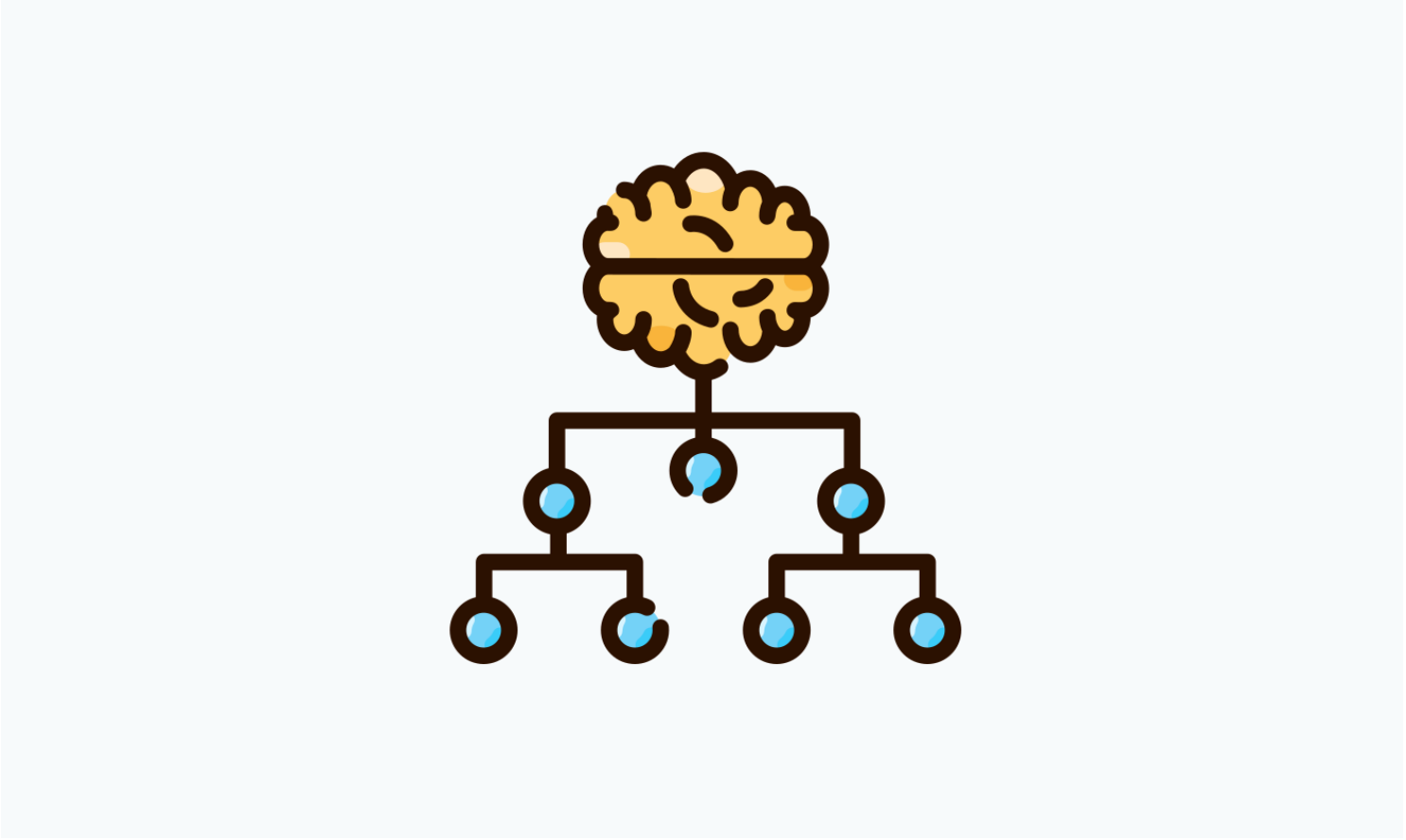
khuntiarishab 2 hours ago 3 min read

IMAGE CLASSIFIER : VEHICLES IMAGE DATA



The goal is to improve the accuracy of image classification.

WHAT IS IMAGE CLASSIFICATION?

