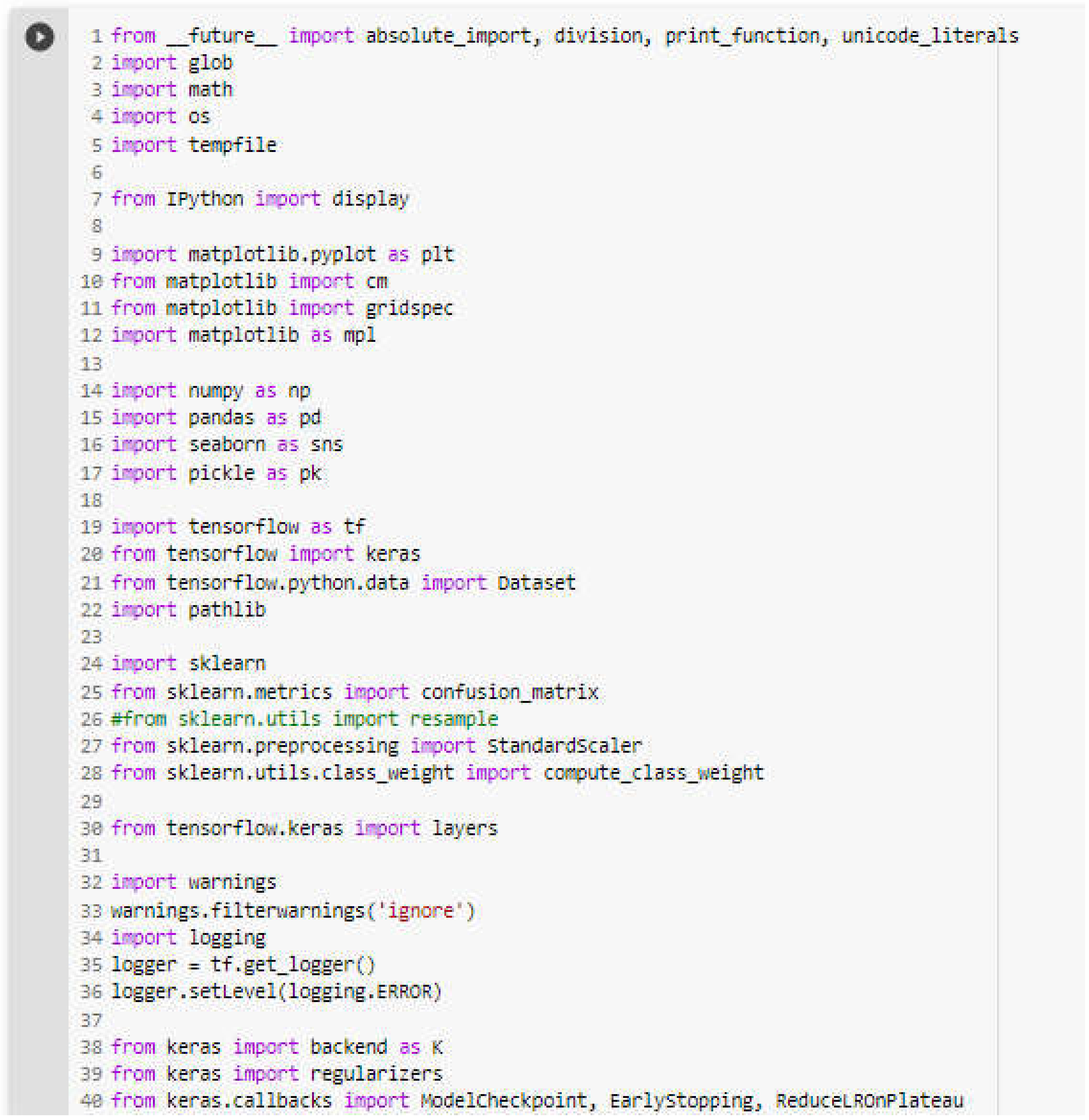
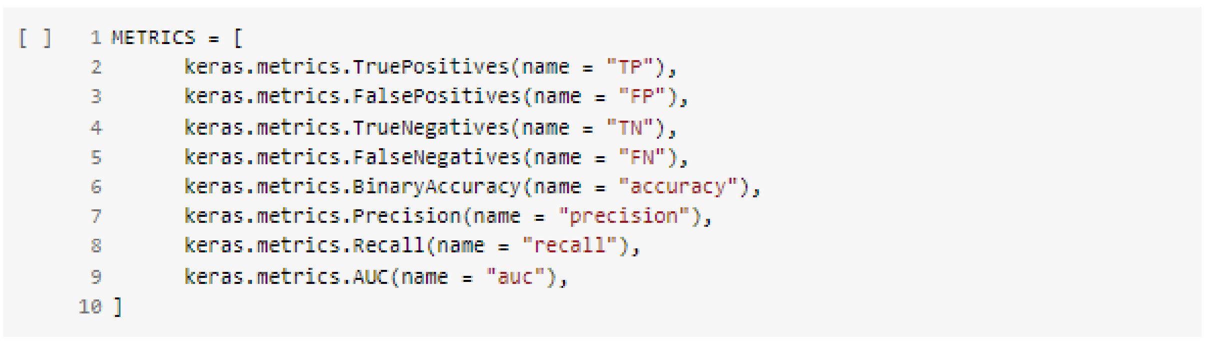
Reference : https://github.com/bessammehenni/Forest_wildfire_spreading_convLSTM_prediction

