Data Augmentation To Address Overfitting In Flower Classification CNN

In this notebook we will build a CNN to classify flower images. We will also see how our model overfits and how overfitting can be addressed using data augmentation. Data augmentation is a process of generating new training samples from current training dataset using transformations such as zoom, rotations, change in contrast etc

Credits: I used tensorflow offical tutorial: https://www.tensorflow.org/tutorials/images/classification as a reference and made bunch of changes to make it simpler

In below image, 4 new training samples are generated from original sample using different transformations

In [2]:
```
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
```

We will download flowers dataset from google website and store it locally. In below call it downloads the zip file (.tgz) in cache_dir which is . meaning the current folder

Load flowers dataset

In [3]:
```
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url,  cache_dir='
# cache_dir indicates where to download data. I specified . which means current dire
# untar true will unzip it
```

In [4]:
```
data_dir
```

Out[4]:
```
'.\\datasets\\flower_photos'
```

In [5]:
```
import pathlib
data_dir = pathlib.Path(data_dir)
data_dir
```

Out[5]:
```
WindowsPath('datasets/flower_photos')
```

In [6]:
```
list(data_dir.glob('*/*.jpg'))[:5]
```

Out[6]:
```
[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')]
```

In [7]:
```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
3670
```

In [8]:
```
roses = list(data_dir.glob('roses/*'))
roses[:5]
```

Out[8]:
```
[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')]
```

In [9]:
```
PIL.Image.open(str(roses[1]))
```

Out[9]:

In [10]:
```
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
```

Out[10]:

Read flowers images from disk into numpy array using opencv

In [9]:
```
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/*')),
}
```

In [10]:
```
flowers_labels_dict = {
    'roses': 0,
    'daisy': 1,
    'dandelion': 2,
    'sunflowers': 3,
    'tulips': 4,
}
```

In [13]:
```
flowers_images_dict['roses'][:5]
```

Out[13]:
```
[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')]
```

In [14]:
```
str(flowers_images_dict['roses'][0])
```

Out[14]:
```
'datasets\\flower_photos\\roses\\10090824183_d02c613f10_m.jpg'
```

In [11]:
```
img = cv2.imread(str(flowers_images_dict['roses'][0])) ## str because of cv2 transfo
```

In [12]:
```
img.shape
```

Out[12]:
```
(240, 179, 3)
```

In [17]:
```
cv2.resize(img,(180,180)).shape # o have the same size
```

Out[17]:
```
(180, 180, 3)
```

In [13]:
```
X, y = [], []

for flower_name, images in flowers_images_dict.items():
    for image in images:
        img = cv2.imread(str(image))
        resized_img = cv2.resize(img,(180,180))
        X.append(resized_img)
        y.append(flowers_labels_dict[flower_name])
```

In [14]:
```
X = np.array(X)
y = np.array(y)
```

Train test split

In [ ]:
```
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

In [ ]:
```
X_train_scaled = X_train / 255
X_test_scaled = X_test / 255
```

Build convolutional neural network and train it

In [22]:
```
num_classes = 5
model = Sequential([
  layers.Conv2D(16, 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.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 [==============================] - 162s 1s/step - loss: 1.3306 - accuracy: 0.4
419
Epoch 2/30
86/86 [==============================] - 94s 1s/step - loss: 1.0132 - accuracy: 0.61
19
Epoch 3/30
86/86 [==============================] - 94s 1s/step - loss: 0.8252 - accuracy: 0.68
93
Epoch 4/30
86/86 [==============================] - 94s 1s/step - loss: 0.6566 - accuracy: 0.75
40
Epoch 5/30
86/86 [==============================] - 94s 1s/step - loss: 0.4553 - accuracy: 0.83
79
Epoch 6/30
86/86 [==============================] - 93s 1s/step - loss: 0.2966 - accuracy: 0.90
19
Epoch 7/30
86/86 [==============================] - 93s 1s/step - loss: 0.1746 - accuracy: 0.94
26
Epoch 8/30
86/86 [==============================] - 93s 1s/step - loss: 0.1063 - accuracy: 0.97
09
Epoch 9/30
86/86 [==============================] - 93s 1s/step - loss: 0.0599 - accuracy: 0.98
51
Epoch 10/30
86/86 [==============================] - 93s 1s/step - loss: 0.0377 - accuracy: 0.98
98
Epoch 11/30
86/86 [==============================] - 92s 1s/step - loss: 0.0356 - accuracy: 0.99
31
```

