

Project Report
on
IMAGE RECOGNITION

Submitted by

G Manisha
R170474

Under the guidance of

M Muni Babu
M.Tech, (Ph.D), Assistant Professor

Department of Computer Science and Engineering



Rajiv Gandhi University of Knowledge and Technologies(RGUKT),

R.K.Valley, Y.S.R Kadapa, Andra Pradesh.



Rajiv Gandhi University of Knowledge Technologies

RK Valley, Kadapa (Dist), Andhra Pradesh, 516330

CERTIFICATE

This is to certify that the project work titled “**Image Recognition**” is a bonafied project work submitted by *G Manisha* in the department of COMPUTER SCIENCE AND ENGINEERING in partial fulfillment of requirements for the award of degree of Bachelor of Technology in Computer science and engineering for the year 2021-2022 carried out the work under the supervision

GUIDE
M MUNIBABU

HEAD OF THE DEPARTMENT
P HARINADA

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to our respected Director, Prof. K. SANDHYA RANI for fostering an excellent academic climate in our institution.

I also express my sincere gratitude to our respected Head of the Department P HARINADA for his encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project.

I would like to convey thanks to our guide at college Mr. M. MUNIBABU for his guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

My sincere thanks to all the members who helped me directly and indirectly in the completion of work I express my profound gratitude to all our friends and family members for their encouragement.

INDEX

S.NO.	INDEX	PAGE NO.
1	Abstract	5
2	Introduction	6
3	purpose	6
4	Scope	6
5	Requirement Specification	7
6	Analysis and design	7-20
6.1	Usecase	20-22
6.2	ER Diagram	22-24
7	Implementation and system testing	25
8	Project Output	26-32
9	Conclusion	32
10	References	32

Abstract

The IMAGE RECOGNITION is internet based Application of deep learning. The main motto of this application is to classify the given images into their respective classes/labels. Image recognition is a vast and vibrant side of computer vision. If there is a single object to be detected in an image, it is known as image Localization.

Image recognition is important side of image processing for deep learning without involving any human support at any step. In this project hundreds of images of every, dogs, cats and many others are taken then distributed them into category of test dataset and training dataset for our learning model. The results are obtained using custom neural network with the architecture of convolutional neural networks and Google colab.

INTRODUCTION

IMAGE RECOGNITION , this model asks the user to upload the image and then to preprocess it. User has to upload an image then run the other cells in google colab then the user will be given output with three fields containing 'serial number in the dataset, Image name and Confidence level' .This model identifies nearly and atleast thousand images including images like butterfly, dog, cat, snake, tissue paper, zebra and so on.

Purpose

The main purpose of IMAGE RECOGNITION is to make the model to recognize and classify the given image into it's respective label among the hundreds of labels.

Scope

This model can be used in various types of real time projects like

- Image classification
- Liscence plate recognition
- Face recognition
- Self driving cars
- Accident avoidance systems

Advantages:

- Image recognition can really **help you with digital marketing**. you will be able to offer visual insights to your customers without the expensive product creation that uses logo detection.
- Image recognition is used to perform many machine-based visual tasks, such as labeling the content of images with meta-tags
- Performing image content search and guiding autonomous robots, self-driving cars and accident-avoidance systems.

Disadvantages:

- It can be accessed only when the internet is available.
- Initial cost is high depending on the system used.
- These are not scalable to large dataset and also require large dataset and time to train a model.

Requirement Specification

There are no Extra Softwares Required for image Classification in CNN.

All you need is and Python compiling IDE.

Analysis and Implementation is done using the methods below:

All the terms used in this model are:

ANALYSIS AND DESIGN

Image:

A picture is spoken to as a two dimensional capacity $f(x, y)$ where x and y are spatial co-ordinates and the adequacy of "T" at any match of directions (x, y) is known as the power of the picture.

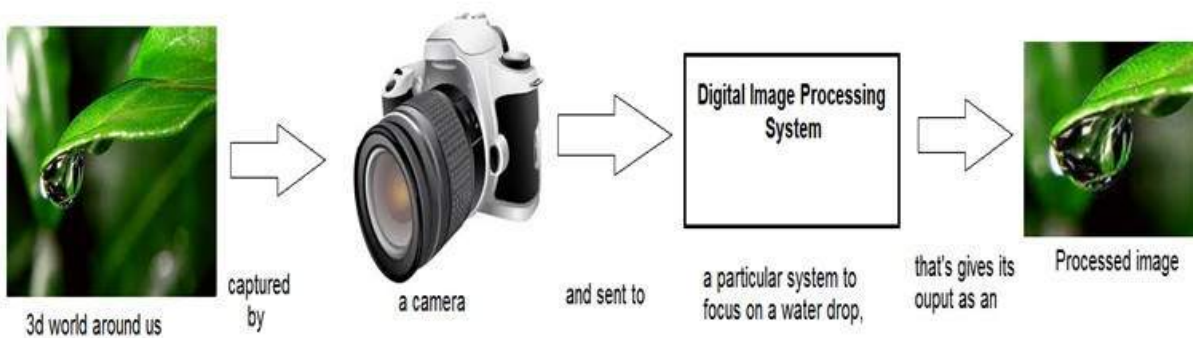


Fig.1 Digital image processing

Why Image Processing?

Since the digital image is invisible, it must be prepared for viewing on one or more output device(laser ,printer, monitor at).The digital image can be optimized for the application by enhancing the appearance of the structures within it.

