

**GOVERNMENT COLLEGE OF ENGINEERING BARGUR**

**(AUTONOMOUS)**

**IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION**

**TEAM MEMBERS:**

Swetha.S

Sruthi.M.P

Kiruthika.P

Jenifer Suganthi.S

Latha Mangeskar.M.S

**PROBLEM STATEMENT:**

Design and implement an image recognition system capable of accurately identifying and categorizing objects classification system that accurately identifies and categorizes images of animals into one of ten predefined classes. The goal is to develop a deep learning model that can distinguish between various animals, including dogs, horses, elephants, butterflies, chickens, cats, cows, sheep, spiders, and squirrels. This system will be able to handle single images as well as batches of test images, providing predictions with associated animal labels. The model should be trained to achieve a high level of accuracy in recognizing and classifying animal images.

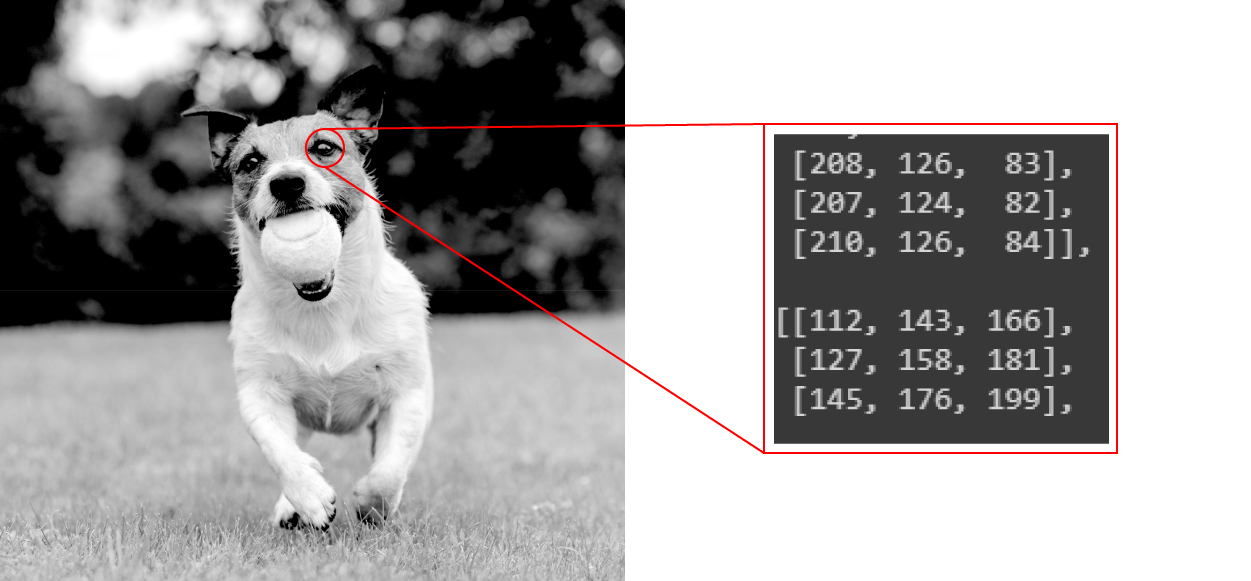
**ABSTRACT:**

The rapid development of computers makes people’s production and life rich and colorful, and people communicate with each other in the world of the Internet. The daily downloads and uploads of network pictures are countless. The existing image recognition technology alone cannot meet the currently required functions, so technology is needed to meet the retrieval requirements. The purpose of this paper is to study the image recognition technology based on the computer platform. This paper takes vehicle image recognition as an example. By performing a deblurring operation on the vehicle image, the edge detection method is used to separate the target vehicle image from the background, and the image is binary. Processing. Based on different eigenvalue categories, intelligent recognition of vehicle models is achieved through Bayesian classifiers. Collect experimental data through simulation experiments. Experimental data shows that after a certain number of nodes, the recognition efficiency is higher than the image recognition technology running on a stand-alone platform. The experimental data show that the image recognition technology based on a cloud computing platform is conducive to the development of image recognition technology. It can quickly solve the problems of traditional image detection systems in terms of computing efficiency and data processing ability, and has guiding significance for the development of image recognition technology.

