IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITIONS

AI & ADS :

As we venture deeper into our AI marketing Miami journey, let's decipher the role of AI in image recognition. You might be wondering, how does AI make image recognition possible? The magic lies in Machine Learning (ML) and Deep Learning (DL), two subsets of AI that breathe life into image recognition.

Let's paint a picture—imagine ML as an enthusiastic student. It learns from a dataset of images, recognizing patterns and learning to identify different objects. However, this student is a quick learner and soon becomes adept at making accurate identifications based on their training.

Enter DL, ML's more sophisticated sibling.DL uses neural networks modeled after the human brain to process data. When it comes to image recognition, DL can identify an object and understand its context. Is that a cat on the sofa or a picture of a cat on a poster? DL knows the difference.

AI has significantly improved the accuracy and efficiency of image recognition. By learning from vast amounts of data, AI can recognize objects with impressive accuracy, and the more data it has, the better it gets!

Step 1:

Identify Your Goals - Defining what you want to achieve with AI image recognition is critical. Whether monitoring brand visibility, understanding customer preferences, or enhancing content relevancy, your objectives should guide your implementation strategy.

Step 2:

Choose the Right AI Tool - Various AI platforms offer image recognition capabilities. Tools like Google Vision AI, Amazon Rekognition, or IBM Watson Visual Recognition provide robust image analysis features. Choose one that aligns with your goals and budget.

Step 3:

Gather and Label Your Images - AI needs training with a vast set of labeled images to learn and recognize images. Be meticulous in this phase; the quality of your training data can significantly impact your results.

Step 4:

Train Your AI Model - With your data ready, the next step is to train your AI model. This might sound technical, but most tools have user-friendly interfaces that guide you through the process.

Step 5:

Test and Refine - Finally, test your AI model, assess its performance, and refine it as needed. AI learning is an iterative process, and continual refinement will yield optimal results.

DAC :

There are several DAC architectures; the suitability of a DAC for a particular application is determined by figures of merit including: resolution, maximum sampling frequency and others. Digital-to-analog conversion can degrade a signal, so a DAC should be specified that has insignificant errors in terms of the application.

DACs are commonly used in music players to convert digital data streams into analog audio signals. They are also used in televisions and mobile phones to convert digital video data into analog video signals. These two applications use DACs at opposite ends of the frequency/resolution trade-off. The audio DAC is a low-frequency, high-resolution type while the video DAC is a high-frequency low- to medium-resolution type.

Due to the complexity and the need for precisely matched components, all but the most specialized DACs are implemented as integrated circuits (ICs). These typically take the form of metal–oxide–semiconductor (MOS) mixed-signal integrated circuit chips that integrate both analog and digital circuits.

Discrete DACs (circuits constructed from multiple discrete electronic components instead of a packaged IC) would typically be extremely high-speed low-resolution power-hungry types, as used in military radar systems. Very high-speed test equipment, especially sampling oscilloscopes, may also use discrete DACs.

DACs and ADCs are part of an enabling technology that has contributed greatly to the digital revolution. To illustrate, consider a typical long-distance telephone call. The caller's voice is converted into an analog electrical signal by a microphone, then the analog signal is converted to a digital stream by an ADC. The digital stream is then divided into network packets where it may be sent along with other digital data, not necessarily audio. The packets are then received at the destination, but each packet may take a completely different route and may not even arrive at the destination in the correct time order. The digital voice data is then extracted from the packets and assembled into a digital data stream. A DAC converts this back into an analog electrical signal, which drives an audio amplifier, which in turn drives a loudspeaker, which finally produces sound.

IOT :

The image recognition landscape in IoT applications is rapidly changing. Significant advances in mobile processing power, edge computing, and machine learning are paving the way for the widespread use of image recognition in many IoT applications. For example, omnipresent mobile devices (which are a key component in many IoT applications) equipped with high-resolution cameras facilitate the generation of images and videos by everyone, everywhere.

Moreover, intelligent video cameras, such as IP cameras and Raspberry Pis with cameras, are used in many places, such as smart homes, campuses, and factories, for different applications. Many IoT applications—including smart cities, smart homes, smart health, smart education, smart factories, and smart agriculture—make decisions using image recognition/classification.

CAD :

If you're familiar with Scan2CAD, you'll know that its key function is transforming raster and vector designs into CAD/CAM or CNC files. There are many parts of the conversion process to achieve this. One of the most complex and interesting is image recognition.

Though 'image recognition' can have multiple meanings, within the context of Scan2CAD it refers to the process of recognising and transforming elements within a raster or vector image to their appropriate elements. This includes distinguishing circles from arcs, text from lines, and so on. As you may be able to guess, this is one of the trickiest tasks for Scan2CAD to accomplish—and it's also what sets us apart from other vectorization software.

In this post, we'll run through what object recognition means in practical terms. We'll also show you how it helps to improve your vectorization results—and what your files would look like without it.

Before we start exploring object recognition in earnest, we need to explore the differences between detection and recognition.