There are three of image processing used. They are

- Image to Image transformation
- Image to Information transformations
- Information to Image transformations

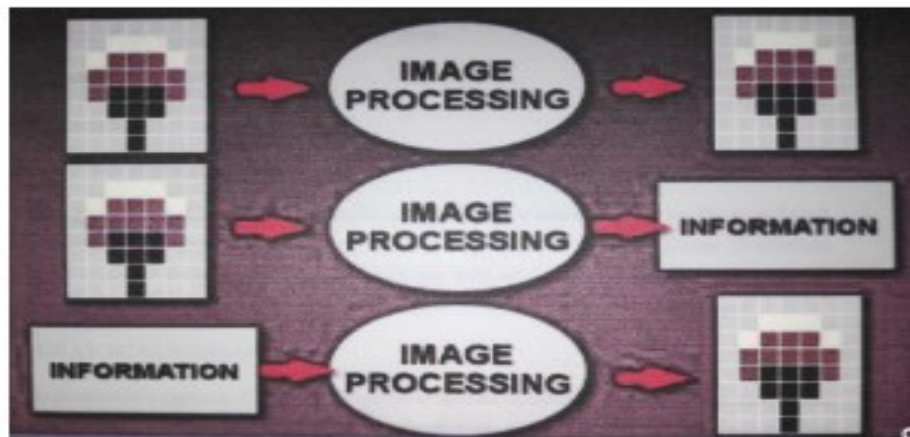


Fig.2 Types of image processing

Pixel:

Pixel is the smallest element of an image. Each pixel correspond to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255.Each pixel store a value proportional to the light intensity at that particular location. It is indicated in either Pixels per inch or Dots per inch.

Resolution:

The resolution can be defined in many ways. Such as pixel resolution, spatial resolution, temporal resolution, spectral resolution.In pixel resolution, the term resolution refers to the total number of count of pixels in an digital image. For example, If an image has M rows and N columns, then its resolution can be defined

as $M \times N$. Higher is the pixel resolution, the higher is the quality of the image.
Resolution of an image is of generally two types.
-LowResolutionimage
-HighResolutionimage

Since high resolution is not a cost effective process It is not always possible to achieve high resolution images with low cost. Hence it is desirable Imaging. In Super Resolution imaging, with the help of certain methods and algorithms we can be able to produce high resolution images from the low resolution image.

DEEP LEARNING:

INTRODUCTION:

Deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information. Observations can be in the form of images, text, or sound. The inspiration for deep learning is the way that the human brain filters information. Its purpose is to mimic how the human brain works to create some real magic. In the human brain, there are about 100 billion neurons. Each neuron connects to about 100,000 of its neighbors. We're kind of recreating that, but in a way and at a level that works for machines. In our brains, a neuron has a body, dendrites, and an axon. The signal from one neuron travels down the axon and transfers to the dendrites of the next neuron. That connection where the signal passes is called a synapse. Neurons by themselves are kind of useless. But when you have lots of them, they work together to create some serious magic. That's the idea behind a deep learning algorithm! You get input from observation and you put your input into one layer. That layer creates an output which in turn becomes the input for the next layer, and so on. This happens over and over until your final output signal! The neuron (node) gets a signal or signals (input values), which pass through the neuron. That neuron delivers the output signal. Think of the input layer as your senses: the things you see, smell, and feel, for example. These are independent variables for one single observation. This information is broken down into numbers and the bits of binary data that a computer can use. You'll need to either standardize or normalize these variables so that they're within the same range. They use many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output of the

previous layer for its input. What they learn forms a hierarchy of concepts. In this hierarchy, each level learns to transform its input data into a more and more abstract and composite representation. That means that for an image, for example, the input might be a matrix of pixels. The first layer might encode the edges and compose the pixels. The next layer might compose an arrangement of edges. The next layer might encode a nose and eyes. The next layer might recognize that the image contains a face, and so on.

What happens inside the neuron?

The input node takes in information in a numerical form. The information is presented as an activation value where each node is given a number. The higher the number, the greater the activation. Based on the connection strength (weights) and transfer function, the activation value passes to the next node. Each of the nodes sums the activation values that it receives (it calculates the weighted sum) and modifies that sum based on its transfer function. Next, it applies an activation function. An activation function is a function that's applied to this particular neuron. From that, the neuron understands if it needs to pass along a signal or not. Each of the synapses gets assigned weights, which are crucial to Artificial Neural Networks (ANNs). Weights are how ANNs learn. By adjusting the weights, the ANN decides to what extent signals get passed along. When you're training your network, you're deciding how the weights are adjusted. The activation runs through the network until it reaches the output nodes. The output nodes then give us the information in a way that we can understand. Your network will use a cost function to compare the output and the actual expected output. The model performance is evaluated by the cost function. It's expressed as the difference between the actual value and the predicted value. There are many different cost functions you can use, you're looking at what the error you have in your network is. You're working to minimize loss function. (In essence, the lower the loss function, the closer it is to your desired output). The information goes back, and the neural network begins to learn with the goal of minimizing the cost function by tweaking the weights. This process is called backpropagation. In forward propagation, information is entered into the input layer and propagates forward through the network to get our output values. We compare the values to our expected results. Next, we calculate the errors and propagate the info backward. This allows us to train the network and update the weights. (Backpropagation allows us to adjust all the weights simultaneously.) During this process, because of the way the algorithm is structured, you're able to adjust all of the weights simultaneously. This allows you to see which part of the error each of your weights in the neural network is responsible for. When you've adjusted the weights to the optimal level, you're ready to proceed to the testing phase. You can create the architecture and then let it

go and learn. Once it's trained up, you can give it a new image and it will be able to distinguish output. Feedforward and feedback networks

A feedforward network is a network that contains inputs, outputs, and hidden layers. The signals can only travel in one direction (forward). Input data passes into a layer where calculations are performed. Each processing element computes based upon the weighted sum of its inputs. The new values become the new input values that feed the next layer (feed-forward). This continues through all the layers and determines the output. Feedforward networks are often used in, for example, data mining.

A feedback network (for example, a recurrent neural network) has feedback paths. This means that they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feedback networks are often used in optimization problems where the network looks for the best arrangement of interconnected factors. Weighted Sum Inputs to a neuron can either be features from a training set or outputs from the neurons of a previous layer. Each connection between two neurons has a unique synapse with a unique weight attached. If you want to get from one neuron to the next, you have to travel along the synapse and pay the "toll" (weight). The neuron then applies an activation function to the sum of the weighted inputs from each incoming synapse. It passes the result on to all the neurons in the next layer. When we talk about updating weights in a network, we're talking about adjusting the weights on these synapses. A neuron's input is the sum of weighted outputs from all the neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights one for each synapse. In a nutshell, the activation function of a node defines the output of that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At it's simplest, the function is binary: yes (the neuron fires) or no (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range. If you were using a function that maps a range between 0 and 1 to determine the likelihood that an image is a cat, for example, an output of 0.9 would show a 90% probability that your image is, in fact, a cat. Activation function In a nutshell, the activation function of a node defines the output of that node. The activation function (or transfer function)

translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At it's simplest, the function is binary: yes (the neuron fires) or no (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range. There are many activation functions, but these are the four very common ones

1.Thresholdfunction

This is a step function. If the summed value of the input reaches a certain threshold the function passes on 0. If it's equal to or more than zero, then it would pass on 1. It's a very rigid, straightforward, yes or no function.

