**Quick Start with Tensorflow Callbacks**

[**Ashish Salaskar**](https://www.analyticsvidhya.com/blog/author/ashish0765/)**— Published On August 31, 2021**

[Deep Learning](https://www.analyticsvidhya.com/blog/category/deep-learning/) [Libraries](https://www.analyticsvidhya.com/blog/category/libraries/) [Programming](https://www.analyticsvidhya.com/blog/category/programming/) [Python](https://www.analyticsvidhya.com/blog/category/python-2/)

**This article was published as a part of the**[**Data Science Blogathon**](https://datahack.analyticsvidhya.com/contest/data-science-blogathon-11/)

What are Tensorflow Callbacks?

Tensorflow callbacks are functions or blocks of code which are executed during a specific instant while training a Deep Learning Model.

 We all are familiar with the Training process of any Deep Learning model. With the models getting more complex and resource-intensive the training times also have significantly increased. So it’s usual for models to take many hours to train. In the usual workflow before training the model, we fix all the options and parameters like learning rate, optimizers, losses. etc and start the model training. Once the training process is started there is no way to pause the training in case we want to change some params. Also, in some cases when the model has been trained for several hours and we want to tweak some parameters at the later stages, it is impossible to do so. This is where TensorFlow callbacks come to the rescue.

How to use Callbacks

1. First define the callbacks  
2. Pass the callbacks when calling the model.fit()

# Stop training if NaN is encountered

NanStop = TerminateOnNaN()

# Decrease lr by 10%

LrValAccuracy = ReduceLROnPlateau(monitor='val\_accuracy', patience=1, factor= 0.9, mode='max', verbose=0)

model.fit(X\_train,y\_train,

epochs=10,

validation\_data=(X\_test,y\_test),

callbacks = [NanStop, LrValAccuracy])

Let us have a look at some of the most useful callbacks

EarlyStopping

When we are training our models, we usually take a look at the metrics in order to monitor how well the model is performing. Usually, if we see extremely high metrics, we can conclude that our model is overfitting and if our metrics are really low then we are underfitting.

**Ready to start your data science journey?**

Master **23+ tools** & learn **50+ real-world projects** to transform your career in Data Science.

[Enroll Now](https://blackbelt.analyticsvidhya.com/plus?utm_source=blog_india&utm_medium=desktop_banner_between_articles&utm_content=tools)

In case if the metrics increase above a certain range we can stop the training to prevent overfitting. The EarlyStopping callback allows us to do exactly this.

early\_stop\_cb = tf.keras.callbacks.EarlyStopping(

monitor='val\_loss', min\_delta=0, patience=0, verbose=0,

mode='auto'

)

* monitor: The metric you want to monitor while training
* min\_delta: The minimum amount of change in the metric you want to consider as an improvement over the previous epoch
* patience: The number of epochs for which you wait for the metric to wait. Else, you stop the training.
* verbose : 0: don’t print anything, 1: show a progress bar, 2: print only epoch number
* mode :
* “auto” – try to detect the behaviour automatically from the metrics are given
* “min” – stop training if metrics stopped decreasing
* “max” – stop training if metrics stopped increasing

LambdaCallback

This callback is used to call certain lambda functions at specific times during the training process.

tf.keras.callbacks.LambdaCallback(

on\_epoch\_begin=None, on\_epoch\_end=None, on\_batch\_begin=None, on\_batch\_end=None,

on\_train\_begin=None, on\_train\_end=None, \*\*kwargs

)

Here we can pass any lambda function we need to execute at the specified time. Let’s see what the arguments mean

* on\_epoch\_begin:  call the function at the beginning of each epoch.
* on\_epoch\_begin: call the function at the end of each epoch.
* on\_batch\_begin:  call the function at the beginning of each batch.
* on\_batch\_end: calls the function at the end of each batch.
* on\_train\_begin: calls the function when the model starts training
* on\_train\_end: calls when the model training is completed

print\_batch\_callback = LambdaCallback(

on\_batch\_begin=lambda bat,log: print(bat),

on\_batch\_begin=lambda bat,log: print(bat)

)

LearningRateScheduler

One of the most common tasks during the training process is to change the learning rates. Usually, as the model approaches the loss-minimization minima (best fit) we gradually start decreasing the learning rate to have better convergence.