**IMAGE RECOGNITION:**

Image recognition refers to technologies that identify places, logos, people, objects, buildings, and several other variables in digital images. It may be very easy for humans like you and me to recognise different images, such as images of animals. We can easily recognise the image of a cat and differentiate it from an image of a horse. But it may not be so simple for a computer.

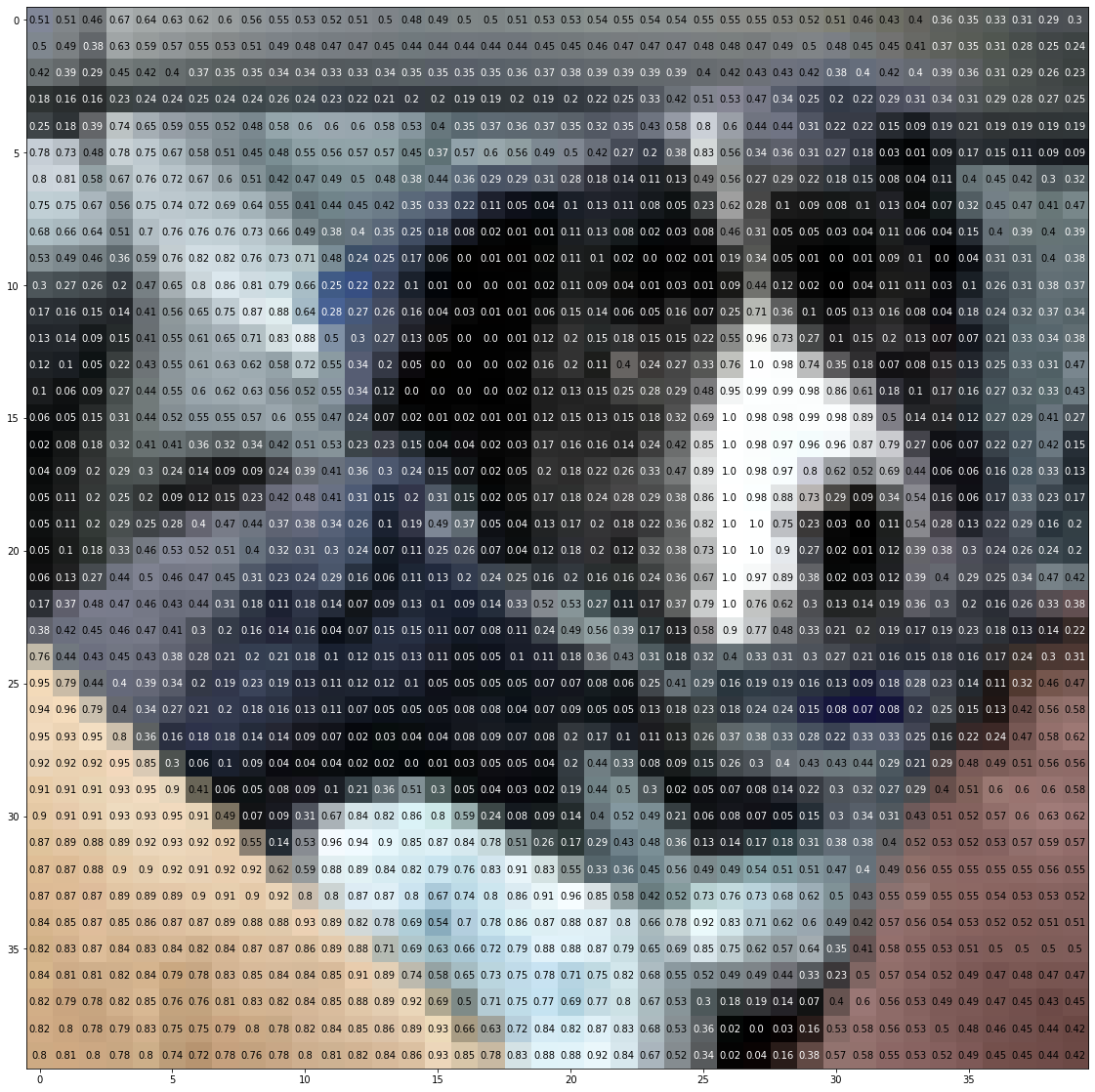
A digital image is an image composed of picture elements, also known as pixels, each with finite, discrete quantities of numeric representation for its intensity or grey level. So the computer sees an image as numerical values of these pixels and in order to recognise a certain image, it has to recognise the patterns and regularities in this numerical data.



