train_transfer_learning

April 29, 2020

```
[ ]: from keras.layers import Input, Dense, Conv2D, MaxPool2D, Flatten, Dropout,␣
      ↪BatchNormalization, Activation, Add
     from keras.optimizers import Adam
     from keras.models import Model
     from keras.callbacks import ModelCheckpoint, EarlyStopping
     import keras
     import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.model_selection import train_test_split
     from keras.preprocessing.image import ImageDataGenerator
     from keras.applications.resnet50 import preprocess_input
     from keras.applications.resnet import ResNet50 as resnet50

     import os
     from tqdm import tqdm
     import json
     import glob
```

```
[ ]: import tensorflow as tf
     gpus= tf.config.experimental.list_physical_devices('GPU')
     tf.config.experimental.set_memory_growth(gpus[0], True)
```

```
[ ]: def resnet50_conv(shape):
         '''
             Load Resnet convolutional layers' and set the input image shape of␣
      ↪network.

             ======================================================================

             input:  A tupe containing 3 dimensions with input shape (H,W,C) RGB.

             returns: Model with convolutional layers only.
         '''
         input_tensor = Input(shape=shape)
         resnet_model_top = resnet50(include_top=True, weights='imagenet',␣
      ↪input_tensor=input_tensor)
         resnet_model_top.summary()
```

```
    return resnet_model_top
```

[ ]:

```
# Defines input shape and gets resnet50v1 backbone
input_shape = (224,224, 3)
(w, h, _) = input_shape
net = resnet50_conv(input_shape)
```

[ ]:

```
# Helps to see the name and position of each layer.
lay_nb = 0
for layer in net.layers:
    #print(layer)
    print(layer.name)
    #print(layer.output)
    lay_nb += 1
    lay_name = layer.name
#     if layer.name == 'conv4_block6_out':
#         break
    if layer.name == 'avg_pool':
        break
print(lay_nb)
print(lay_name)
```

[ ]:

```
def create_custom_model(net, optm, classes, questao6):
    '''
        Fine tunning implementation according to exercise 5 and 6.

        ========================================================================

        net:  Resnet50 pre-trained model

        optm: Optmizer used to train the network.

        classes: Number of classes in dataset.

        questao6: If wants to implement the Exercise 5 implementation put the␣
 ↪flag to False. Othewise,
        It will implement exercise 6 with custom fully-connected layers and␣
 ↪another convolutional block.


        returns: Model compiled.
    '''

    # Gets the average pooling layer.
    x = net.layers[-2].output
    # flatten + fully-connected + softmax activation
    if(questao6 == True):
        x = Dense(1024, activation='relu')(x)
        x = Dense(1024, activation='relu')(x)
        x = Dense(1024, activation='relu')(x)
    x = Dropout(0.25)(x)
    x = Dense(classes, activation='softmax')(x)

    custom_model = Model(net.input,x)

    # Froze layers
    for layer in custom_model.layers[:176]:
        if( "conv5_block3" in layer.name or "avg_pool" in layer.name):
            layer.trainable = True
        else:
            layer.trainable = False
        if(questao6 == True):
            if( "conv5_block2" in layer.name):
                layer.trainable = True

    # Compile it
    custom_model.compile(loss='categorical_crossentropy',
                       optimizer=optm,
                       metrics=['accuracy'])

    return custom_model
```