Let’s see a simple example where we want to reduce our learning rate by 5% for every 3rd epoch. Here we need to pass in a function to the schedule argument which specifies the logic for change in learning rate.

def schedule(epoch,lr):

if epoch % 3 == 0:

lr = lr - (lr\*.05)

return lr

return lr

# Decrease lr by 5% for every 3rd epoch

LrScheduler = tf.keras.callbacks.LearningRateScheduler(schedule,verbose=1)

ModelCheckpoint

We use this callback in order to save our Model at different frequencies. This allows us to save weights at intermediate steps so that if needed we can load weights later.

tf.keras.callbacks.ModelCheckpoint(

filepath, monitor='val\_loss', verbose=0, save\_best\_only=False,

save\_weights\_only=False, mode='auto', save\_freq='epoch'

)

**file-path**: the location where the mode  
**monitor**: metric to be monitored  
**save\_best\_only**: True: Save only the best model,  False: Save all the models when metric improves  
**mode**: min, max, or auto  
**save\_weights\_only**: False: save only model weights, True: Save both model weights and model architecture

For example, let’s see an example to save the model having the best accuracy

filePath = "models/Model1\_weights.{epoch:02d}.hdf5"

model\_checkpoint\_callback = tf.keras.callbacksModelCheckpoint(

filepath=filePath,

save\_weights\_only=True,

monitor='val\_accuracy',

mode='max')

Here we specify the file path using some template strings. {epoch:02d} is substituted by the epoch number when saving the model

ReduceLROnPlateau

This callback is used to reduce the training rate when the specific metric has stopped increasing and reached a plateau.

tf.keras.callbacks.ReduceLROnPlateau(

monitor='val\_loss', factor=0.1, patience=10, verbose=0,

mode='auto', min\_delta=0.0001, cooldown=0, min\_lr=0, \*\*kwargs

)

factor: the factor by which LR is reduced. New learning rate = old\_learning\_rate \* factor  
min\_delta: minimum change needed to be considered as an improvement  
cooldown: number of epochs to wait until the LR is reduced  
min\_lr: a minimum value below which the Learning rate cant go

TerminateOnNaN

This callback stops the training process when any loss becomes NaN

tf.keras.callbacks.TerminateOnNaN()

Tensorboard

Tensorboard allows us to display information regarding the training process like Metrics, Training graphs, Activation function histograms, and other distribution of gradients. To use tensorboard we first need to set up a log\_dir where the tensorboard files get saved to.

log\_dir="logs"

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1, write\_graph=True)

* log\_dir: directory to which the files are saved
* histogram\_freq: epochs frequency for which the histogram and gradient maps are computed
* write\_graph: whether we need to display and visualize graphs in the tensorboard