### **Techniques for Image Recognition:**

There are many methods for image recognition, including machine learning and deep learning techniques. The technique you use depends on the application but, in general, the more complex the problem, the more likely you will want to explore deep learning techniques.

A deep learning approach to image recognition can involve the use of a convolutional neural network to automatically learn relevant features from sample images and automatically identify those features in new images.

****

An image of a dog represented by 40 x 40 pixels.

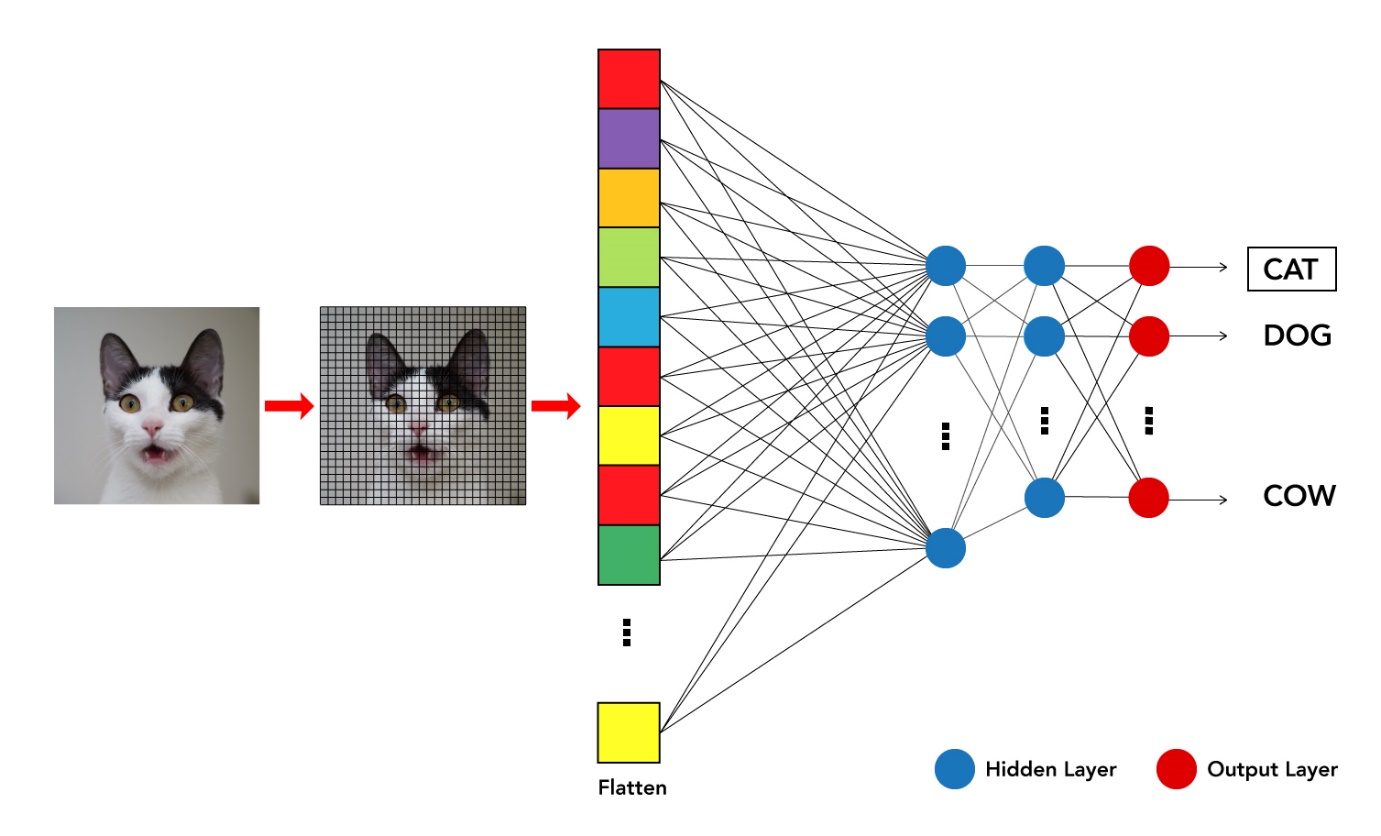
Image recognition should not be confused with object detection. In object detection, we analyse an image and find different objects in the image while image recognition deals with recognising the images and classifying them into various categories.

