Task-1-keras-Xception

March 13, 2023

This notebook contains the Tensorflow Keras implementation of the Xception model. Refer this paper for Xception model specification: https://arxiv.org/pdf/1610.02357.pdf

It's worth noting this keras implementation is same as the pytorch implementation of Xception model, except for here we output only one scalar probability. In case of pytorch implementation we output two scalar probabilities (softmax). This small difference is due to unavailability of multi class AUC metric in tf/keras (https://github.com/tensorflow/addons/issues/265).

[1]:

```
import tensorflow as tf
import tensorflow.keras

from tensorflow.keras import models, layers
from tensorflow.keras.models import Model, model_from_json, Sequential
from tensorflow.keras.layers import (
    Dense,
    Activation,
    Conv2D,
    Lambda,
    Resizing,
    MaxPooling2D,
    SeparableConv2D,
    BatchNormalization,
    Input,
    GlobalAveragePooling2D
)
from tensorflow.keras.optimizers import SGD
import os
import h5py
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
C:\Users\MSI\anaconda3\envs\tensorflow_learning\lib\site-
packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and
<1.23.0 is required for this version of SciPy (detected version 1.24.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[2]:

```
os.listdir("../dataset")
```

[2]:

```
['QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.snappy.parquet',
 'SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5',
 'SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5']
```

[3]:

```
# import dataset
electron_dataset = h5py.File("../dataset/SingleElectronPt50_IMGCROPS_n249k_RHv1.
 ↪hdf5","r")
electron_imgs=np.array(electron_dataset["X"])
electron_labels=np.array(electron_dataset["y"],dtype=np.int64)

photon_dataset = h5py.File("../dataset/SinglePhotonPt50_IMGCROPS_n249k_RHv1.
 ↪hdf5","r")
photon_imgs=np.array(photon_dataset["X"])
photon_labels=np.array(photon_dataset["y"],dtype=np.int64)
```

[4]:

```
img_arrs = np.vstack((photon_imgs,electron_imgs))
labels = np.hstack((photon_labels,electron_labels)).astype(np.int64)
```

[5]:

```
num_train = int(img_arrs.shape[0]*0.7) if img_arrs.shape[0]%10==0 else␣
 ↪int(img_arrs.shape[0]*0.7)+1
num_val_test = img_arrs.shape[0] - num_train
num_val = int(num_val_test*(2/3)) if num_val_test%3==0 else int(num_val_test*(2/
 ↪3))+1
num_test =num_val_test-num_val

split_seed =42

X_train, X_val_test, y_train, y_val_test = train_test_split(img_arrs, labels,
                                                                          ␣
 ↪test_size=num_val_test, train_size =num_train,
                                                                          ␣
 ↪random_state=split_seed)
X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test,
                                                test_size=num_test, train_size␣
 ↪=num_val,
                                                random_state=split_seed)
```

[7]:

```
def preprocess(inputs,mean=0.5,std=0.5,size=96):
    x = Lambda(lambda inputs: (inputs - mean) /std)(inputs)
    x = Resizing(size,size)(x)
    return x
```

```
def forward_pass(inputs,num_middle_blocks =8,num_classes=2):

    # Begin of entry flow
    x = Conv2D(32, 3, strides = 2, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(64,3,padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    Z = x

    for size in [128, 256, 728]:
        x = Activation('relu')(x)
        x = SeparableConv2D(size, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(size, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = MaxPooling2D(3, strides=2, padding='same')(x)

        # skip connection
        residual = Conv2D(size, 1, strides=2, padding='same')(Z)
        residual = BatchNormalization()(residual)
        x += residual

        Z = x

     # Begin of middle flow
    for _ in range(num_middle_blocks) :
        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x += Z
        Z = x

    # Begin of exist flow
    x = SeparableConv2D(728, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = Activation('relu')(x)
    x = SeparableConv2D(1024, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D(3, strides=2, padding='same')(x)

    # skip connection
    residual = Conv2D(1024, 1, strides=2, padding='same')(Z)
    residual = BatchNormalization()(residual)
    x += residual

    x = Activation('relu')(x)
    x = SeparableConv2D(728, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = Activation('relu')(x)
    x = SeparableConv2D(1024, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = GlobalAveragePooling2D()(x)
    act = 'softmax'
    if num_classes == 1:
        act = 'sigmoid'
    output = Dense(num_classes, activation=act)(x)
    return output
```

[8]:

```
inputs = Input(shape=(32, 32, 2))
outputs = forward_pass(
    preprocess(inputs,
               mean=0.5,std=0.5,size=96),
    num_middle_blocks=8,num_classes=1
)
model = Model(inputs, outputs,name="xception")
```

[9]:

```
initial_epoch = 0
checkpoint_dir = "../models/tf-ckpts/{model}".format(model=model.name)
checkpoints = os.listdir(checkpoint_dir)
checkpoint = list(filter(lambda i : ".ckpt" in i, checkpoints))

checkpoint_path = checkpoint_dir+"/cp-{epoch}.ckpt"
```

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)


if len(checkpoint)>0:
    initial_epoch = sorted([int(ck.split(".")[0].split("-")[1]) for ck in␣
 ↪checkpoint])[-1]
    latest = tf.train.latest_checkpoint(checkpoint_dir)
    model.load_weights(latest)
```

[ ]:

```
model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(),␣
 ↪metrics=[tf.keras.metrics.AUC()])

history = model.fit(X_train, y_train, epochs=2, batch_size=64,␣
 ↪validation_data=(X_val, y_val),initial_epoch=initial_epoch)
```

```
Epoch 1/2
 750/5447 [===>…] - ETA: 3:41:03 - loss: 0.6539 - auc:
0.6587
```