deep learning CNN Data augmentation to address overfitting

May 11, 2023

[5]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
import PIL
import tensorflow as tf


from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

[6]:
```
!pip install cv2
```

```
ERROR: Could not find a version that satisfies the requirement cv2 (from
versions: none)
ERROR: No matching distribution found for cv2
```

[3]:
```
!pip install opencv-python
```

```
Collecting opencv-python
  Downloading opencv_python-4.7.0.72-cp37-abi3-win_amd64.whl (38.2 MB)
Requirement already satisfied: numpy>=1.19.3 in
c:\users\rakesh\anaconda3\lib\site-packages (from opencv-python) (1.21.5)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.7.0.72
```

[4]:
```
import cv2
```

[7]:
```
import PIL
```

[12]:
```
dataset_url="https://storage.googleapis.com/download.tensorflow.org/
 ↪example_images/flower_photos.tgz"
data_dir=tf.keras.utils.get_file('flower_photos',origin=dataset_url,␣
 ↪cache_dir='.',untar=True)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/exa
mple_images/flower_photos.tgz
228813984/228813984 [==============================] - 24s 0us/step
```

[13]:
```
data_dir
```

[13]:
```
'.\\datasets\\flower_photos'
```

[14]:
```
import pathlib
data_dir=pathlib.Path(data_dir)
data_dir
```

[14]:
```
WindowsPath('datasets/flower_photos')
```

[15]:
```
list(data_dir.glob('*/*.jpg'))[:5]
```

[15]:
```
[WindowsPath('datasets/flower_photos/daisy/100080576_f52e8ee070_n.jpg'),
 WindowsPath('datasets/flower_photos/daisy/10140303196_b88d3d6cec.jpg'),
 WindowsPath('datasets/flower_photos/daisy/10172379554_b296050f82_n.jpg'),
 WindowsPath('datasets/flower_photos/daisy/10172567486_2748826a8b.jpg'),
 WindowsPath('datasets/flower_photos/daisy/10172636503_21bededa75_n.jpg')]
```

[17]:
```
image_count=len(list(data_dir.glob('*/*.jpg')))
image_count
```

[17]:
```
3670
```

[18]:
```
PIL.Image.open(str(roses[1]))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [18], in <cell line: 1>()
----> 1 PIL.Image.open(str(roses[1]))

NameError: name 'roses' is not defined
```

[19]:
```
roses=list(data_dir.glob('roses/*'))
roses[:5]
```

[19]:
```
[WindowsPath('datasets/flower_photos/roses/10090824183_d02c613f10_m.jpg'),
 WindowsPath('datasets/flower_photos/roses/102501987_3cdb8e5394_n.jpg'),
 WindowsPath('datasets/flower_photos/roses/10503217854_e66a804309.jpg'),
 WindowsPath('datasets/flower_photos/roses/10894627425_ec76bbc757_n.jpg'),
 WindowsPath('datasets/flower_photos/roses/110472418_87b6a3aa98_m.jpg')]
```

[85]:
```
PIL.Image.open(str(roses[2]))
```

[85]:

[25]:
```
tulips=list(data_dir.glob('tulips/*'))
tulips[:5]
```

[25]:
```
[WindowsPath('datasets/flower_photos/tulips/100930342_92e8746431_n.jpg'),
 WindowsPath('datasets/flower_photos/tulips/10094729603_eeca3f2cb6.jpg'),
 WindowsPath('datasets/flower_photos/tulips/10094731133_94a942463c.jpg'),
 WindowsPath('datasets/flower_photos/tulips/10128546863_8de70c610d.jpg'),
 WindowsPath('datasets/flower_photos/tulips/10163955604_ae0b830975_n.jpg')]
```

[29]:
```
PIL.Image.open(str(tulips[0]))
```

[29]:

Read flowers images from disk into numpy array using opencv

[31]:
```
flowers_images_dict={
    'roses':list(data_dir.glob('roses/*')),
     'daisy': list(data_dir.glob('daisy/*')),
    'dandelion': list(data_dir.glob('dandelion/*')),
    'sunflowers': list(data_dir.glob('sunflowers/*')),
    'tulips': list(data_dir.glob('tulips/*')),

}
```

[32]:
```
flower_label_dict={
    'roses':0,
    'daisy': 1,
    'dandelion': 2,
    'sunflowers': 3,
    'tulips': 4,
}
```