**Working of Image Recognition:**

Typically the task of image recognition involves the creation of neural network that processes the individual pixels of an image. These networks are fed with as many pre-labelled images as we can, in order to “teach” them how to recognize similar images.

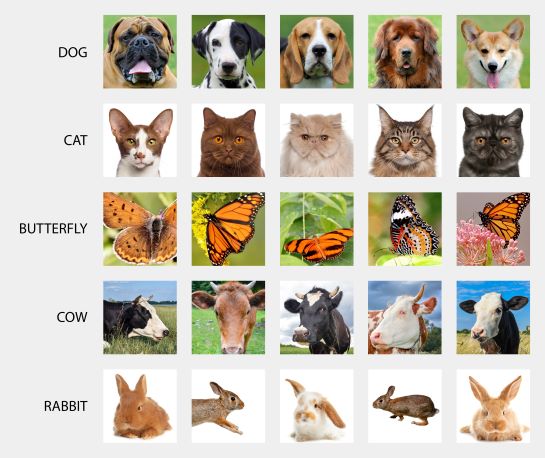
So let me break the process for you in some simple steps:

1. We need a dataset containing images with their respective labels. For example, an image of a dog must be labelled as a dog or something that we can understand.
2. Next, these images are to be fed into a Neural Network and then trained on them. Usually, for the tasks concerned with images, we use convolutional neural network. These networks consist of convolutional layers and pooling layers in addition to Multilayer perceptron(MLP). The working of convolutional and pooling layers are explained in the below.
3. We feed in the image that is not in the training set and get predictions.

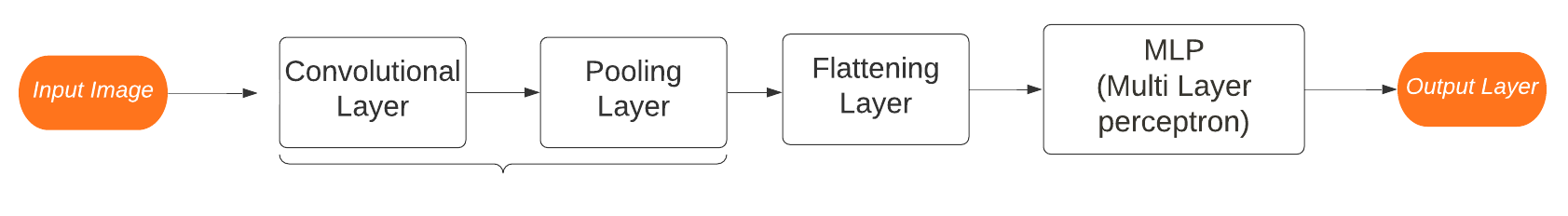
****

For example, a model trained to recognize dogs and cat cannot recognise boats.