Image classification is the process of taking an input like a picture and outputting a class(like "motorbike") or a probability that the input is a particular class("there is a 90% probability that this input is a motorbike").

Types of Images used in my Classifier :

There are three kinds of vehicles used in the image classifier.

ALGORITHM USED : CONVOLUTIONAL NEURAL NETWORK(CNN)

A convolution neural network (CNN) is a type of deep neural network that is applied to process visual imagery. These neural networks are modeled after the human visual system, consisting of networks of neurons.
A convolutional neural network is made of two main layers - the input and output layers, as well as several hidden layers (A neural layer is a stack of neurons in a single line). An input is received by a neuron in the input layer, the neuron processes it and does some computation on it, then transfers a non-linear function called activation function to yield a final output of a neuron.
Neurons are named by the activation functions they use; for example, **sigmoid neuron**, **RELU neuron**, and **TanH neurons**. Each of these neurons may form connections with multiple neurons, with each connections having a different weight.

Types of Layers in a Convolutional Network

As noted earlier, a neural layer is a collection of neurons in a single line. And all neurons in one layer do the same type of mathematical operations, by which the layer is named. The following are the popular kinds of layers in a convolutional neural network.

Convolutional Layers

All neurons in a convolutional layer apply the convolution operation to the inputs they receive. The common hyper parameters of a convolutional layer are:

- Filter size
- Stride

For example, let's use a layer with filter size 5*5*3. Assuming the input transmitted to the convolution neuron is an image of size 40*40 with 3 channels, to calculate the convolution (dot product) with our filter, the third dimension of the filter must be equal to the number of channels in the input.

Pooling Layers

The pooling layers are used to reduce the size of inputs, an action which speeds up the speed of processing and analyzing the input. Pooling layers are typically used after convolutional layers to reduce the spatial size (width and height) of the input. This reduces the number of parameters and, hence, the computation. The hyperparameters of a pooling layer include:

- Filter size
- Stride
- Max or average pooling

Fully Connected Layer

Fully connected layers are so-called because they connect every neuron in one layer to every neuron in another. In fully connected layers, each output dimension is in tandem with each input dimension.

SOURCE CODE EXPLANATION

1) Importing all the libraries

- Declaring all the required libraries used in the code and installation (if required by the environment).

[74]:

```
import os
print(os.listdir('../input/caltech101-airplanes-motorbikes-schooners/caltech101_classification'))


#Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#importing sklearn libraries and its dependencies
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix,roc_curve
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#importing tensorflow libraries and kera dependencies
import tensorflow
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
```

```
#COnfiguration
# Putting matplotlib to inline and setting the style
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

from keras.preprocessing.image import ImageDataGenerator

# libraries needed for CNN model
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
import tensorflow as tf
import random as rn

import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image


import time
```

2) Extracting Images

- Taking the images as input, reading it and storing it as a list parameter.

```
Add a code cell
X=[]
Z=[]
IMG_SIZE=256
Motorbikes_class='../input/caltech101-airplanes-motorbikes-schooners/caltech101_classification/Motorbikes'

Airplanes_class='../input/caltech101-airplanes-motorbikes-schooners/caltech101_classification/airplanes'

Schooner_class='../input/caltech101-airplanes-motorbikes-schooners/caltech101_classification/schooner'

def Label(img,Type_of_vehicle):
    return Type_of_vehicle
```

3) Training the Data

Creating individual function to train the data for each vehicle class.

Here I have created 4 test cases to test the data using different hyper parameters.

Test case 1

```
#TEST CASE 1
#Creating the CNN model and mentioning all parameters

model = Sequential()
#1st layer
model.add(Conv2D(filters = 32, kernel_size = (3,3),activation ='relu', input_shape = (256,256,3)))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))


#2nd layer
model.add(Conv2D(filters = 64, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


#3rd layer
model.add(Conv2D(filters =64, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