2.Sigmoid Function

This function is used in logistic regression. Unlike the threshold function, it's a smooth, gradual progression from 0 to 1. It's useful in the output layer and is used heavily for linear regression.

3.Hyperbolic Tangent Function

This function is very similar to the sigmoid function. But unlike the sigmoid function which goes from 0 to 1, the value goes below zero, from -1 to 1. Even though this isn't a lot like what happens in a brain, this function gives better results when it comes to training neural networks. Neural networks sometimes get "stuck" during training with the sigmoid function. This happens when there's a lot of strongly negative input that keeps the output near zero, which messes with the learning process.

4.Rectifier function

This might be the most popular activation function in the universe of neural networks. It's the most efficient and biologically plausible. Even though it has a kink, it's smooth and gradual after the kink at 0. This means, for example, that your output would be either "no" or a percentage of "yes." This function doesn't require normalization or other complicated calculations.

CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks are very similar to ordinary Neural Networks they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimize through learning. Each neuron will still receive an input and perform an operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply. The only notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model. One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. Common machine learning benchmarking datasets such as the MNIST database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just 28×28 . With this dataset a single neuron in the first hidden layer will contain 784 weights ($28 \times 28 \times 1$ where 1 bare in mind that MNIST is normalised to just black and white values), which is manageable for most forms of ANN. If you consider a more substantial coloured image input of 64×64 , the number of weights on just a single neuron of the first layer increases substantially to 12, 288. Also take into account that to deal with this scale of input, the network will also need to be a lot larger than one used to classify colour-normalised MNIST digits.

CNN ARCHITECTURE:

CNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells (Hubel & Wiesel, 1959, 1962), motivates their architecture. CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top

of each other to form a deep model. It illustrates typical CNN architecture for a toy image classification task. An image is input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations from these operations feed one or more fully connected layers. Finally, the last fully connected layer outputs the class label. Despite this being the most popular base architecture found in the literature, several architecture changes have been proposed in recent years with the objective of improving image classification accuracy or reducing computation costs. Although for the remainder of this section, we merely fleetingly introduce standard CNN architecture.

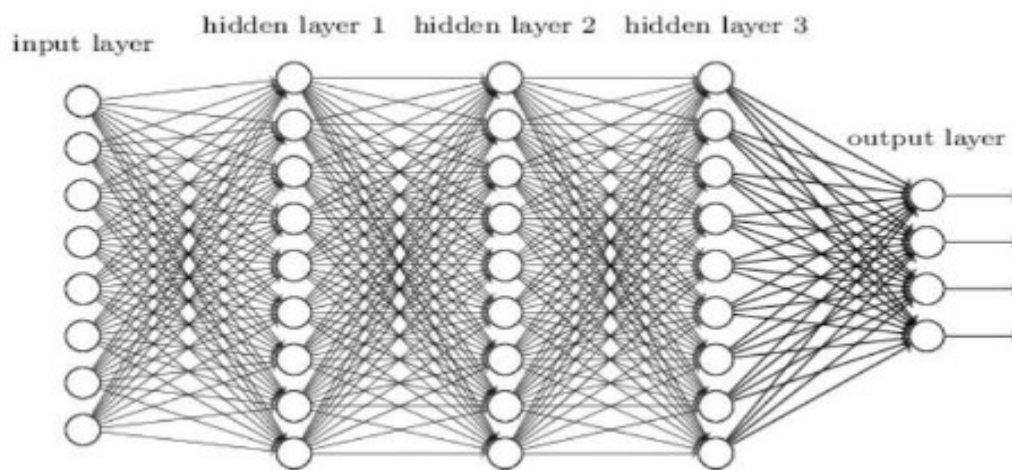


Fig.3 CNN Architecture

CONVOLUTIONAL LAYERS:

The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent through a nonlinear activation function. All neurons within a feature map have weights that are constrained to be equal; however, different feature maps within the same convolutional layer have different weights so that several features

can be extracted at each location. As the name implies, the convolutional layer plays a vital role in how CNNs operate. The layer's parameters focus around the use of learnable kernels. These kernels are usually small in spatial dimensionality, but spread along the entirety of the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. These activation maps can be visualised. As we glide through the input, the scalar product is calculated for each value in that kernel. From this the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as activations.