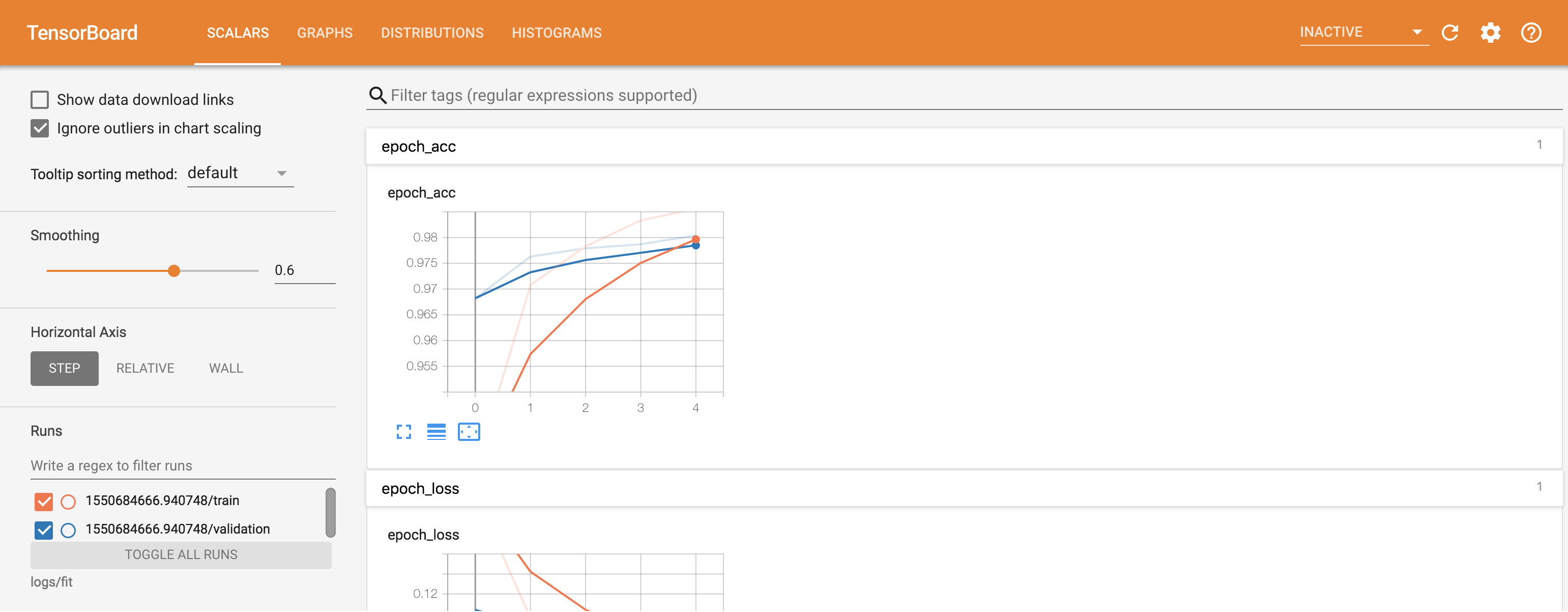


Image 1 (link below)

Write your own Callbacks

Apart from the inbuilt callbacks, we can define and use our own callbacks for different purposes. For example, let us say we want to define our own metric which gets calculated at the end of each epoch.

# Monitor MicroF1 and AUC Score

class Metrics\_Callback(tf.keras.callbacks.Callback):

def \_\_init\_\_(self,x\_val,y\_val):

self.x\_val = x\_val

self.y\_val = y\_val

def on\_train\_begin(self, logs={}):

self.history = {"auc\_score":[],"micro\_f1":[]}

def on\_epoch\_end(self, epoch, logs={}):

auc\_score = roc\_auc\_score(self.y\_val, model.predict\_proba(self.x\_val))

y\_true = [0 if x[0]==1.0 else 1 for x in self.y\_val]

f1\_s = f1\_score(y\_true,self.model.predict\_classes(self.x\_val), average='micro')

self.history["auc\_score"].append(auc\_score)

self.history["micro\_f1"].append(f1\_s)

Metrics = Metrics\_Callback(X\_test,y\_test)

Here we want to calculate the F1 score and AUC score at the end of each epoch. in the \_\_init\_\_ method we read the data needed to calculate the scores. Then at the end of each epoch, we calculate the metrics in the on\_epoch\_end function. We can use the following methods to execute code at different times-

on\_epoch\_begin: called at the beginning of each epoch.

on\_epoch\_begin: called at the end of each epoch.

on\_batch\_begin: called at the beginning of each batch.

on\_batch\_end: called at the end of each batch.

on\_train\_begin: called when the model starts training

on\_train\_end: called when the model training is completed

Conclusion

These were a few commonly used and most popular callbacks. The official TensorFlow documentation: https://www.tensorflow.org/api\_docs/python/tf/keras/callbacks/Callback gives us in-detail information about various other callbacks and their related use cases.