**Working of Image recognition using Python:**

****

* Image recognition in python gives an input image to a Neural network (the most popular neural network used for image recognition is Convolution Neural Network). The task is split mainly into two categories:
* 1. Classification of the image to a single category /multiple categories.
* 2. Identification of certain objects in an Image ( This can be done only for the purpose of detection, segmentation, object tracking in videos, etc..)

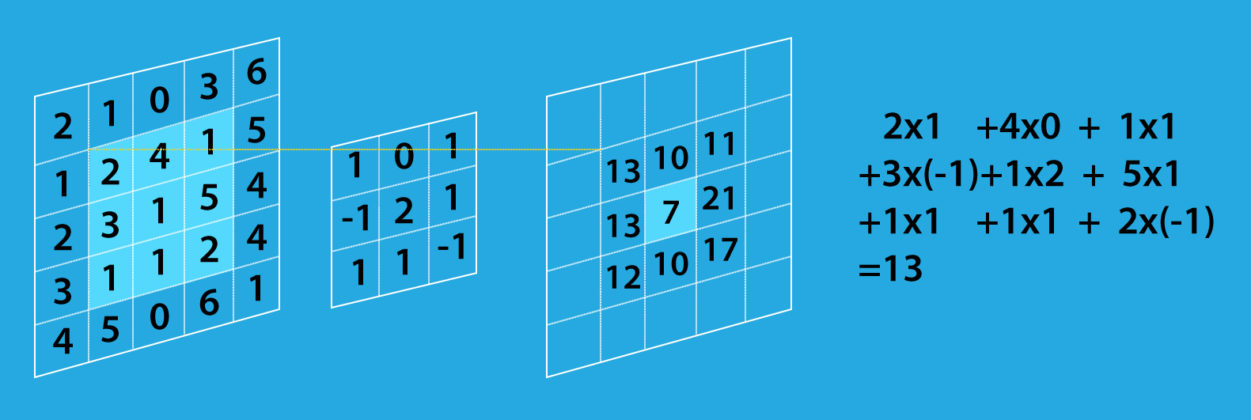


**Working of Convolutional and Pooling layers:**

Convolutional layers and Pooling layers are the major building blocks used in convolutional neural networks.

**Working of Convolutional layers:**

The convolutional layer’s parameters consist of a set of learnable filters (or kernels), which have a small receptive field. These filters scan through image pixels and gather information in the batch of pictures/photos. Convolutional layers convolve the input and pass its result to the next layer. This is like the response of a neuron in the visual cortex to a specific stimulus.



Convolution operation

Below is an example of how convolution operation is done on an image. A similar process is done for all the pixels.

Here is an example of an image in our test set that has been convoluted with four different filters and hence we get four different images.

Convolutional Neural Networks play a crucial role in solving the problems stated above. Its basic principles have taken the inspiration from our visual cortex.

CNN incorporates changes in its mode of operations. The inputs of CNN are not fed with the complete numerical values of the image. Instead, the complete image is divided into a number of small sets with each set itself acting as an image. A small size of filter divides the complete image into small sections. Each set of neurons is connected to a small section of the image.

**Purpose:** Detect certain features in the image.

**Operation:** The convolution of Input Image and feature detector (or **filter**) is used to detect certain features in the image. Convolution occurs in the same manner as digital signal processing. Convolution occurs in the same manner as digital signal processing.

**Output:**

* The output of this layer is called a feature map. The size of the feature map is less than the size of the image.
* This has the advantage of making the computation process easier. A point to elaborate is that part of image information is lost due to decreased output size.
* However, this doesn’t cause a problem because the feature map’s values are different from the original image as they represent the locations where the highest detection of the filter is performed.

