64)	256	conv_21[0][0]
conv_20 (Conv2D)	(None, 6, 6, 1024)	9437184	leaky_relu_62[0][0]
leaky_re_lu_64 (LeakyReLU)	(None, 12, 12, 64)	0	norm_21[0][0]
norm_20 (BatchNormalization)	(None, 6, 6, 1024)	4096	conv_20[0][0]
lambda_2 (Lambda)	(None, 6, 6, 256)	0	leaky_relu_64[0][0]
leaky_re_lu_63 (LeakyReLU)	(None, 6, 6, 1024)	0	norm_20[0][0]
concatenate_2 (Concatenate)	(None, 6, 6, 1280)	0	lambda_2[0][0]

```
                                                                 leaky_re_l
u_63[0][0]
__________________________________________________________________________
__________________________
conv_22 (Conv2D)                (None, 6, 6, 1024)   11796480    concatenat
e_2[0][0]
__________________________________________________________________________
__________________________
norm_22 (BatchNormalization)    (None, 6, 6, 1024)   4096        conv_22[0]
[0]
__________________________________________________________________________
__________________________
leaky_re_lu_65 (LeakyReLU)      (None, 6, 6, 1024)   0           norm_22[0]
[0]
__________________________________________________________________________
__________________________
flatten_2 (Flatten)             (None, 36864)        0           leaky_re_l
u_65[0][0]
__________________________________________________________________________
__________________________
dropout_2 (Dropout)             (None, 36864)        0           flatten_2[
0][0]
__________________________________________________________________________
__________________________
dense_2 (Dense)                 (None, 4)            147460      dropout_2[
0][0]
==========================================================================
========================
Total params: 50,695,396
Trainable params: 50,674,724
Non-trainable params: 20,672
__________________________________________________________________________
__________________________
```