[33]:
```
flowers_images_dict['roses'][:5]
```

[33]:
```
[WindowsPath('datasets/flower_photos/roses/10090824183_d02c613f10_m.jpg'),
 WindowsPath('datasets/flower_photos/roses/102501987_3cdb8e5394_n.jpg'),
 WindowsPath('datasets/flower_photos/roses/10503217854_e66a804309.jpg'),
 WindowsPath('datasets/flower_photos/roses/10894627425_ec76bbc757_n.jpg'),
 WindowsPath('datasets/flower_photos/roses/110472418_87b6a3aa98_m.jpg')]
```

[34]:
```
str(flowers_images_dict['roses'][0])
```

[34]:
```
'datasets\\flower_photos\\roses\\10090824183_d02c613f10_m.jpg'
```

[35]:
```
img=cv2.imread(str(flowers_images_dict['roses'][0]))
```

[36]:
```
img.shape
```

[36]:
```
(240, 179, 3)
```

[37]:
```
cv2.resize(img,(180,180)).shape
```

[37]:
```
(180, 180, 3)
```

[40]:
```
x,y=[],[]
for flower_name, images in flowers_images_dict.items():
    for image in images:
        img=cv2.imread(str(image))
        resized_images=cv2.resize(img,(180,180))
        x.append(resized_images)
        y.append(flower_label_dict[flower_name])
```

[41]:
```
x=np.array(x)
y=np.array(y)
```

Train test split

[42]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)
```

Preprocessing: scale images

[44]:
```
x_train_scaled=x_train/255
x_test_scaled=x_test/255
```

Build convolutional neural network and train it

[51]:
```
num_classes=5


model=Sequential([
    layers.Conv2D(16,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),
     layers.Conv2D(16,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),
     layers.Conv2D(16,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),

    layers.Flatten(),
    layers.Dense(128,activation='relu'),
    layers.Dense(num_classes)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
  ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy']
              )

model.fit(x_train_scaled,y_train,epochs=30)
```

```
Epoch 1/30
86/86 [==============================] - 37s 415ms/step - loss: 1.3714 -
accuracy: 0.4033
Epoch 2/30
86/86 [==============================] - 36s 418ms/step - loss: 1.0256 -
accuracy: 0.6017
Epoch 3/30
86/86 [==============================] - 34s 396ms/step - loss: 0.8632 -
accuracy: 0.6686
Epoch 4/30
86/86 [==============================] - 34s 392ms/step - loss: 0.7067 -
accuracy: 0.7344
Epoch 5/30
86/86 [==============================] - 34s 397ms/step - loss: 0.5470 -
accuracy: 0.8063
Epoch 6/30
86/86 [==============================] - 34s 393ms/step - loss: 0.4038 -
accuracy: 0.8616
Epoch 7/30
86/86 [==============================] - 34s 392ms/step - loss: 0.2449 -
accuracy: 0.9182
Epoch 8/30
86/86 [==============================] - 34s 392ms/step - loss: 0.1765 -
accuracy: 0.9455
Epoch 9/30
86/86 [==============================] - 34s 394ms/step - loss: 0.1035 -
accuracy: 0.9677
Epoch 10/30
86/86 [==============================] - 34s 396ms/step - loss: 0.0758 -
accuracy: 0.9811
Epoch 11/30
86/86 [==============================] - 36s 413ms/step - loss: 0.0702 -
accuracy: 0.9833
Epoch 12/30
86/86 [==============================] - 35s 402ms/step - loss: 0.0434 -
accuracy: 0.9873
Epoch 13/30
86/86 [==============================] - 34s 391ms/step - loss: 0.0378 -
accuracy: 0.9916
```