[ ]:

```
def create_custom_model_q7(net, optm, classes):

    '''
        Fine tunning implementation according to exercise 7. convolutional␣
  ↪layer 5 is implemented from
        scratch. Other layers before convolutional layer 5 are frozen.
        Custom dense layers are implemented also.

        ========================================================================

        net:  Resnet50 pre-trained model

        optm: Optmizer used to train the network.

        classes: Number of classes in dataset.

        returns: Model compiled.
    '''

    layer_dict = dict([(layer.name, layer) for layer in net.layers])
    conv4_block6_out = layer_dict['conv4_block6_out'].output

    # Custom Convolutional layers
    # custom_conv_block1
    momemtum=0.8
    activation_f = 'relu'
    # conv_block_1_1
    x = Conv2D(512, (1,1), strides=1, activation='linear',␣
 ↪padding='same')(conv4_block6_out)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    # conv_block_1_2
    x = Conv2D(512, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    # conv_block_1_0
    y = Conv2D(2048, (1,1), strides=1, activation='linear',␣
 ↪padding='same')(conv4_block6_out)
    y = BatchNormalization(momentum=momemtum)(y)
    # conv_block_1_3
    x = Conv2D(2048, (1,1), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Add()([y, x])
    custom_conv_block1_out = Activation(activation_f)(x)

    # custom_conv_block2
    x = Conv2D(512, (1,1), strides=1, activation='linear',␣
 ↪padding='same')(custom_conv_block1_out)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = Conv2D(512, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = Conv2D(2048, (1,1), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Add()([custom_conv_block1_out, x])
    custom_conv_block2_out = Activation(activation_f)(x)

    # custom_conv_block3
    x = Conv2D(512, (1,1), strides=1, activation='linear',␣
 ↪padding='same')(custom_conv_block2_out)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = Conv2D(512, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
```

```
    x = Conv2D(2048, (1,1), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Add()([custom_conv_block2_out, x])
    custom_conv_block3_out = Activation(activation_f)(x)

    #Global Aerage Pooling
    x = layer_dict['avg_pool'](custom_conv_block3_out)

    # custom fully-connected + softmax activation
    x = Dense(1024, activation='relu')(x)
    x = Dense(1024, activation='relu')(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.25)(x)
    x = Dense(classes, activation='softmax')(x)

    custom_model = Model(net.input,x)

    # Froze layers
    for layer in custom_model.layers[:143]:
            layer.trainable = False


    # Compile it
    custom_model.compile(loss='categorical_crossentropy',
                         optimizer=optm,
                         metrics=['accuracy'])

    return custom_model
```

[ ]:
```
lr = 0.5e-5
optm = Adam(learning_rate = lr, decay=0.001)
classes = 5
#custom_resnet =␣
 ↪create_custom_model(net,optm,lay_name,lay_nb,classes,questao6=False)
custom_resnet = create_custom_model_q7(net,optm,lay_name,lay_nb,classes)
```

[ ]:
```
custom_resnet.summary()
```

[ ]:
```
train_datagen = ImageDataGenerator(validation_split=0.2,␣
 ↪preprocessing_function=preprocess_input) # set validation split
```

[ ]:
```
#Defines the batch-size and keras generators
batch_size = 32

train_data_dir = 'Dataset/flower_photos/all'
```

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(w,h),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True, seed=42,
    subset='training') # set as training data

val_generator = train_datagen.flow_from_directory(
    train_data_dir, # same directory as training data
    target_size=(w,h),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True, seed=42,
    subset='validation') # set as validation data
```

[ ]:

```
# Gets the total number of images in dataset
filenames = train_generator.filenames
samples = len(filenames)
print(samples)
```

[ ]:

```
# Defines Early Stopping and sabe the best model during training before␣
 ↪overfitting.
file_name = 'best_model.h5'
checkpointer = ModelCheckpoint(file_name, monitor='val_accuracy',␣
 ↪save_best_only=True)
early_stop = EarlyStopping(monitor = 'val_accuracy', min_delta = 0.001,
mode = 'max', patience = 10)
callbacks=[checkpointer,early_stop]
```

[ ]:

```
# Defines number of epochs and train the model
epochs = 100
steps_in_epoch = samples // batch_size

history = custom_resnet.fit_generator(train_generator,␣
 ↪steps_per_epoch=steps_in_epoch, epochs=epochs,
                              validation_data=val_generator, ␣
 ↪validation_steps=1,
                              verbose=1,callbacks=callbacks)
```

[ ]:

```
def graph_training_history(history):
    acc_train = history['accuracy']
    acc_test = history['val_accuracy']
    loss_train = history['loss']
    loss_test = history['val_loss']
    plt.rcParams['axes.facecolor']='white'
    fig = plt.figure(1)
    # summarize history for accuracy
    plt.subplot(121)
    plt.plot(acc_train)
    plt.plot(acc_test)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.tight_layout()

    # summarize history for loss
    plt.subplot(122)
    plt.plot(loss_train)
    plt.plot(loss_test)
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper right')
    plt.tight_layout()

    plt.show()
    fig.savefig('weights/history.png', dpi=fig.dpi)
```

[ ]:

```
graph_training_history(history.history)
```

[ ]:

```
# Saving history
with open('weights/history_model.json', 'w') as f:
    json.dump(str(custom_resnet.history.history), f)
```

[ ]:

```
```