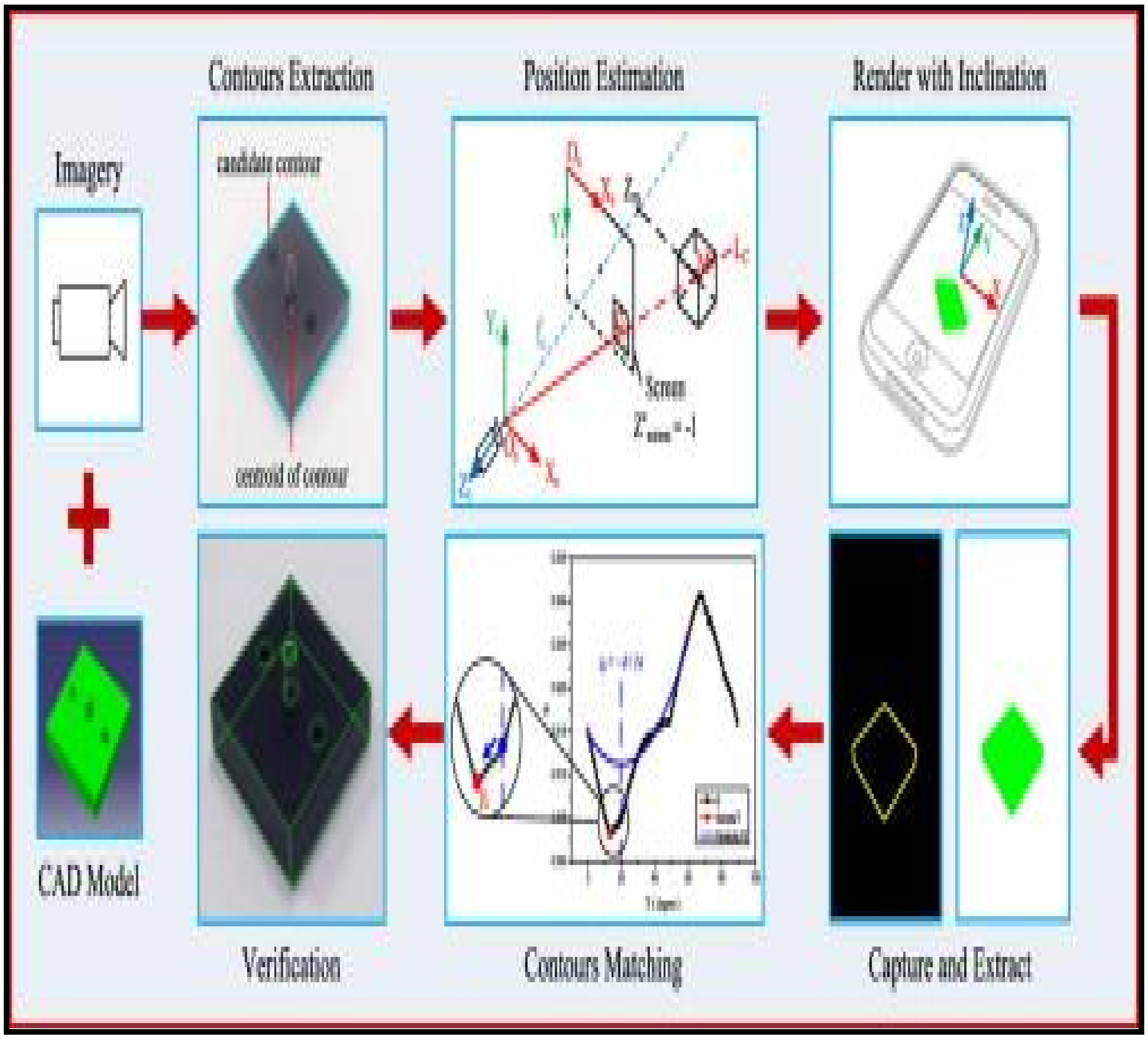
In certain contexts, detection is sufficient. It is not, however, conducive to good results when vectorizing for CAD. Vectorization, after all, is about transforming a raster image into a usable vector file—and it's the word usable which is key here.

It would be possible, for example, to simply convert a raster image into a series of vector polygons and lines. This may be fine if, for example, you simply wished to vectorize a logo for printing.

However, this would not satisfy the requirements of someone who wished to edit their design in CAD software. Take, for instance, the example of exploded vector text. Though this may initially seem to be a vector representation of raster text, it is not editable and not useful.

In certain contexts, detection is sufficient. It is not, however, conducive to good results when vectorizing for CAD. Vectorization, after all, is about transforming a raster image into a usable vector file—and it's the word usable which is key here.

However, this would not satisfy the requirements of someone who wished to edit their design in CAD software.