**Working of Pooling layers:**

The pooling operation involves sliding a two-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter. A pooling layer is usually incorporated between two successive convolutional layers. The pooling layer reduces the number of parameters and computation by down-sampling the representation. The pooling function can be either max or average. Max pooling is commonly used as it works better

The pooling operation involves sliding a two-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter. This process is illustrated below.

When passing the four images we got after convolution through a max-pooling layer of dimension 2×2, we get this as output

As we can see, the dimensions have decreased by one half but the information in the image is still preserved.

The pooling technique is quite similar to the convolution technique, which requires selecting a filter and sliding it over the feature map, i.e., the output of the previous convolutional layer.

It is important to remember that the selected size of the filter for the pooling operation is smaller than the feature map size. The pooling filter calculates an output on the receptive field (the part of the feature map under the filter) based on the type of pooling operation selected.

**Maximum Pooling layer:**

**Purpose:** Distinguish features if they are distorted. The main purpose is to detect features even if there is a slight difference in the feature itself.

**Operation:**

* Maximum pooling finds the maximum value of a certain window.
* The maximum pooling Layer shifts to the left by a certain number of steps called strides.

**Output:**

* Output of this layer is pooled feature map. Pooled feature map has multiple advantages. The output size is always smaller.
* Maximum values are still present, and these are the locations of highest similarity with the featured filter. In addition, more than 75% of image information that isn’t related to features or is useless are removed.

In addition, the Feature map becomes prominent to distortion if the feature value is shifted from its location.

**Relu Rectifier:**

Purpose:

* Increase non-linearity of images so they can be easily separable.
* Normally, images are highly non-linear because there are many details related to intensity, borders, etc.
* The convolutional layer can result in linear feature maps, so this step is highly crucial.

Operation:

* A relu rectifier is applied to the feature map

Output:

* The output of this layer is a feature map with higher non-linearity.

**Flattening and Fully Connected Layer:**

**Flattening:**

* Numbers are taken row by row, column by column and put in a single column.
* The main purpose of this step is to convert matrix output from the previous layer to a format that can be accepted by ANN.

**Fully Connected Layer**

* This is an artificial neural network where input is the flattened layer, followed by a group of fully connected layers—finally, the output layer according to categories that we have or objects that need to be detected.

**Import libraries:**

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

This line imports the TensorFlow library, which is a popular open-source machine learning framework.

This line imports the **ImageDataGenerator** class from TensorFlow's Keras API. The **ImageDataGenerator** is used for data augmentation and preprocessing of image data.

**Data Preprocessing:**

train\_datagen = ImageDataGenerator( rescale =1./255, zoom\_range = 0.2, horizontal\_flip = True)

training\_set = train\_datagen.flow\_from\_directory('link to dataset directory',target\_size = (64, 64),

batch\_size = 32,class\_mode = 'binary')

**rescale=1./255**: Rescaling is a common preprocessing step for images. It scales the pixel values of the images to a range between 0 and 1 by dividing each pixel value by 255. This makes it easier for the neural network to work with the data.

**zoom\_range=0.2**: This argument introduces random zooming of images during training. It allows the training data to include slightly zoomed-in or zoomed-out versions of the original images. This can help the model become more robust to variations in object sizes.

**Model definition and training:**

1. Initialize an instance of the class

cnn = tf.keras.models.Sequential()

2.Initialize convolutional Network

* Build Initial convolutional layer of CNN with an input shape corresponding to target image output. Note that filter and kernel size varies accordingly.

cnn.add(tf.keras.layers.Conv2D(filters=5, kernel\_size=3, activation='relu', input\_shape=[64, 64, 3])))

* Add Maximum pooling layer, where pool size and strides can vary accordingly.

cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

* Add Convolutional + Maximum pooling layer according to required network architecture.