```
Epoch 12/30
86/86 [==============================] - 92s 1s/step - loss: 0.0288 - accuracy: 0.99
24
Epoch 13/30
86/86 [==============================] - 92s 1s/step - loss: 0.0227 - accuracy: 0.99
71
Epoch 14/30
86/86 [==============================] - 92s 1s/step - loss: 0.0329 - accuracy: 0.99
09
Epoch 15/30
86/86 [==============================] - 92s 1s/step - loss: 0.0355 - accuracy: 0.98
98
Epoch 16/30
86/86 [==============================] - 92s 1s/step - loss: 0.0403 - accuracy: 0.98
87
Epoch 17/30
86/86 [==============================] - 92s 1s/step - loss: 0.0495 - accuracy: 0.98
55
Epoch 18/30
86/86 [==============================] - 92s 1s/step - loss: 0.0401 - accuracy: 0.98
69
Epoch 19/30
86/86 [==============================] - 92s 1s/step - loss: 0.0527 - accuracy: 0.98
26
Epoch 20/30
86/86 [==============================] - 92s 1s/step - loss: 0.0422 - accuracy: 0.98
80
Epoch 21/30
86/86 [==============================] - 92s 1s/step - loss: 0.0094 - accuracy: 0.99
78
Epoch 22/30
86/86 [==============================] - 92s 1s/step - loss: 0.0037 - accuracy: 0.99
96
Epoch 23/30
86/86 [==============================] - 93s 1s/step - loss: 0.0029 - accuracy: 0.99
96
Epoch 24/30
86/86 [==============================] - 92s 1s/step - loss: 0.0018 - accuracy: 0.99
96
Epoch 25/30
86/86 [==============================] - 92s 1s/step - loss: 0.0036 - accuracy: 0.99
93
Epoch 26/30
86/86 [==============================] - 92s 1s/step - loss: 0.0020 - accuracy: 0.99
96
Epoch 27/30
86/86 [==============================] - 92s 1s/step - loss: 0.0043 - accuracy: 0.99
93
Epoch 28/30
86/86 [==============================] - 93s 1s/step - loss: 0.0036 - accuracy: 0.99
93
Epoch 29/30
86/86 [==============================] - 92s 1s/step - loss: 0.0025 - accuracy: 0.99
93
Epoch 30/30
86/86 [==============================] - 92s 1s/step - loss: 0.0034 - accuracy: 0.99
93
```

Out[22]:
```
<keras.callbacks.History at 0x1655680b130>
```

In [23]:
```
model.evaluate(X_test_scaled,y_test)
```

```
29/29 [==============================] - 44s 328ms/step - loss: 2.5548 - accuracy:
```

Out[23]:
```
0.6558
[2.5548229217529297, 0.655773401260376]
```

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

In [24]:
```
predictions = model.predict(X_test_scaled)
predictions
```

Out[24]:
```
29/29 [==============================] - 13s 314ms/step
array([[ -8.034836  ,  25.752117  ,  -1.1941547 , -18.619875  ,
        -11.227778  ],
       [ 10.011834  ,  -3.6381605 , -11.387341  ,  -5.632186  ,
          2.2027192 ],
       [ -7.209098  ,   4.971569  ,  18.343872  , -13.315592  ,
         -5.322657  ],
       ...,
       [  3.8395944 ,  -7.615803  , -13.429502  ,  12.253704  ,
          9.793321  ],
       [  4.1103244 ,  -0.11814304,  -9.373617  ,  -3.000748  ,
         -0.36116636],
       [  1.9852644 , -13.491437  ,   0.55421954,   9.249021  ,
          7.549555  ]], dtype=float32)
```

In [25]:
```
score = tf.nn.softmax(predictions[0])
```

In [26]:
```
np.argmax(score)
```

Out[26]:
```
1
```

In [27]:
```
y_test[0]
```

Out[27]:
```
1
```

Improve Test Accuracy Using Data Augmentation

In [ ]:
```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

Original Image

In [ ]:
```
plt.axis('off')
plt.imshow(X[0])
```

Newly generated training sample using data augmentation

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

Train the model using data augmentation and a drop out layer

In [ ]:
```
num_classes = 5

model = Sequential([
  data_augmentation,
  layers.Conv2D(16, 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)
])

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

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

In [ ]:
```
model.evaluate(X_test_scaled,y_test)
```

You can see that by using data augmentation and drop out layer the accuracy of test set predictions is increased to 73.74%

In [ ]:
```

```