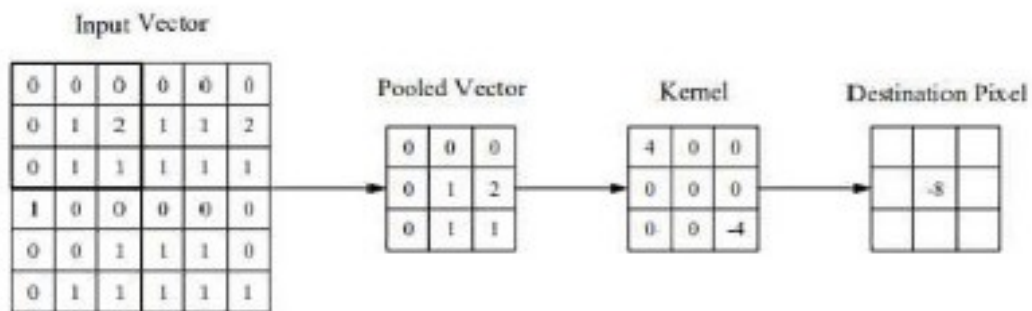


Fig.4 Visual representation of a convolutional layer

The centre element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels. Every kernel will have a corresponding activation map, of which will be stacked along the depth dimension to form the full output volume from the convolutional layer. As we alluded to earlier, training ANNs on inputs such as images results in models of which are too big to train effectively. This comes down to the fully connected manner of standard ANN neurons, so to mitigate against this every neuron in a convolutional layer is only connected to small region of the input volume. The dimensionality of this region is commonly referred to as the receptive field size of the neuron. The magnitude of the connectivity through the depth is nearly always equal to the depth of the input. For example, if the input to the network is an image of size $64 \times 64 \times 3$ (a RGB coloured image with a dimensionality of 64×64) and we set the receptive field size as 6×6 , we would have a total of 108 weights on each neuron within the convolutional layer. ($6 \times 6 \times 3$ where 3 is the magnitude of connectivity across the depth of the volume) To put this into perspective, a standard neuron seen in other forms of ANN would contain 12,288 weights each. Convolutional layers are also able to significantly reduce the complexity of the model through the optimisation of its output. These are optimised through three hyperparameters, the depth, the stride and setting zero-padding.

The depth of the output volume produced by the convolutional layers can be manually set through the number of neurons within the layer to a the same region of the input. This can be seen with other forms of ANNs, where the all of the neurons in the hidden layer are directly connected to every single neuron beforehand. Reducing this hyperparameter can significantly minimise the total number of neurons of the network, but it can also significantly reduce the pattern recognition capabilities of the model.

We are also able to define the stride in which we set the depth around the spatial dimensionality of the input in order to place the receptive field. For example if we were to set a stride as 1, then we would have a heavily overlapped receptive field producing extremely large activations. Alternatively, setting the stride to a greater number will reduce the amount of overlapping and produce an output of lower spatial dimensions.

Zero-padding is the simple process of padding the border of the input, and is an effective method to give further control as to the dimensionality of the output volumes. It is important to understand that through using these techniques, we will alter the spatial dimensionality of the convolutional layers output. Despite our best efforts so far we will still find that our models are still enormous if we use an image input of any real dimensionality. However, methods have been developed as to greatly curtail the overall number of parameters within the convolutional layer. Parameter sharing works on the assumption that if one region feature is useful to compute at a set spatial region, then it is likely to be useful in another region. If we constrain each individual activation map within the output volume to the same weights and bias, then we will see a massive reduction in the number of parameters being produced by the convolutional layer.

As a result of this as the backpropagation stage occurs, each neuron in the output will represent the overall gradient of which can be totalled across the depth - thus only updating a single set of weights, as opposed to every single one.

Training:

CNNs and ANN in general use learning algorithms to adjust their free parameters in order to attain the desired network output. The most common algorithm used for this purpose is backpropagation. Backpropagation computes the gradient of an objective function to determine how to adjust a network's parameters in order to minimize errors that affect performance. A commonly experienced problem with training CNNs, and in particular DCNNs, is overfitting, which is poor performance on a held-out test set after the network is trained on a small or even large training set. This affects the model's ability to generalize on unseen data and is a major challenge for DCNNs that can be assuaged by regularization.

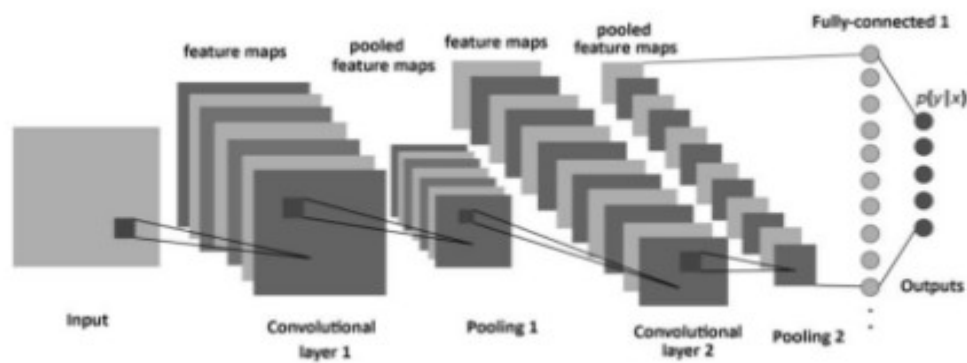


Fig.5 Training in CNN

SOME IMPORTANT MODULES USED:

PyTorch is a library for Python programs that make it easy to create deep learning models. Like Python does for programming, PyTorch provides a great introduction to deep learning.

The PyTorch framework supports over 200 different mathematical operations. The popularity of PyTorch continues to rise as it simplifies the creation of artificial neural network (ANN) models.

The **torchvision** package consists of popular datasets, model architectures, and common image transformations for computer vision.

The **torchvision.models** subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection, video classification, and optical flow.

TorchVision offers pre-trained weights for every provided architecture, using the PyTorch torch.hub. Instantiating a pre-trained model will download its weights to a cache directory. This directory can be set using the TORCH_HOME environment variable.

Before using the pre-trained models, one must preprocess the image (resize with right resolution/interpolation, apply inference transforms, rescale the values etc). There is no standard way to do this as it depends on how a given model was trained. It can vary across model families, variants or even weight versions. Using the correct preprocessing method is critical and failing to do so may lead to decreased accuracy or incorrect outputs.

The **AlexNet** model was originally introduced in the ImageNet Classification with Deep Convolutional Neural Networks(CNN) paper. The implemented architecture is slightly different from the original one, and is based on One weird trick for parallelizing convolutional neural networks.

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.

Python Imaging Library which provides the python interpreter with image editing capabilities.

Compose(transforms)

Composes several transforms together.

Resize(size[, interpolation, max_size, ...])

Resize the input image to the given size.

CenterCrop(size)

Crops the given image at the center.

ToTensor()

Convert a PIL Image or numpy.ndarray to tensor.

normalize(tensor, mean, std[, inplace])

Normalize a float tensor image with mean and standard deviation.

`torch.nn.functional.softmax`

`torch.nn.functional.softmax(input, dim=None, _stacklevel=3, dtype=None)[source]`

Applies a softmax function.

Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

It is applied to all slices along dim, and will re-scale them so that the elements lie in the range [0, 1] and sum to 1.

Parameters

input (Tensor) – input

dim (int) – A dimension along which softmax will be computed.

dtype (torch.dtype, optional) – the desired data type of returned tensor. If specified, the input tensor is casted to dtype before the operation is performed. This is useful for preventing data type overflows. Default: None.

PIL image:

Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux.

Tensor:

The size of each dimension in a Tensor we call its shape. For example, a Tensor to represent a black and white image would have the shape [width , height , colors]. For a 640 x 480 pixel black and white image, the shape would be [640,480, 1].

Design Introduction:

Design is the first step in the development phase for any techniques and principles for the purpose of defining a device, a process or system in sufficient detail to permit its physical realization. Once the software requirements have been analyzed and specified the software design involves three technical activities - design, coding, implementation and testing that are required to build and verify the software.

The design activities are of main importance in this phase, because in this activity

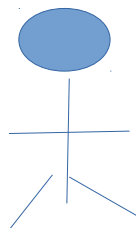
decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customer's requirements into finished software or a system.

Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

UML Diagrams:

Actor:

A coherent set of roles that users of use cases play when interacting with the use cases an observable result of value of an actor.



Use case:

A description of sequence of actions, including variants, that a system performs yields an observable result of value of an actor. actor diagram is drawn in a eclipse shape.



UML stands for Unified Modeling Language. UML is a language for specifying, visualizing and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to be built. The representation of the entities that are to be used in the product being developed need to be designed.

USECASE DIAGRAMS:

Use case diagrams model behavior within a system and helps the developers understand of what the user require. The stick man represents what's called an actor.

Use case diagram can be useful for getting an overall view of the system and clarifying that can do and more importantly what they can't do.

Use case diagram consists of use cases and actors and shows the interaction between the use case and actors.

- The purpose is to show the interactions between the use case and actor.
- To represent the system requirements from user's perspective.
- An actor could be the end-user of the system or an external system.

USECASE DIAGRAM: A Use case is a description of set of sequence of actions. Graphically it is rendered as an ellipse with solid line including only its name. Use case diagram is a behavioral diagram that shows a set of use cases and actors and their relationship. It is an association between the use cases and actors. An actor represents a real-world object. Primary Actor – Sender, Secondary Actor Receiver.

Use Case Diagram:

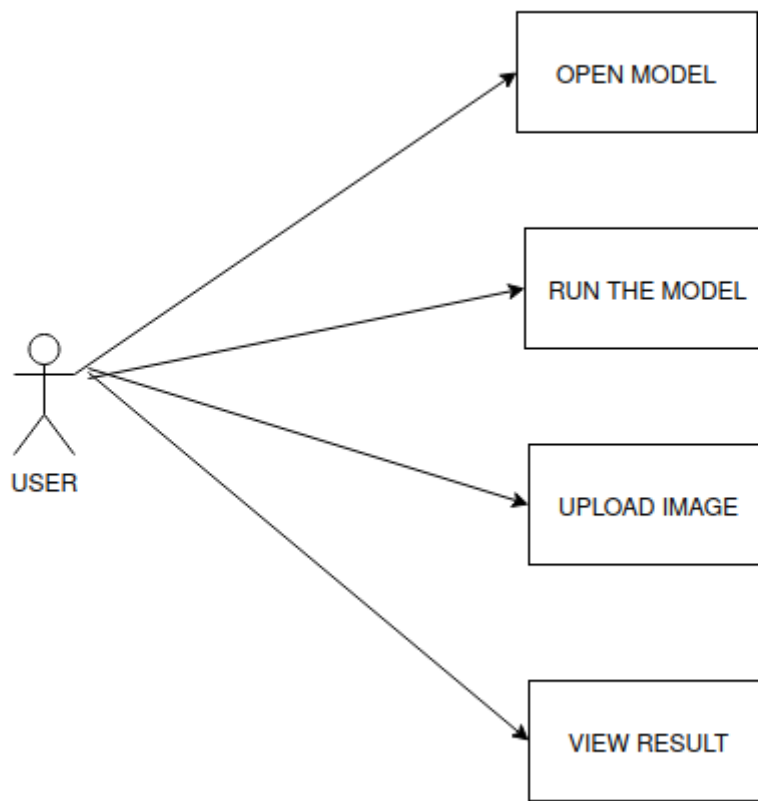


Fig.6 Usecase Uml diagram for Image recognition

ER Diagram:

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design for the database designer, the utility of the ER model is:

-It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.

-It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.

-In addition, the model can be used as a design plan by the database developer to implement a data model in specific database management software.

ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is widely used in academics texts and journals but rarely seen in either CASE tools or publications by non-academics. Today, there are a number of notations used; among the more common are Bachman, crow's foot, and IDEFIX.

All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

-**Entities** are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.

-**Relationships** are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs

-**Attributes**, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.

-**Cardinality** of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

ER DIAGRAM:

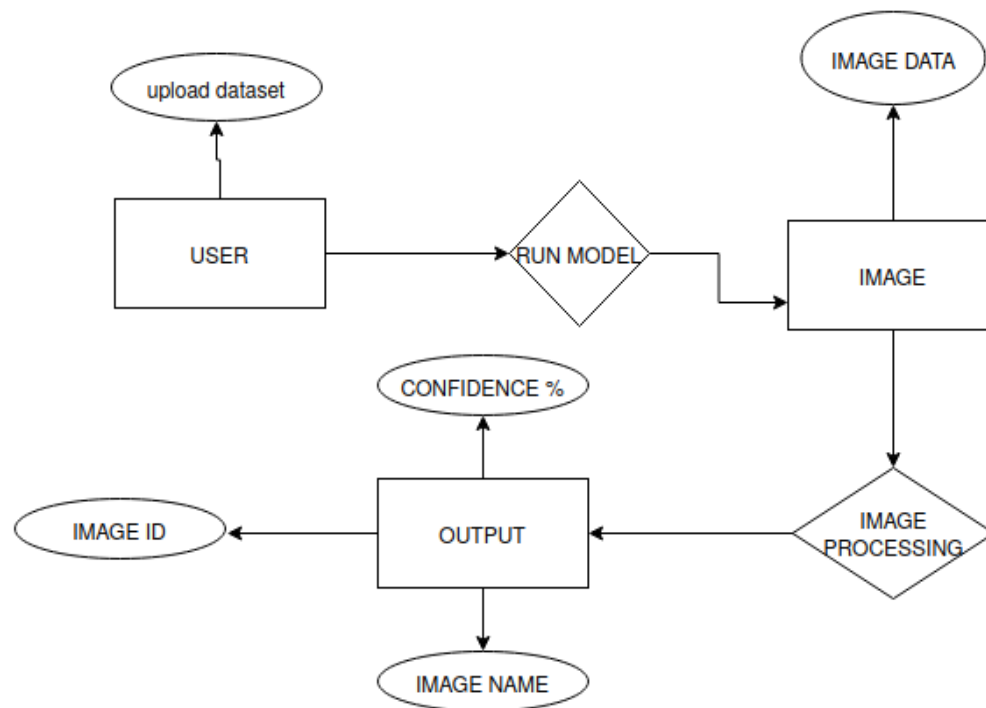


Fig.7 ER Diagram for Image Recognition

Implementation and System Testing

After all Requirements gathering, Analysis and Coding phases have been perfectly implemented Testing phase has to be implemented.

System Testing

The goal of the system testing process was to determine all faults in our project .The

program was subjected to a set of test inputs and many explanations were made and based on these explanations it will be decided whether the program behaves as expected or not. Our Project went through two levels of testing

1. Unit testing
- 2 .Integration testing

Unit Testing

Unit testing is commenced when a unit has been created and effectively reviewed .In order to test a single module we need to provide a complete environment i.e. besides the section we would require The procedures belonging to other units that the unit under test calls Non local data structures that module accesses .A procedure to call the functions of the unit under test with appropriate parameters

Integration Testing

In the Integration testing we test various combination of the project model by providing the input images.

The primary objective is to test the model interfaces in order to confirm that no errors are occurring when one cell invokes the other cell.

EVALUATION

Imagenet_classes.txt:

{0: 'tench, Tinca tinca',
1: 'goldfish, Carassius auratus',
2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
3: 'tiger shark, Galeocerdo cuvieri',
4: 'hammerhead, hammerhead shark',
5: 'electric ray, crampfish, numbfish, torpedo',
6: 'stingray',
7: 'cock',
8: 'hen',
9: 'ostrich, Struthio camelus',
10: 'brambling, Fringilla montifringilla',
11: 'goldfinch, Carduelis carduelis',
12: 'house finch, linnet, Carpodacus mexicanus',
13: 'junco, snowbird',

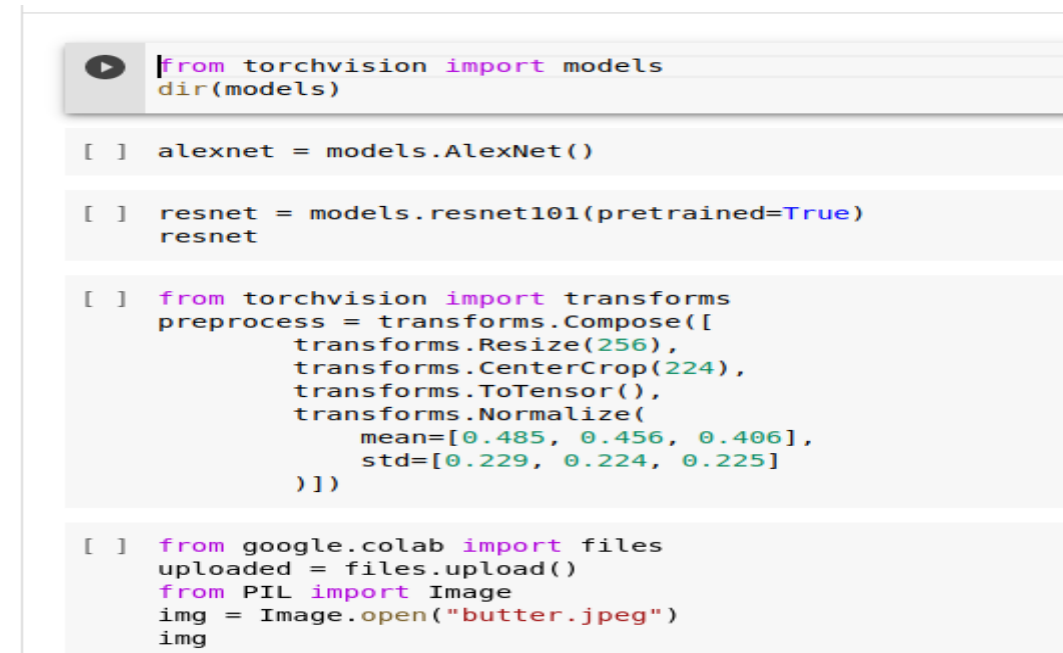
14: 'indigo bunting, indigo finch, indigo bird, Passerina cyanea',
15: 'robin, American robin, Turdus migratorius',
16: 'bulbul',
17: 'jay',
18: 'magpie',
19: 'chickadee',
20: 'water ouzel, dipper',
“
“
100: 'black swan, Cygnus atratus',
101: 'tusker',
102: 'echidna, spiny anteater, anteater',
103: 'platypus, duckbill, duckbilled platypus, duck-billed platypus, Ornithorhynchus anatinus',
104: 'wallaby, brush kangaroo',
105: 'koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus',
106: 'wombat',
107: 'jellyfish',
108: 'sea anemone, anemone',
109: 'brain coral',
110: 'flatworm, platyhelminth',
“
“
200: 'Tibetan terrier, chrysanthemum dog',
201: 'silky terrier, Sydney silky',