```
Epoch 1/3
16/16 [==============================] - 61s 4s/step - loss: 2.7231 - accuracy: 0.5325 - val_loss: 0.7476 - val_accurac
y: 0.7596
Epoch 2/3
16/16 [==============================] - 60s 4s/step - loss: 0.6094 - accuracy: 0.7663 - val_loss: 0.3821 - val_accurac
y: 0.8630
Epoch 3/3
16/16 [==============================] - 60s 4s/step - loss: 0.3154 - accuracy: 0.8924 - val_loss: 0.2517 - val_accurac
y: 0.9303
```

+ Code + Markdown

]:

```
#Printing accuracy
test_acc1=model.evaluate(x_test, y_test)[1]*100
print("The test 1 accuracy: {0:.2f}%".format(test_acc1))
```

```
13/13 [==============================] - 3s 248ms/step - loss: 0.2517 - accuracy: 0.9303
The test 1 accuracy: 93.03%
```

Test case 2

```
#TEST CASE 2
#Creating the CNN model and mentioning all parameters

model = Sequential()

#1st layer
model.add(Conv2D(filters = 32, kernel_size = (3,3),activation ='relu',input_shape = (256,256,3)))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


#2nd layer
model.add(Conv2D(filters =64, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


#3rd layer
model.add(Conv2D(filters =128, kernel_size = (3,3),activation ='softmax'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

```
#Making the prediction using the training data

Predictions = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                                  epochs = epochs, validation_data = (x_test,y_test))
```

```
Epoch 1/5
16/16 [==============================] - 73s 4s/step - loss: 1.1470 - accuracy: 0.4964 - val_loss: 0.8807 - val_accurac
y: 0.4952
Epoch 2/5
16/16 [==============================] - 71s 4s/step - loss: 0.8884 - accuracy: 0.5133 - val_loss: 0.7665 - val_accurac
y: 0.6755
Epoch 3/5
16/16 [==============================] - 73s 4s/step - loss: 0.6373 - accuracy: 0.7390 - val_loss: 0.5432 - val_accurac
y: 0.8077
Epoch 4/5
16/16 [==============================] - 71s 4s/step - loss: 0.5206 - accuracy: 0.8145 - val_loss: 0.4492 - val_accurac
y: 0.8173
Epoch 5/5
16/16 [==============================] - 72s 4s/step - loss: 0.4494 - accuracy: 0.8345 - val_loss: 0.4698 - val_accurac
y: 0.8293
```

+ Code + Markdown

```
#Printing accuracy
test_acc2=model.evaluate(x_test, y_test)[1]*100
print("The test 2 accuracy: {0:.2f}%".format(test_acc2))
```

```
13/13 [==============================] - 5s 349ms/step - loss: 0.4698 - accuracy: 0.8293
The test 2 accuracy: 82.93%
```

Test case 3:

```
#TEST CASE 3
#Creating the CNN model and mentioning all parameters

model = Sequential()
#1st layer
model.add(Conv2D(filters = 32, kernel_size = (5,5),activation ='relu', input_shape = (256,256,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

#2nd layer
model.add(Conv2D(filters = 64, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

#3rd layer
model.add(Conv2D(filters =128, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

#4th layer
model.add(Conv2D(filters =256, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

#5th layer
model.add(Conv2D(filters =512, kernel_size = (3,3),activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

```
#Printing accuracy
test_acc3=model.evaluate(x_test, y_test)[1]*100
print("The test 3 accuracy: {0:.2f}%".format(test_acc3))
```