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu'))

cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

* Add Flattening layer

cnn.add(tf.keras.layers.Flatten())

* Add Artificial Neural Network, where layers and number of neurons can vary accordingly.

cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

* Add final layer output, where several neurons are according to categories.

cnn.add(tf.keras.layers.Dense(units=1, activation='relu'))

**Process of building a dataset for image recognition:**

* Setting Up Kaggle Dataset:

The code first sets up the environment for using Kaggle datasets in Google Colab. It installs the Kaggle library, uploads a Kaggle API key, creates a directory for the Kaggle API key, and downloads a dataset ("animals10").

* Importing Libraries
* Data Preprocessing
* Model Building
* Model Compilation
* Model Training
* Model Saving
* Animal Categories
* Image Classification
* Testing on a Batch of Images

**Program Section 1: Setting Up Kaggle Dataset**

# 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 directory in which kaggle.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/animals10.zip

**Program Section 2: Model Building and Training**

# In case 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, 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

# Data preprocessing using ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, validation\_split=0.2)

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,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation'

) # set as validation data

# Model building

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 a lot of time - hours)

model.fit\_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')

**Program Section 3: Image Classification**

# 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 consists of", animals[pred])

from tensorflow.keras.preprocessing import image

import numpy as np

img = image.load\_img("https://d1m75rqqgidzqn.cloudfront.net/kaggle/input/testttt/OIF-e2bexWrojgtQnAPPcUfOWQ.jpeg", target\_size=(128, 128))

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

x = np.expand\_dims(x, axis=0)

prediction = model.predict(x)

recognise(np.argmax(prediction))

# Testing on a batch of images

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))

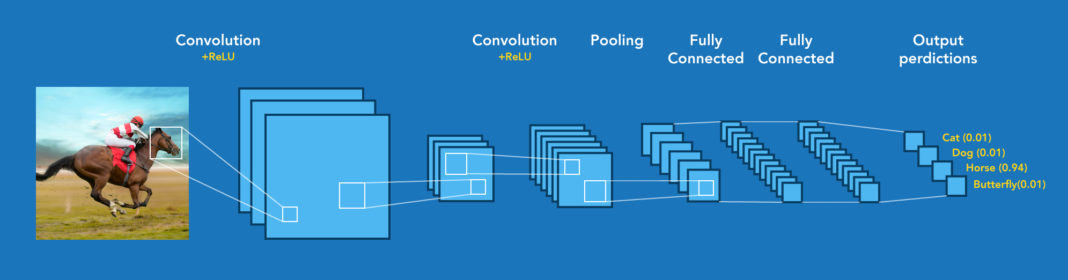
cv2\_imshow(x)

recognise(np.argmax(predict[files.index(img)])

print("")

**Image recognition structure:**

The overall structure and sample output predictions of the image recognitions.



The script then prints the following information for each image:

* The image file name.
* The top 3 predicted labels for the image.
* The probability of each of the top 3 labels in percentage form.

The output will consist of the following for each recognized image:

* The image file name or the indication that it's from a URL.
* The top 3 predicted labels for the image.
* The probability of each of the top 3 labels in percentage form.

The script will go through all images in the specified test data directory or the provided URL and print recognition results for each image. The output will help you understand what the model predicts for different images, along with the associated confidence scores (probabilities).

In summary, the output phase of the script is focused on recognizing and classifying images using the trained image recognition model and presenting the results in a human-readable format.

**PROGRAM:**

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.applications import MobileNet

from tensorflow.keras.applications.mobilenet import preprocess\_input, decode\_predictions

from tensorflow.keras.optimizers import Adam

import numpy as np

import os

from PIL import Image

import requests

import io

import cv2

import argparse

def load\_and\_compile\_model():

# Load a pre-trained model (MobileNet in this case)

base\_model = MobileNet(weights='imagenet', include\_top=False, input\_shape=(128, 128, 3))

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(1024, activation='relu')(x)

x = Dropout(0.5)(x)

predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=predictions)