Extract of the code, from author B. Mehenni

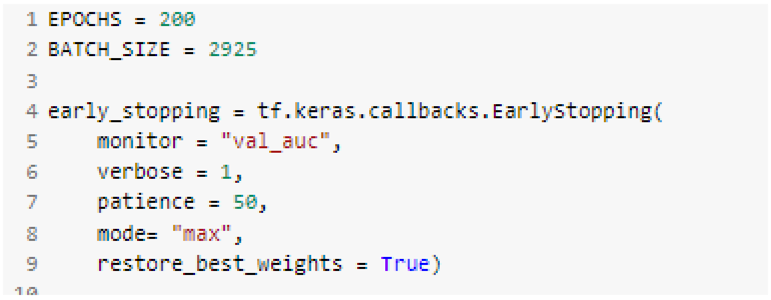


```python
from __future__ import absolute_import, division, print_function, unicode_literals
import glob
import math
import os
import tempfile

from IPython import display

import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import gridspec
import matplotlib as mpl

import numpy as np
import pandas as pd
import seaborn as sns
import pickle as pk

import tensorflow as tf
from tensorflow import keras
from tensorflow.python.data import Dataset
import pathlib

import sklearn
from sklearn.metrics import confusion_matrix
#from sklearn.utils import resample
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight

from tensorflow.keras import layers

import warnings
warnings.filterwarnings('ignore')
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)

from keras import backend as K
from keras import regularizers
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
```

Defining metrics and early stopping

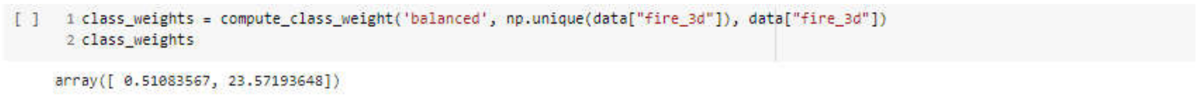


```python
[ ]
METRICS = [
      keras.metrics.TruePositives(name = "TP"),
      keras.metrics.FalsePositives(name = "FP"),
      keras.metrics.TrueNegatives(name = "TN"),
      keras.metrics.FalseNegatives(name = "FN"),
      keras.metrics.BinaryAccuracy(name = "accuracy"),
      keras.metrics.Precision(name = "precision"),
      keras.metrics.Recall(name = "recall"),
      keras.metrics.AUC(name = "auc"),
]
```



```python
EPOCHS = 200
BATCH_SIZE = 2925

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor = "val_auc",
    verbose = 1,
    patience = 50,
    mode= "max",
    restore_best_weights = True)
```

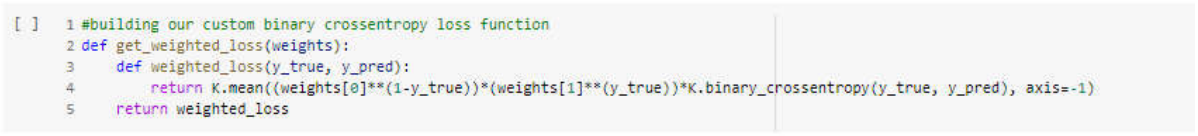
Train the model with class weights



```python
class_weights = compute_class_weight('balanced', np.unique(data["fire_3d"]), data["fire_3d"])
class_weights
```

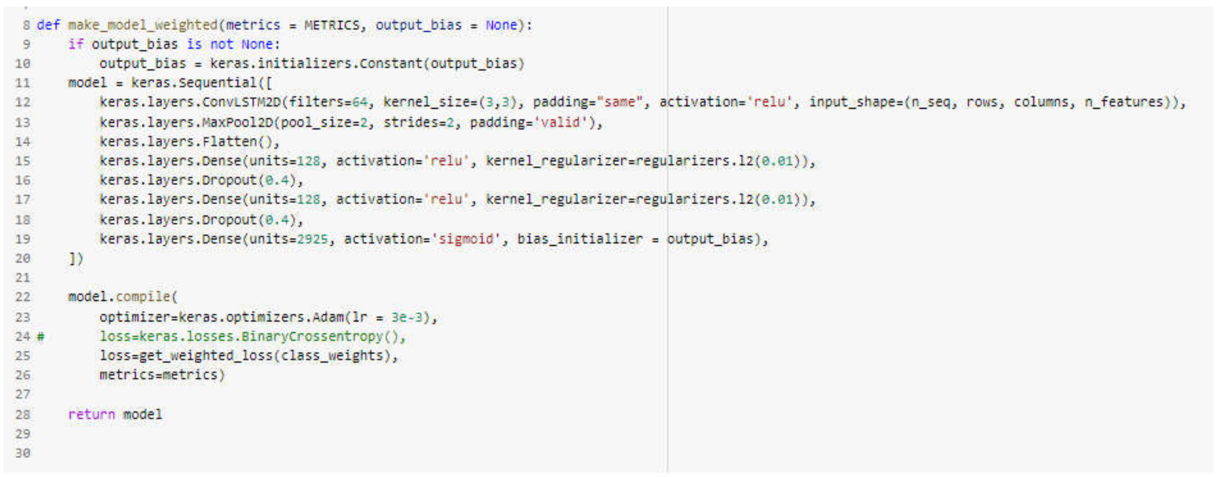
```
array([ 0.51083567, 23.57193648])
```

```python
#class_weights = np.zeros((1,2))
#class_weights[:, 0] += weight_for_0
#class_weights[:, 1] += weight_for_1
```