```
Epoch 14/30
86/86 [==============================] - 34s 397ms/step - loss: 0.0170 -
accuracy: 0.9967
Epoch 15/30
86/86 [==============================] - 34s 392ms/step - loss: 0.0246 -
accuracy: 0.9949
Epoch 16/30
86/86 [==============================] - 34s 398ms/step - loss: 0.0276 -
accuracy: 0.9949
Epoch 17/30
86/86 [==============================] - 34s 390ms/step - loss: 0.0134 -
accuracy: 0.9978
Epoch 18/30
86/86 [==============================] - 34s 397ms/step - loss: 0.1131 -
accuracy: 0.9651
Epoch 19/30
86/86 [==============================] - 34s 391ms/step - loss: 0.0653 -
accuracy: 0.9797
Epoch 20/30
86/86 [==============================] - 34s 396ms/step - loss: 0.0319 -
accuracy: 0.9927
Epoch 21/30
86/86 [==============================] - 33s 381ms/step - loss: 0.0097 -
accuracy: 0.9989
Epoch 22/30
86/86 [==============================] - 34s 391ms/step - loss: 0.0124 -
accuracy: 0.9975
Epoch 23/30
86/86 [==============================] - 33s 382ms/step - loss: 0.0113 -
accuracy: 0.9982
Epoch 24/30
86/86 [==============================] - 34s 390ms/step - loss: 0.0082 -
accuracy: 0.9993
Epoch 25/30
86/86 [==============================] - 33s 382ms/step - loss: 0.0041 -
accuracy: 0.9993
Epoch 26/30
86/86 [==============================] - 36s 415ms/step - loss: 0.0057 -
accuracy: 0.9993
Epoch 27/30
86/86 [==============================] - 33s 387ms/step - loss: 0.0019 -
accuracy: 0.9996
Epoch 28/30
86/86 [==============================] - 34s 390ms/step - loss: 0.0032 -
accuracy: 0.9996
Epoch 29/30
86/86 [==============================] - 33s 381ms/step - loss: 0.0025 -
accuracy: 0.9996
```

```
Epoch 30/30
86/86 [==============================] - 34s 397ms/step - loss: 0.0058 -
accuracy: 0.9993
```

[51]:
```
<keras.callbacks.History at 0x1f2aba71a00>
```

[53]:
```
model.evaluate(x_test_scaled,y_test)
```

```
29/29 [==============================] - 4s 125ms/step - loss: 2.8721 -
accuracy: 0.6264
```

[53]:
```
[2.8721227645874023, 0.6263616681098938]
```

Here we see that while train accuracy is very high (99%), the test accuracy is significantly low (66.99%) indicating overfitting. Let's make some predictions before we use data augmentation to address overfitting

[55]:
```
predictions=model.predict(x_test_scaled)
predictions
```

```
29/29 [==============================] - 4s 137ms/step
```

[55]:
```
array([[  8.336534  ,  15.13717   ,   5.4003415 , -10.471546  ,
         -8.57661   ],
       [  8.395139  ,  -2.4289827 ,  -6.2180243 ,   1.9653721 ,
          5.3486357 ],
       [ -8.2515745 ,   2.6009653 ,  15.647339  , -11.676489  ,
         -5.8864093 ],
       …,
       [ -4.237256  ,   5.8607697 ,  -1.9202993 ,   6.85216   ,
         -0.16207433],
       [ -3.3413472 ,  -0.15840845,   3.7371826 ,  -3.348835  ,
          7.909586  ],
       [  3.9100323 , -13.481786  ,   2.1709642 ,   9.328533  ,
          5.725955  ]], dtype=float32)
```

[57]:
```
score=tf.nn.softmax(predictions[0])
```

[58]:
```
np.argmax(score)
```

[58]:
```
1
```

[59]:
```
y_test[0]
```

[59]:
```
1
```

0.0.1 Improve Test Accuracy Using Data Augmentation

[75]:
```
data_augmentation=keras.Sequential([
     layers.experimental.preprocessing.RandomFlip("horizontal",
                                                  input_shape=(img_height,
                                                               img_width,
                                                               3)),

    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.9)
])
```

[71]:
```
plt.axis('off')
plt.imshow(x[0])
```

[71]:
```
<matplotlib.image.AxesImage at 0x1f1067fcc70>
```

[77]:
```
plt.axis('off')
plt.imshow(data_augmentation(x)[0].numpy().astype("uint8"))
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
```

```
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
```

[77]: `<matplotlib.image.AxesImage at 0x1f106d46c10>`

[82]:
```
num_classes=5


model1=Sequential([
       data_augmentation,
        layers.Conv2D(18,3,padding='same',activation='relu'),
        layers.MaxPooling2D(),

    layers.Conv2D(32,3,padding='same',activation='relu'),
        layers.MaxPooling2D(),
    layers.Conv2D(64,3,padding='same',activation='relu'),
        layers.MaxPooling2D(),
```

