Training_CIFAR

February 12, 2021

Script for Training with standard and manipulated CIFAR10 dataset
(see Script 'Dataset.py')with a simple CNN and AlexNet

Written by C.Joachim based on different sources (see references in code)
January 2021

If the manipulated dataset is to be used, run this script in advance such that the dataset can be loaded

[ ]:
```
import tensorflow as tf
import time
from tensorflow import keras
```

[ ]:
```
#Choose parameters for training
Dataset = 1    #Dataset=1 for standard CIFAR10, Dataset=2 for own dataset
Model   = 1    #Model=1 for simple CNN, Model=2 for AlexNet
Epochs  = 100  #choose number of epochs
Batch   = 256  #choose Batch Size (standard is 32, 64, 128, 256)
```

[ ]:
```
#import dataset
if Dataset==1:
    #load CIFAR10
    from tensorflow.keras import datasets
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
    #10 Categories --> 10 output neurons needed
    cat = 10
if Dataset==2:
    #load manipulated CIFAR10 dataset (already normalized and OneHot-encoded)
    %store -r
    (X_train, y_train, X_test, y_test) = Data_CIFAR
    #11 Categories --> 11 output neurons needed
    cat = 11
```

[ ]:
```
if Model==1:
    #use Model from MNIST-Tutorial from 'ct Python-Projekte by Heise
    #with modified input size and adjustable number of output neurons
    #
    #if the simple CNN is chosen, normalize data
    X_train = X_train/255
```

```
    X_test  = X_test/255
    #if the Dataset is original CIFAR-10, we still need to
    #convert to OneHot-Vector
    if Dataset==1:
        y_train  = tf.keras.utils.to_categorical(y_train)
        y_test   = tf.keras.utils.to_categorical(y_test)
    #build Model
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense,Flatten
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.layers import Conv2D
    from tensorflow.keras.layers import MaxPooling2D
    model = Sequential()
    model.add(Conv2D(32,
        kernel_size = (3, 3),
        activation  = 'relu',
        input_shape = (32,32,3)))
    model.add(Conv2D(64,
        kernel_size = (3, 3),
        activation  = 'relu'))
    model.add(MaxPooling2D(
        pool_size   = (2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(200,
        activation  = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(cat,
        activation  = 'softmax'))
if Model==2:
    #recreate AlexNet
    #
    #validation dataset is created from training dataset
    #last 20% of Training Set used for Validation
    valslice = int(round(len(y_train)*0.2))
    X_val    = X_train[:valslice]
    y_val    = y_train[:valslice]
    X_train  = X_train[valslice:]
    y_train  = y_train[valslice:]
    #define function which is used to normalize and reshape the images such␣
 ↪that the
    #input is in the form that is expected by AlexNet (normalized, size 227x227)
    #modified function (not used label as in original) from
    #https://towardsdatascience.com/
 ↪implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98
    def process_images(image):
        # Normalize images to have a mean of 0 and standard deviation of 1
```

```
        image = tf.image.per_image_standardization(image)
        # Resize images from 32x32 to 227x227
        image = tf.image.resize(image, (227,227))
        return image
    #process images with the above function
    X_train   = process_images(X_train)
    X_val     = process_images(X_val)
    X_test    = process_images(X_test)
    #Setup Alexnet as shown at
    #https://towardsdatascience.com/
↪implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98
    #but with variable number of output neurons
    model = keras.models.Sequential([
        keras.layers.Conv2D(
        filters     = 96,
        kernel_size = (11,11),
        strides     = (4,4),
        activation  = 'relu',
        input_shape = (227,227,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(
        pool_size   = (3,3),
        strides     = (2,2)),
    keras.layers.Conv2D(
        filters     = 256,
        kernel_size = (5,5),
        strides     = (1,1),
        activation  = 'relu',
        padding     = "same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(
        pool_size   = (3,3),
        strides     = (2,2)),
    keras.layers.Conv2D(
        filters     = 384,
        kernel_size = (3,3),
        strides     = (1,1),
        activation  = 'relu',
        padding     = "same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters     = 384,
        kernel_size = (1,1),
        strides     = (1,1),
        activation  = 'relu',
        padding="same"),
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Conv2D(
        filters     = 256,
        kernel_size = (1,1),
        strides     = (1,1),
        activation  = 'relu',
        padding     = "same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(
        pool_size   = (3,3),
        strides     = (2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096,
        activation  = 'relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096,
        activation  = 'relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(cat,
        activation  = 'softmax')])
```

[ ]:
```
#compile the models with loss functions and optimizers as given by
#the respective source after which the model was being built
if Model==1:
    from tensorflow.keras.losses import categorical_crossentropy
    from tensorflow.keras.optimizers import Adam
    model.compile(
    loss      = categorical_crossentropy,
    optimizer = Adam(),
    metrics=['accuracy'])
if Model==2:
    if Dataset==1:
        model.compile(loss='sparse_categorical_crossentropy',
        optimizer = tf.optimizers.SGD(lr=0.001),
        metrics   = ['accuracy'])
    if Dataset==2:
        #for own dataset, have to change to categorical crossentropy
        #because labels are OneHot-encoded
        model.compile(loss='categorical_crossentropy',
        optimizer = tf.optimizers.SGD(lr=0.001),
        metrics   = ['accuracy'])
```

[ ]:
```
#log for tensorboard with chosen variables in logfile-name
log_dir              = "logs/fit/" + "Dataset %s Model %s Epochs %s Batch Size␣
 ↪%s" %(Dataset, Model, Epochs, Batch)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
 ↪histogram_freq=1)
```

[ ]:
```
#training (with callback added for tensorboard)
#measure time between start and end of training
if Model==1:
    start   = time.perf_counter()
    history = model.fit(X_train, y_train,
        batch_size       = Batch,
        epochs           = Epochs,
        verbose          = 1,
        validation_split = 0.2,
        callbacks        = [tensorboard_callback])
    elapsed=time.perf_counter()-start
if Model==2:
    start   = time.perf_counter()
    history = model.fit(X_train, y_train,
        epochs          = Epochs,
        validation_data = (X_val, y_val),
        validation_freq = 1,
        callbacks       = [tensorboard_callback])
    elapsed = time.perf_counter()-start
```

[ ]:
```
#evaluate the model with test data
[loss, accuracy] = model.evaluate(X_test, y_test)
```

[ ]:
```
#write evaluation training time and loss and accuracy to a file
tf.io.write_file("training_time/"+"Dataset %s Model %s Epochs %s Batch Size %s"␣
 ↪%(Dataset, Model, Epochs, Batch), "%f" %elapsed)
tf.io.write_file("evaluate_model/"+"Dataset %s Model %s Epochs %s Batch Size␣
 ↪%s" %(Dataset, Model, Epochs, Batch), "%f %f" %(loss, accuracy))
```

[ ]:
```
#save the model in the TensorFlow-Format
model.save("saved_model/"+"Dataset %s Model %s Epochs %s Batch Size %s"␣
 ↪%(Dataset, Model, Epochs, Batch))
```

[ ]:
```

```