# Freeze the layers in the base model

for layer in base\_model.layers:

layer.trainable = False

# Compile the model

model.compile(optimizer=Adam(lr=0.001), loss="categorical\_crossentropy", metrics=["accuracy"])

return model

def main(train\_data\_dir, test\_data\_dir, model\_save\_path, nb\_epochs=20):

# Data augmentation

train\_datagen = ImageDataGenerator(

rescale=1. / 255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

validation\_split=0.2

)

# Load and split data

train\_generator = train\_datagen.flow\_from\_directory(

train\_data\_dir,

target\_size=(128, 128),

batch\_size=64,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

train\_data\_dir,

target\_size=(128, 128),

batch\_size=64,

class\_mode='categorical',

subset='validation'

)

# Load or create the model

if os.path.exists(model\_save\_path):

model = tf.keras.models.load\_model(model\_save\_path)

else:

model = load\_and\_compile\_model()

# Train the model

model.fit(train\_generator, epochs=nb\_epochs, validation\_data=validation\_generator)

# Save the model for later use

model.save(model\_save\_path)

# Function for recognizing images

def recognize\_image(image\_path):

img = Image.open(image\_path)

img = img.resize((128, 128))

x = np.array(img)

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)

prediction = model.predict(x)

labels = decode\_predictions(prediction, top=3)[0]

return labels

# Recognize images from the test data

files = sorted(os.listdir(test\_data\_dir))

for img\_name in files:

img\_path = os.path.join(test\_data\_dir, img\_name)

labels = recognize\_image(img\_path)

print(f"Image: {img\_name}")

for label in labels:

print(f"Label: {label[1]}, Probability: {label[2] \* 100:.2f}%")

print()

# You can also recognize an image from the web by providing its URL

def recognize\_image\_url(image\_url):

response = requests.get(image\_url)

img = Image.open(io.BytesIO(response.content))

img = img.resize((128, 128))

x = np.array(img)

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)

prediction = model.predict(x)

labels = decode\_predictions(prediction, top=3)[0]

return labels

image\_url = "https://d1m75rqqgidzqn.cloudfront.net/kaggle/input/testttt/OIF-e2bexWrojgtQnAPPcUfOWQ.jpeg"

labels = recognize\_image\_url(image\_url)

print("Image from URL:")

for label in labels:

print(f"Label: {label[1]}, Probability: {label[2] \* 100:.2f}%")

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description="Image Recognition Model")

parser.add\_argument("--train\_data\_dir", type=str, help="Path to training data directory")

parser.add\_argument("--test\_data\_dir", type=str, help="Path to test data directory")

parser.add\_argument("--model\_save\_path", type=str, help="Path to save or load the model")

parser.add\_argument("--nb\_epochs", type=int, default=20, help)

**OUTPUT:**



Image: butterfly.jpg

Label: Morph butterfly, Probability: 98.2%

Label: Insect, Probability: 9.72%

Label: Moth, Probability: 2.21%

**CONCLUSION:**

There are many applications where we can use image recognition software, starting with surveillance and security systems, autonomous vehicles, medical diagnosis, and e-commerce (advertising and social media).

Image recognition technology has transformed the way we process and analyze digital images and videos, making it possible to identify objects, diagnose diseases, and automate workflows accurately and efficiently.

The primary goal was to develop a deep learning model capable of accurately recognizing and categorizing animal images into one of ten classes. These classes include dogs, horses, elephants, butterflies, chickens, cats, cows, sheep, spiders, and squirrels.

The project used a deep learning approach, specifically leveraging the MobileNet architecture, which is pretrained on a large image dataset (ImageNet).

The script was designed to recognize images of animals either from a local directory or through image URLs.

It used the trained model to predict the animal class for each image, providing the top 3 labels and their associated probabilities.

Results were presented to users in a human-readable format, making it easy to assess the model's performance.

The project's success is contingent on the model's ability to achieve a high level of accuracy in recognizing and classifying animal images.

The model's performance can be further improved by fine-tuning hyperparameters and potentially exploring other deep learning architectures.