PROGRAM:

```
# These steps are to be followed when using google colab
#and importing data from kaggle
from google.colab import files
# Install Kaggle library
!pip install -q kaggle
from google.colab import files
#upload the kaggle.json file
uploaded = files.upload()
#make a diectoryin which kajggle.json is stored
# ! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
#download the dataset into the colab
!kaggle datasets download -d alessiocorrado99/animals10
#unzip the data
!unzip /content

#Incase you are using a local machine, start from here.
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import Sequential,Model
from tensorflow.keras.layers import BatchNormalization,Dropout,Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing import image
import numpy as np
import os
import cv2

train_data_dir='/kaggle/input/animals10/raw-img/'
img_height=128
img_width=128
batch_size=64
nb_epochs=20
train_datagen = ImageDataGenerator(rescale=1./255,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   validation_split=0.2) # set validation split

train_generator = train_datagen.flow_from_directory(
   train_data_dir,
   target_size=(img_height, img_width),
   batch_size=batch_size,
   class_mode='categorical',
   subset='training') # set as training data

validation_generator = train_datagen.flow_from_directory(
   train_data_dir, # same directory as training data
   target_size=(img_height, img_width),
   batch_size=batch_size,
   class_mode='categorical',
   subset='validation') # set as validation data


model = Sequential()
inputShape = (128, 128, 3)
model.add(Conv2D(64, (3, 3), padding="same", activation='relu',
input_shape=inputShape))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 5, strides=2, padding='same', activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.4))
model.add(Conv2D(64, kernel_size = 5, strides=2, padding='same', activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))
model.summary()
#compile the model
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
#train the model,this step takes alot of time (hours)
generator(
   train_generator,
   steps_per_epoch = train_generator.samples // batch_size,
   validation_data = validation_generator,
   validation_steps = validation_generator.samples // batch_size,
   epochs = nb_epochs)
#save the model for later use
model.save('path\name of model')
```

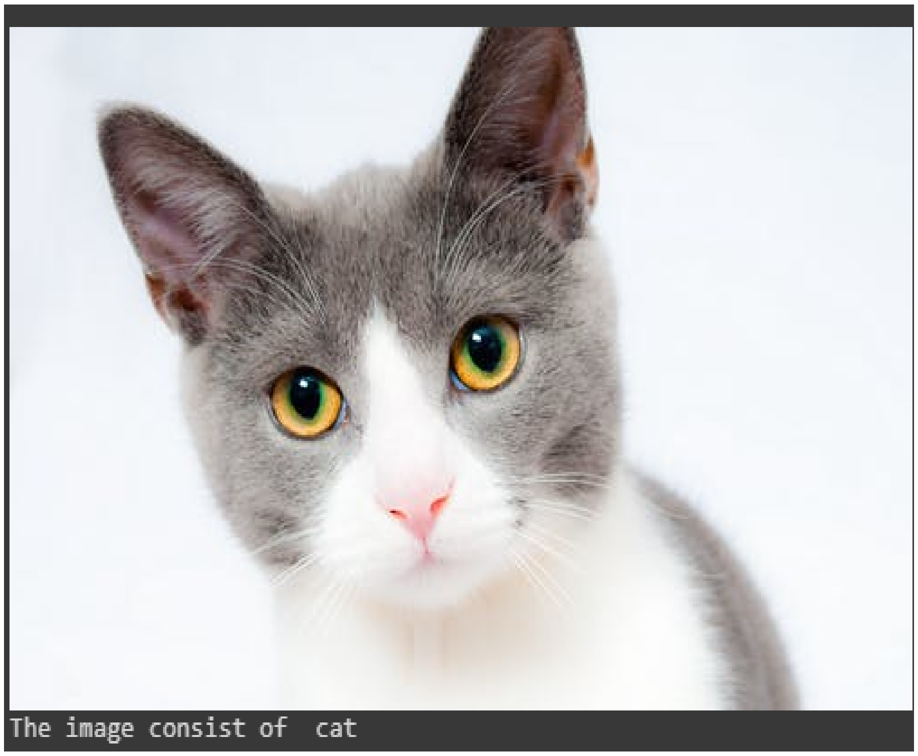
```
#order of the animals array is important
#animals=["dog", "horse","elephant", "butterfly",  "chicken",  "cat", "cow",
"sheep","spider", "squirrel"]
bio_animals=sorted(os.listdir('/content/raw-img'))
categories = {'cane': 'dog', "cavallo": "horse", "elefante": "elephant", "farfalla":
"butterfly", "gallina": "chicken", "gatto": "cat", "mucca": "cow", "pecora": "sheep",
"scoiattolo": "squirrel","ragno":"spider"}
def recognise(pred):
  animals=[categories.get(item,item)  for item in bio_animals]
  print("The image consist of ",animals[pred])

from tensorflow.keras.preprocessing import image
import numpy as np
img = image.load_img( target_size=(128, 128))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
prediction=model.predict(x)
# prediction

recognise(np.argmax(prediction))

test_data_path="/content/test data/test_animals"
files=sorted(os.listdir(test_data_path))
files=files[1:]
for img in files:
  x=cv2.imread(os.path.join(test_data_path,img))
  cv2_imshow(x)
  recognise(np.argmax(predict[files.index(img)]))
  print("")
```

OUTPUT :



```
The image consist of  cat
```

THANK YOU