202: 'soft-coated wheaten terrier',
 203: 'West Highland white terrier',
 204: 'Lhasa, Lhasa apso',
 205: 'flat-coated retriever',
 206: 'curly-coated retriever',
 207: 'golden retriever',
 208: 'Labrador retriever',
 209: 'Chesapeake Bay retriever',
 210: 'German short-haired pointer',
 “
 “
 300: 'tiger beetle',
 301: 'ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle',
 302: 'ground beetle, carabid beetle',
 303: 'long-horned beetle, longicorn, longicorn beetle',
 304: 'leaf beetle, chrysomelid',
 305: 'dung beetle',
 306: 'rhinoceros beetle',
 307: 'weevil',
 308: 'fly',
 309: 'bee',
 310: 'ant, emmet, pismire',
 “
 400: 'academic gown, academic robe, judge's robe',
 401: 'accordion, piano accordion, squeeze box',
 402: 'acoustic guitar',
 403: 'aircraft carrier, carrier, flattop, attack aircraft carrier',
 404: 'airliner',
 405: 'airship, dirigible',
 406: 'altar',
 407: 'ambulance',
 408: 'amphibian, amphibious vehicle',
 409: 'analog clock',
 410: 'apiary, bee house',
 “
 “
 990: 'buckeye, horse chestnut, conker',
 991: 'coral fungus',
 992: 'agaric',
 993: 'gyromitra',
 994: 'stinkhorn, carrion fungus',
 995: 'earthstar',
 996: 'hen-of-the-woods, hen of the woods, Polyporus frondosus, Grifola frondosa',
 997: 'bolete',

998: 'ear, spike, capitulum',

999: 'toilet tissue, toilet paper, bathroom tissue']

Execution process and flow of model is depicted below:



```
[ ] from torchvision import models
    dir(models)

[ ] alexnet = models.AlexNet()

[ ] resnet = models.resnet101(pretrained=True)
    resnet

[ ] from torchvision import transforms
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

[ ] from google.colab import files
    uploaded = files.upload()
    from PIL import Image
    img = Image.open("butter.jpeg")
    img
```

Fig.8 Beginning of the model

```
[ ] from google.colab import files
    uploaded = files.upload()
    from PIL import Image
    img = Image.open("butter.jpeg")
    img
```

No files selected.

Upload widget is only available when the cell has been

Saving butter.jpeg to butter.jpeg



```
[ ] img_t = preprocess(img)
```

```
[ ] import torch
    batch_t = torch.unsqueeze(img_t, 0)
    resnet.eval()
    out = resnet(batch_t)
    out
```

Fig.9 Medieval of the model

```
import torch
batch_t = torch.unsqueeze(img_t, 0)
resnet.eval()
out = resnet(batch_t)
out
```

4.6955e-01	-2.5632e+00	6.5750e-01	-4.4255e-01	1.7201e-01
-1.5699e+00	-3.4670e-02	-3.2870e+00	3.1803e-01	-1.3205e-01
8.5488e-01	2.6875e+00	7.2645e-01	1.5204e+00	3.2000e+00
-2.8466e-01	2.2579e+00	-7.8125e-01	2.1672e+00	-2.2527e+00
1.2308e+00	4.8534e+00	-4.2532e+00	3.6055e+00	1.0552e+00
1.7676e-01	3.6569e-01	-9.3871e-01	-0.0524e+00	-3.3815e+00
1.2301e+00	4.2732e+00	2.7850e-01	-6.5395e+00	1.0524e-01
1.3515e-01	-2.6897e+00	2.6505e-01	-7.2422e-01	-2.7770e+00
2.3512e-01	3.1222e-01	3.2932e+00	-9.6434e-01	1.6924e+00
2.5932e+00	7.5384e-01	-3.2329e+00	-2.0855e+00	-6.9923e-01
1.8077e+00	-2.2669e+00	-6.2135e-01	-9.7730e+00	3.6240e+00
6.9821e+00	1.1175e+00	3.4405e+00	-6.4800e+00	3.7084e+00
1.9028e+00	5.8000e-01	2.88021e+00	-4.0203e+00	1.4000e+00
-2.8323e+00	1.6083e+00	2.2829e+00	7.3651e-01	-2.4434e+00
1.8828e+00	1.3181e-01	-3.0900e+00	3.0954e+00	2.1741e-01
-2.6609e+00	1.2013e+00	-8.2330e-01	-1.3127e+00	-1.7839e+00
-2.4098e+00	3.2300e-01	-2.8230e+00	2.2654e+00	2.7312e+00
-2.7655e+00	-1.2933e+00	-2.6837e+00	0.09227e+00	-2.3537e+00
-2.3271e+00	3.6348e+00	3.1353e+00	1.8227e+00	1.4526e+00
-8.4498e-01	1.6384e+00	3.3500e+00	8.0908e-01	2.6520e+00
3.8980e+00	1.2441e+00	3.4831e+00	2.2720e+00	1.5102e+00
2.7078e+00	6.7422e-01	3.2350e+00	3.4575e-01	3.8800e+00
-1.1317e+00	5.0648e+00	-3.7782e+00	-9.4844e-01	2.3831e-01
1.8678e+00	6.8977e+00	3.6382e+00	-9.9814e-01	-3.1898e+00
9.8432e-01	6.2449e-01	-3.6054e+00	2.2897e+00	1.4527e+00
3.8934e+00	9.8780e+00	7.3337e-01	-9.1973e-01	3.4608e+00
-1.1297e+00	5.8730e+00	5.1655e-02	-4.1668e+00	1.0649e+00
1.1319e+00	1.4682e+00	-9.9590e-01	-2.6554e+00	-3.3551e+00
-4.4853e+00	3.8383e+00	-5.8861e-01	-3.8482e+00	9.0123e-01
-1.3213e+00	-5.0601e-01	3.5856e+00	1.5093e+00	5.0847e+00
-2.7881e-01	1.9225e+00	-2.1609e+00	2.88545e+00	4.7121e+00
-2.4798e-01	4.8920e+00	-4.6760e+00	-3.6274e+00	1.4288e+00
-3.8854e+00	1.95497e+00	-2.1785e+00	1.6831e+00	1.4144e+00
9.9732e-01	2.7733e+00	4.6705e-01	3.8864e+00	-1.8923e+00
1.8371e+00	1.84403e+00	-3.5687e-02	-7.8854e+00	-1.7134e+00
1.1918e+00	9.6221e-01	8.7407e-01	2.7895e+00	-3.3480e-01
-4.8016e+00	1.6387e+00	3.9850e+00	-3.28973e+00	2.3480e+00
-2.7848e+00	3.8390e+00	-2.7234e+00	-6.3862e-01	1.60855e-01
1.4034e+00	3.9969e-01	3.8342e-01	-5.1668e+00	1.65737e+00
-3.8827e+00	6.1571e-01	3.7588e+00	0.3274e+00	1.60855e-01
3.7458e-01	5.1738e-01	3.1188e+00	-2.7780e+00	-2.4124e+00
1.8767e+00	7.2283e-01	3.4161e-01	-3.3750e+00	1.3860e+00
4.8298e+00	4.1370e+00	2.2900e-01	3.8802e+00	7.0640e-01
2.7488e+00	9.3609e-01	4.99125e-02	1.08843e+00	-9.4880e-01
1.1013e-01	5.8423e-01	-4.8857e+00	4.8883e-01	-2.2780e+00
5.4296e+00	3.2321e+00	-4.5235e-01	-2.5790e+00	1.2228e+00
-6.4536e-01	1.6333e+00	-7.4227e+00	1.1529e+00	-8.7355e+00
9.7443e-01	-2.8962e+00	-2.58337e+00	9.7747e-01	-2.5897e+00
1.2205e-01	3.2839e+00	3.5522e+00	-6.5280e+00	1.70278e+00
-1.8278e+00	8.7284e-01	-4.8627e-01	-5.8830e+00	-2.3524e+00
1.8829e+00	4.6620e-01	3.6622e+00	-7.3160e-01	-4.8826e+00
-3.1353e+00	-4.8549e-01	3.1880e-01	0.5380e+00	-2.2527e+00
-2.9609e+00	1.3549e+00	-2.2030e-01	0.8015e+00	-1.6831e+00

