```
# Crée un réseau de neurones avec Keras

from keras.models import Sequential
from keras.layers import Dense, Dropout
import numpy as np
import h5py
import sys
from os import remove, path
from glob import glob
import matplotlib.pyplot as plt

# Paramètres
#Affichage du traitement des données
AFFICHAGE = True
# Nombre de coefficients cepstraux
nbCoef = 13
# Nombre de valeurs à prélever pour obtenir une "fenêtre de parole"
nbVal = 31
# Décalage entre chaque prélevement de fenêtre de parole
shift = 10
#Nombre de langages différents = de neurones en sortie
nbSorties = 4

#Formatte les données en le mettant dans un tableau hdf5 temporaire (utile pour les
    fichiers dépassant la taille de la ram)
def formatDataHdf5(dataFileName, hdf5Tmp, withoutLabel=False):
[...]

def formatDataMemory(dataFileName, withoutLabel=False):
[...]

def writeProbaToFile(outFileName, exampleFileName, exampleFileLang, probaArray,
    langStrList, numberPrecision=2, numberOfMinSpaces=2):
[...]

#fonction temporaire : génère les fichiers txt pour voir les probas résultat
def generatePredict(model, predictFolder, language):
[...]

# temp  : pour calculer la variance de chaque coeff mfcc
def calculateVariances(dataFileName):
    hdf5In = h5py.File(dataFileName, "r")

    nbValue = 0
    averages = np.zeros([nbCoef], dtype=np.float32)
    for dataset in hdf5In.values():                              # Pour chaque dataset du
        fichier hdf5
        print(dataset)
        nbValue += dataset.shape[0]
        for mfccArray in dataset:
            averages += mfccArray[:nbCoef]
    averages /= nbValue  # calcul moyenne

    variances = np.zeros([nbCoef], dtype=np.float32)
    for dataset in hdf5In.values():                              # Pour chaque dataset du
        fichier hdf5
```

```
        print(dataset)
        for mfccArray in dataset:
            variances += (mfccArray[:nbCoef] - averages)**2
    variances /= nbValue

    return variances

if __name__ == '__main__':
    # On récupère les données
    dataFileName = sys.argv[1]
    devFileName = sys.argv[2]
    testFileName = sys.argv[3]

    # On formatte les données
    (X, Y) = formatDataMemory(dataFileName)
    (Xdev, Ydev) = formatDataMemory(devFileName)
    (Xtest, Ytest) = formatDataMemory(testFileName)

    # On calcule la variance de chaque coefficient mfcc
    # variances obtenues à l'aide de la fonction calculateVariances
    variances = [ 8.44184971,  3.82373691,  2.00800991,  1.17921674,  1.13604367,
        0.72418207,  0.55174363,  0.44808063,  0.40456247,  0.30078402,  0.27061722,
        0.24347636, 0.05386801]#calculateVariances(dataFileName)
    # On concatène nbVal fois le tableau de variances afin de pouvoir effectuer la division
        en une seule fois
    variancesDuplicate = []
    for i in range(nbVal): variancesDuplicate[i*nbVal:] = variances
    # et on divise chaque coefficient mfcc du train et du dev par sa variance
    X /= variancesDuplicate
    Xdev /= variancesDuplicate

    # On crée le modèle séquentiel, avec 4 couches
    inputShape = nbVal * nbCoef
    model = Sequential()
    model.add(Dense(inputShape, input_shape=(inputShape,),
        kernel_initializer='glorot_normal', activation='relu'))
    model.add(Dropout(0.2, input_shape=(inputShape,)))
    model.add(Dense(256, kernel_initializer='glorot_normal', activation='relu'))
    model.add(Dropout(0.2, input_shape=(inputShape,)))
    model.add(Dense(256, kernel_initializer='glorot_normal', activation='relu'))
    model.add(Dense(nbSorties, kernel_initializer='glorot_normal', activation='softmax'))

    # On compile le modèle
    model.compile(loss='categorical_crossentropy', optimizer='RMSprop',
        metrics=['accuracy'])

    # On entraîne le modèle
    history = model.fit(X, Y, epochs=100, batch_size=128, validation_data=(Xdev, Ydev))

    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'dev'], loc='upper left')
    plt.savefig('testAvecDivVariance.png')
```

```
generatePredict(model, 'hdf5Predict/Arabic', 'Arabic')
generatePredict(model, 'hdf5Predict/English', 'English')
generatePredict(model, 'hdf5Predict/French', 'French')
generatePredict(model, 'hdf5Predict/German', 'German')
```