|  |  |
| --- | --- |
| **DATE** | **10-05-2023** |
| **TEAM ID** | **NM2023TMID06904** |
| **CODE** | **PYTHON** |

**1**.**Data Augmentation**

Insufficient learning examples prevent you from training a model that can generalize to new data, which leads to overfitting. If you had unlimited data, your model would be exposed to all characteristics of the current data distribution, preventing overfitting. By increasing the samples with different random changes that produce realistic-looking images, ***data augmentation*** uses the existing training samples to generate more training data. Your model should never view the same image twice during training. This makes the model more generic and exposes the other features of the data.

This is possible with Keras by defining a variety of stochastic transforms to be applied to the images with the *ImageDataGenerator* function. Let’s begin with an illustration.

* *rotation:* This is a range with which the images are rotated randomly. Its capacity lies from (0-180) degrees.
* *width\_shift and height\_shift:* ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
* *shear:* is for randomly applying shearing transformations.
* *zoom:*is for zooming the images randomly.
* *horizontal\_flip:*is for randomly flipping half the images horizontally
* *fill\_mode:* is the method used to fill in newly produced pixels that may arise following a rotation or width/height change.

**2.BUILT THE CNN MODEL**

####-----Let's display some randomly augmented training images-------####

from keras.preprocessing import image

fnames = [os.path.join(train\_cats\_dir, fname) for fname in os.listdir(train\_cats\_dir)]

img\_path = fnames[3]

img = image.load\_img(img\_path, target\_size=(150, 150))

x = image.img\_to\_array(img)

x = x.reshape((1,) + x.shape)

i = 0

for batch in datagen.flow(x, batch\_size=1):

  plt.figure(i)

  imgplot = plt.imshow(image.array\_to\_img(batch[0]))

  i += 1

if i % 4 == 0:

break

plt.show()



import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from \_\_future\_\_ import print\_function

import keras

from keras.models import Sequential

from keras.utils import to\_categorical

from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten

**A.INPUT LAYER**

print('Training data shape : ', train\_images.shape, train\_labels.shape)

print('Testing data shape : ', test\_images.shape, test\_labels.shape)

# Find the unique numbers from the train labels

classes = np.unique(train\_labels)

nClasses = len(classes)

print('Total number of outputs : ', nClasses)

print('Output classes : ', classes)

plt.figure(figsize=[4,2])

**B.Minimum 1 Convolution & 1 Pooling layer**

# Display the first image in training data

plt.subplot(121)

plt.imshow(train\_images[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(train\_labels[0]))

**C.1 Flatten layer**

# Display the first image in testing data

plt.subplot(122)

plt.imshow(test\_images[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(test\_labels[0]))

**OUTPUT LAYER**



**D.Minimum of 2 Hidden layers**

|  |
| --- |
| train\_data /= 255 |
|  | test\_data /= 255 |
|  |  |
|  | train\_labels\_one\_hot = to\_categorical(train\_labels) |
|  | test\_labels\_one\_hot = to\_categorical(test\_labels) |
|  |  |

**print('Original label 0 : ', train\_labels[0])**

**print('After conversion to categorical ( one-hot ) : ', train\_labels\_one\_hot[0])**

**E.OUTPUT LAYER:**

Original label 0 : [6]  
After conversion to categorical ( one-hot ) :   
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

**3.TEST THE MODEL**

**import matplotlib.pyplot as plt**

**import matplotlib.image as mpimg**

**import seaborn as sns**

**import numpy as np**

**import pandas as pd**

**import re**

**import os**

**import glob**

**import cv2**

***TEST***

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import\***

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import\***

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.feature\_selection import VarianceThreshold**

**from sklearn.pipeline import make\_pipeline**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**from sklearn.decomposition import TruncatedSVD**

**from skimage import data, color**

**from skimage.transform import rescale, resize, downscale\_local\_mean**

**import matplotlib.image as mpimg**

**from sklearn.linear\_model import\***

**from sklearn.preprocessing import\***

**from sklearn.ensemble import\***

**from sklearn.neighbors import\***

**from sklearn import svm**

**from sklearn.naive\_bayes import\***

**import xgboost as xgb**

***#deep learning libraries***

**import tensorflow as tf**

**from tensorflow.keras import Sequential**

**from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten,Dropout,BatchNormalization**

**from tensorflow.keras.optimizers import Adam**

**print(tf.\_\_version\_\_)**

***# for confusion matrix plotting***

**from mlxtend.plotting import plot\_confusion\_matrix**

**from sklearn.metrics import multilabel\_confusion\_matrix,confusion\_matrix**

**2.3.1**

**In [5]:**

**os.listdir('/kaggle/input/african-wildlife/buffalo')[:7]**

**Out[5]:**

**['361.txt', '208.jpg', '029.jpg', '245.txt', '014.jpg', '141.txt', '372.txt']**

**In [6]:**

**data=open('/kaggle/input/african-wildlife/buffalo/184.txt')**

**data.read()**

OUTPUT :

**'0 0.482284 0.471683 0.821334 0.824703\n'**

**TESTING THE MODEL :**

**In [7]:**

**k=0**

**for i in os.listdir('/kaggle/input/african-wildlife/buffalo/'):**

**if i[-3:] !='txt':**

**img=mpimg.imread('/kaggle/input/african-wildlife/buffalo/'+i)**

**plt.imshow(img)**

**plt.show()**

**k+=1**

**if k==3:**

**break**