```
    layers.Dropout(0.2),
    layers.Flatten(),

    layers.Dense(128,activation='relu')    ,
    layers.Dense(num_classes)
])

model1.compile(optimizer='adam',
              loss=tf.keras.losses.
  ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy']
)

model.fit(x_train_scaled,y_train,epochs=30)
```

```
Epoch 1/30
86/86 [==============================] - 40s 461ms/step - loss: 9.3052e-04 -
accuracy: 0.9996
Epoch 2/30
86/86 [==============================] - 49s 569ms/step - loss: 0.0013 -
accuracy: 0.9993
Epoch 3/30
86/86 [==============================] - 42s 487ms/step - loss: 9.2269e-04 -
accuracy: 0.9996
Epoch 4/30
86/86 [==============================] - 34s 398ms/step - loss: 0.0030 -
accuracy: 0.9993
Epoch 5/30
86/86 [==============================] - 34s 398ms/step - loss: 0.4573 -
accuracy: 0.8583
Epoch 6/30
86/86 [==============================] - 35s 412ms/step - loss: 0.1017 -
accuracy: 0.9709
Epoch 7/30
86/86 [==============================] - 33s 388ms/step - loss: 0.0358 -
accuracy: 0.9887
Epoch 8/30
86/86 [==============================] - 34s 393ms/step - loss: 0.0102 -
accuracy: 0.9978
Epoch 9/30
86/86 [==============================] - 33s 387ms/step - loss: 0.0033 -
accuracy: 0.9996
Epoch 10/30
86/86 [==============================] - 35s 403ms/step - loss: 0.0025 -
accuracy: 0.9996
Epoch 11/30
86/86 [==============================] - 34s 401ms/step - loss: 0.0018 -
```

```
accuracy: 0.9996
Epoch 12/30
86/86 [==============================] - 34s 396ms/step - loss: 0.0021 -
accuracy: 0.9993
Epoch 13/30
86/86 [==============================] - 33s 386ms/step - loss: 0.0019 -
accuracy: 0.9993
Epoch 14/30
86/86 [==============================] - 34s 392ms/step - loss: 0.0015 -
accuracy: 0.9996
Epoch 15/30
86/86 [==============================] - 34s 390ms/step - loss: 0.0014 -
accuracy: 0.9996
Epoch 16/30
86/86 [==============================] - 34s 391ms/step - loss: 0.0016 -
accuracy: 0.9996
Epoch 17/30
86/86 [==============================] - 33s 387ms/step - loss: 0.0014 -
accuracy: 0.9996
Epoch 18/30
86/86 [==============================] - 34s 391ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 19/30
86/86 [==============================] - 34s 390ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 20/30
86/86 [==============================] - 34s 401ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 21/30
86/86 [==============================] - 34s 394ms/step - loss: 0.0021 -
accuracy: 0.9993
Epoch 22/30
86/86 [==============================] - 34s 394ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 23/30
86/86 [==============================] - 33s 388ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 24/30
86/86 [==============================] - 34s 393ms/step - loss: 0.0015 -
accuracy: 0.9993
Epoch 25/30
86/86 [==============================] - 33s 387ms/step - loss: 0.0011 -
accuracy: 0.9996
Epoch 26/30
86/86 [==============================] - 34s 394ms/step - loss: 8.7374e-04 -
accuracy: 0.9996
Epoch 27/30
86/86 [==============================] - 33s 388ms/step - loss: 0.0011 -
```

```
accuracy: 0.9996
Epoch 28/30
86/86 [==============================] - 34s 395ms/step - loss: 0.0015 -
accuracy: 0.9993
Epoch 29/30
86/86 [==============================] - 33s 388ms/step - loss: 0.0013 -
accuracy: 0.9993
Epoch 30/30
86/86 [==============================] - 34s 394ms/step - loss: 0.0012 -
accuracy: 0.9993
```

[82]:
```
<keras.callbacks.History at 0x1f25e6182b0>
```

[83]:
```
model1.evaluate(x_test,y_test)
```

```
29/29 [==============================] - 6s 167ms/step - loss: 26.0198 -
accuracy: 0.1438
```

[83]:
```
[26.019821166992188, 0.1437908560037613]
```

[ ]:
```

```