```
13/13 [==============================] - 7s 544ms/step - loss: 0.1581 - accuracy: 0.9543
The test 3 accuracy: 95.43%
```

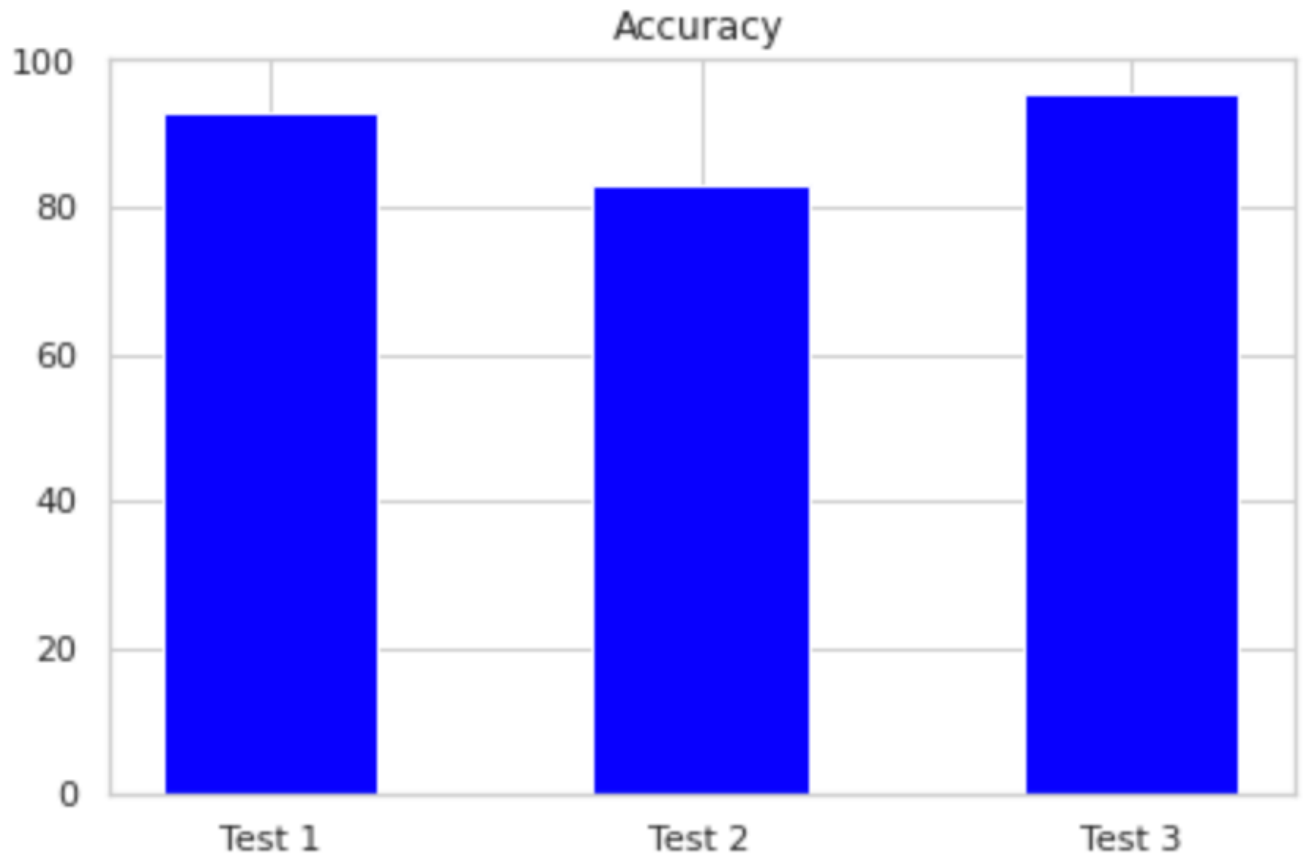
4) CONTRIBUTION

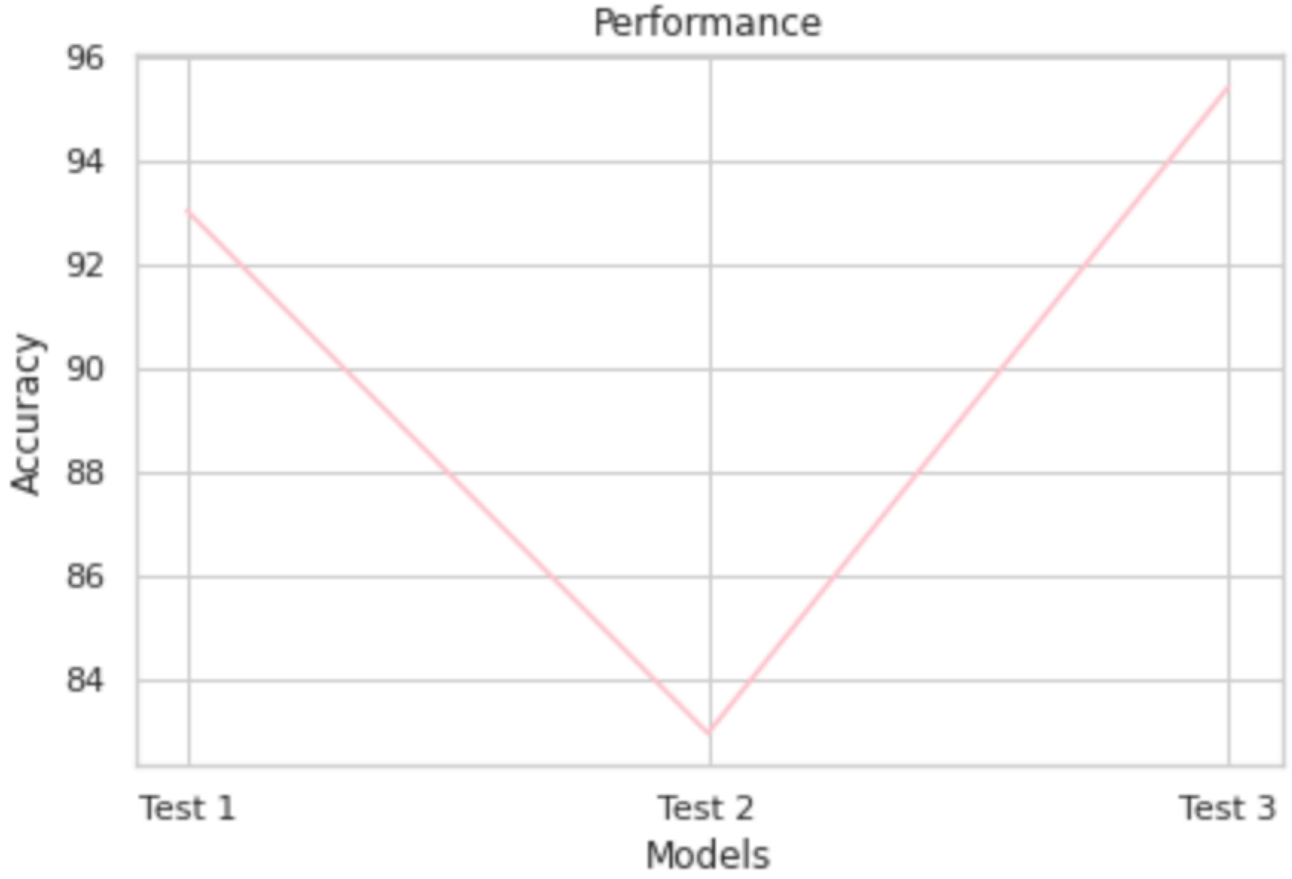
I have referred sources from kaggle and did changes to my code in which there are some parts to which I contributed.
Changing the various hyper parameters made me achieve the respective accuracy.
In the CNN model, adding batch normalization and drop out function was making my accuracy decrease drastically. By making combinations of all hyper parameters combined I finally reached the following test cases which gave good accuracy.
The first experiment I took 3 layers having epoch = 3 and batch size = 80
Accuracy = 93.03 %
The second experiment I took 3 layers having epoch = 5 and batch size = 80
Accuracy = 82.93 %. The decrease might be due to data and training overload.

The third experiment I took 3 layers having epoch = 10 and batch size = 100
Accuracy = 95.43 %

RESULTS

The final accuracy reached is 95.43%.





REFERENCE

1) https://www.kaggle.com/datasets/maricinnamon/caltech101-airplanes-motorbikes-schooners

2) https://www.clarifai.com/blog/what-is-convolutional-networking?hs_amp=true&utm_term=&utm_campaign=DSA-Community&utm_source=adwords&utm_medium=ppc&hsa_acc=4305946045&hsa_cam=18142553015&hsa_grp=141361868638&hsa_ad=618056207992&hsa_src=g&hsa_tgt=dsa-19959388920&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjwhsmaBhCvARIsAIbEbH4cdGgD4dqFdCY2zDPBP0dq8kXSU7fuptsBvjvT_nq204MtW5eyHDEaAlxZEALw_wcB