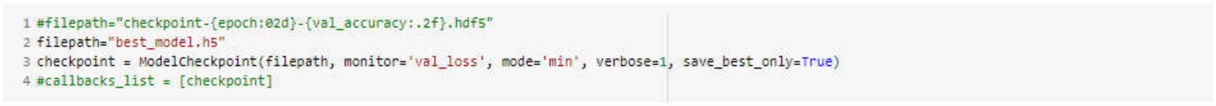


```python
#building our custom binary crossentropy loss function
def get_weighted_loss(weights):
    def weighted_loss(y_true, y_pred):
        return K.mean((weights[0]**(1-y_true))*(weights[1]**(y_true))*K.binary_crossentropy(y_true, y_pred), axis=-1)
    return weighted_loss
```

```python
# input shape is [samples, timesteps, rows, columns, features]
n_seq = 3
rows=45
columns=65
#n_features=9

```



```python
def make_model_weighted(metrics = METRICS, output_bias = None):
    if output_bias is not None:
        output_bias = keras.initializers.Constant(output_bias)
    model = keras.Sequential([
        keras.layers.ConvLSTM2D(filters=64, kernel_size=(3,3), padding="same", activation='relu', input_shape=(n_seq, rows, columns, n_features)),
        keras.layers.MaxPool2D(pool_size=2, strides=2, padding='valid'),
        keras.layers.Flatten(),
        keras.layers.Dense(units=128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(units=128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(units=2925, activation='sigmoid', bias_initializer = output_bias),
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(lr = 3e-3),
#       loss=keras.losses.BinaryCrossentropy(),
        loss=get_weighted_loss(class_weights),
        metrics=metrics)

    return model


```



```python
#filepath="checkpoint-{epoch:02d}-{val_accuracy:.2f}.hdf5"
filepath="best_model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', mode='min', verbose=1, save_best_only=True)
#callbacks_list = [checkpoint]
```

```python
train_feat=Xt
val_feat=Xv
test_feat=Xts
```

```
[ ]
train_labels=yt
val_labels=yv
test_labels=yts
```

```
weighted_model = make_model_weighted()
weighted_model.load_weights(initial_weights)

weighted_history = weighted_model.fit(
    train_feat,
    train_labels,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    callbacks = [early_stopping, reduce_lr],
#     callbacks = [early_stopping],
    validation_data = (val_feat, val_labels),
    verbose=0)

```

```
Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.002100000018253922.

Epoch 00056: ReduceLROnPlateau reducing learning rate to 0.0014699999475851653.

Epoch 00064: ReduceLROnPlateau reducing learning rate to 0.0010289999307133257.

Epoch 00072: ReduceLROnPlateau reducing learning rate to 0.0007202999433502554.

Epoch 00083: ReduceLROnPlateau reducing learning rate to 0.0005042099684942513.

Epoch 00102: ReduceLROnPlateau reducing learning rate to 0.0003529469657223671.

Epoch 00110: ReduceLROnPlateau reducing learning rate to 0.0002470628678565845.
Restoring model weights from the end of the best epoch.
Epoch 00112: early stopping
```

Evaluate metrics

```
[ ]
train_predict_weighted = weighted_model.predict(train_feat, batch_size = BATCH_SIZE)
test_predict_weighted = weighted_model.predict(test_feat, batch_size = BATCH_SIZE)
```

```
[ ]
weighted_results = weighted_model.evaluate(test_feat, test_labels,
                                           batch_size=BATCH_SIZE, verbose=0)
for name, value in zip(weighted_model.metrics_names, weighted_results):
    print(name, ': ', value)
print()

plot_cm(test_labels.reshape([-1]), test_predict_weighted.reshape([-1]))
plt.savefig('cm_weighted_model.png')
```

```
loss :  1.7021307945251465
TP :  53.0
FP :  2300.0
TN :  23648.0
FN :  324.0
accuracy :  0.9003229141235352
precision :  0.022524436935782433
recall :  0.1405835598707199
auc :  0.6007218956947327

True Negatives:  24807
False Positives:  1141
False Negatives:  355
True Positives:  22
Total Fire cells:  377
```