Fig. 10 Torch execution

CODE TEXT

```
[ ] 1.8371e+00, 1.0403e+00, -1.5687e-02, -3.8464e+00, -1.8923e+00,
1.3918e+00, -9.6214e-01, 8.7407e-01, -2.7053e+00, 1.7114e+00,
4.3016e+00, 1.6187e+00, 3.9850e+00, -3.2871e+00, 2.3340e-01,
-2.7848e+00, 3.8190e+00, -2.7234e+00, 1.5166e+00, -1.6685e-01,
1.4034e+00, 3.9969e-01, 1.4342e-01, 4.6302e-01, 1.6737e+00,
4.1382e+00, -6.3571e-01, 3.8344e-01, -2.8580e+00, 6.0324e-02,
-3.7545e-01, 5.1718e-01, -2.3188e+00, -2.7786e+00, -2.4126e+00,
1.9767e+00, 7.2283e-01, 1.4161e-01, -2.3750e+00, -1.1860e+00,
-4.6236e+00, 4.1376e+00, 7.2904e-01, -3.8902e-01, -7.9445e-01,
2.7480e+00, 9.3699e-01, 4.9126e-02, -1.0843e+00, -9.4800e-01,
1.3913e-01, -5.0425e-01, -2.4857e+00, -4.0091e-01, -2.2730e+00,
5.4206e+00, -3.2121e+00, -4.5218e-01, -5.2570e+00, -1.2220e+00,
-6.4534e-01, 1.6336e+00, 2.7427e+00, 1.1592e+00, 4.7355e+00,
-9.7443e-01, -2.8962e+00, -2.5817e+00, 9.7747e-01, -2.5897e+00,
-1.2205e-01, -3.2839e+00, 1.5522e+00, 1.6528e+00, -1.7078e+00,
-1.9278e+00, 8.7284e-01, -4.4627e-01, -1.5030e+00, 2.5324e+00,
3.0820e+00, -8.9503e-01, 1.6624e+00, -6.7316e-01, -2.4826e+00,
-2.1163e+00, -4.0629e-03, -1.1803e-01, 2.5300e+00, 2.2337e+00,
3.9004e+00, 1.4249e+00, 1.2203e+00, 2.0810e+00, -1.6858e+00]],
grad_fn=<AddmmBackward0>)
```

```
▶ with open('/content/imagenet_classes.txt') as f:
    labels = [line.strip() for line in f.readlines()]
    _, index = torch.max(out, 1)

percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
labels[index[0]], percentage[index[0]].item()

❏ ("323: 'monarch, monarch butterfly, milkweed butterfly, Danaus plexippus',",
99.99750518798828)
```

Fig.11 Final step in model

```
❏ ("323: 'monarch, monarch butterfly, milkweed butterfly, Danaus plexippus',",
99.99750518798828)
```

Fig.12 Output of the model

Sample Output:2

```
[10] from google.colab import files
      uploaded = files.upload()
      from PIL import Image
      img = Image.open("snake.jpeg")
      img
```

[Browse...](#) snake.jpeg

snake.jpeg(image/jpeg) - 7013 bytes, last modified: n/a - 100% done
Saving snake.jpeg to snake (1).jpeg



```
▶ img_t = preprocess(img)
```

```
[12] import torch
      batch_t = torch.unsqueeze(img_t, 0)
      resnet.eval()
```

Fig. 13 Execution of sample input 2

out

```
with open('/content/imagenet_classes.txt') as f:
    labels = [line.strip() for line in f.readlines()]
_, index = torch.max(out, 1)

percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
labels[index[0]], percentage[index[0]].item()
```

Conclusion

Image Recognition model worked effectively in classifying the image loaded. It can be used in various projects like image classifying , License plate recognition which will be useful in solving real world problems.

This model has a very good future scope in deep and machine learning involving objects like self-driving cars, and also it can be used to help the blind people recognizing the objects by making enhancements in the model.

References

at Google colab:

<https://colab.research.google.com/>

for deeplearning tutorial:

<https://machinelearningmastery.com>

er images:

<https://www.google.com>

*~**~*~*~*~*~*~*~*~*THANKYOU~*~**~*~*~*~*~*~*
