BIRDS IDENTIFICATION USING IMAGE PROCESSING

[1]A. CHARETHARDHA, [2]A. CHAITANYA NAGA SAI, [3]CH.AKHIL, [4]P. BABA RAKESH, [5]MR. R. AZHAGUSUNDARAM

[5]Guide
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING
BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Deemed to be University Estd u/s 3 of UGC Act, 1956)
CHENNAI 600 073, TAMILNADU, INDIA

Abstract-

- Identification of bird species is a challenging task often resulting in ambiguous labels.
- Although different bird species share the same basic set of parts, different bird species can vary dramatically in shape and appearance.
- Our project. aims to employ the power of deep learning to help amateur bird watchers identify Bird species from the images they capture

INTRODUCTION:

- In this presentation i am demonstrating a method using a convolutional neural network (CNN) to extract information from bird images captured previously or in real time by identifying local features.
- First, raw input data myriad semantic parts of birds were gathered and localized.
- Second, the feature vector of each part of the body is recognized, gathered and filter based on shape, size, color
- Third a CNN model was trained with bird pictures for feature extraction with consideration of some features and mentioned characteristics and subsequently the classified, trained data were stored to server to identify target object

LITERATURE SURVEY:

Author	Year	Title	Method Used	Key Findings
A. Marini, A. J. Turatti, A. S. Britto Jr., A. L. Koerich	2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)	Visual and Acoustic Identification of Bird Species	bird image and the resulting feature vector is classified by the multiclass SVM.	Acoustic features that are relevant to improve the identification of bird species based on bird image.
Aki Härmä	2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP'03)	Automatic Identification of Bird Species Based on Sinusoidal Modeling of Syllables	HMM	Automatic sound-based identification of bird species.
Andreia Marini, Jacques Facon and Alessandro L. Koerichy	2013 IEEE International Conference on Systems, Man, and Cybernetics	Bird Species Classification Based on Color Features	SVM, KNN	Bird species classification based on color. Feature extracted from unconstrained images
Marcelo T. Lopes, Lucas L. Gioppo, Thiago T. Higushi, Celso A. A. Kaestner	2011 IEEE International Symposium on Multimedia	Automatic Bird Species Identification for Large Number of Species	Decision tree with SVM, Back propagation neural network	Automatic bird species identification from bird audio recorded songs

EXISTING SYSTEM:

It uses the CNN (vgg16) algorithm to identify the bird species from image. In this the data is used directly without any preprocessing and vgg16 model has less layers so it can give accuracy upto 80% only.

Disadvantages: -

No data preprocessing is used.
It can identify only 200 bird species
Less accuracy (78%-80%)

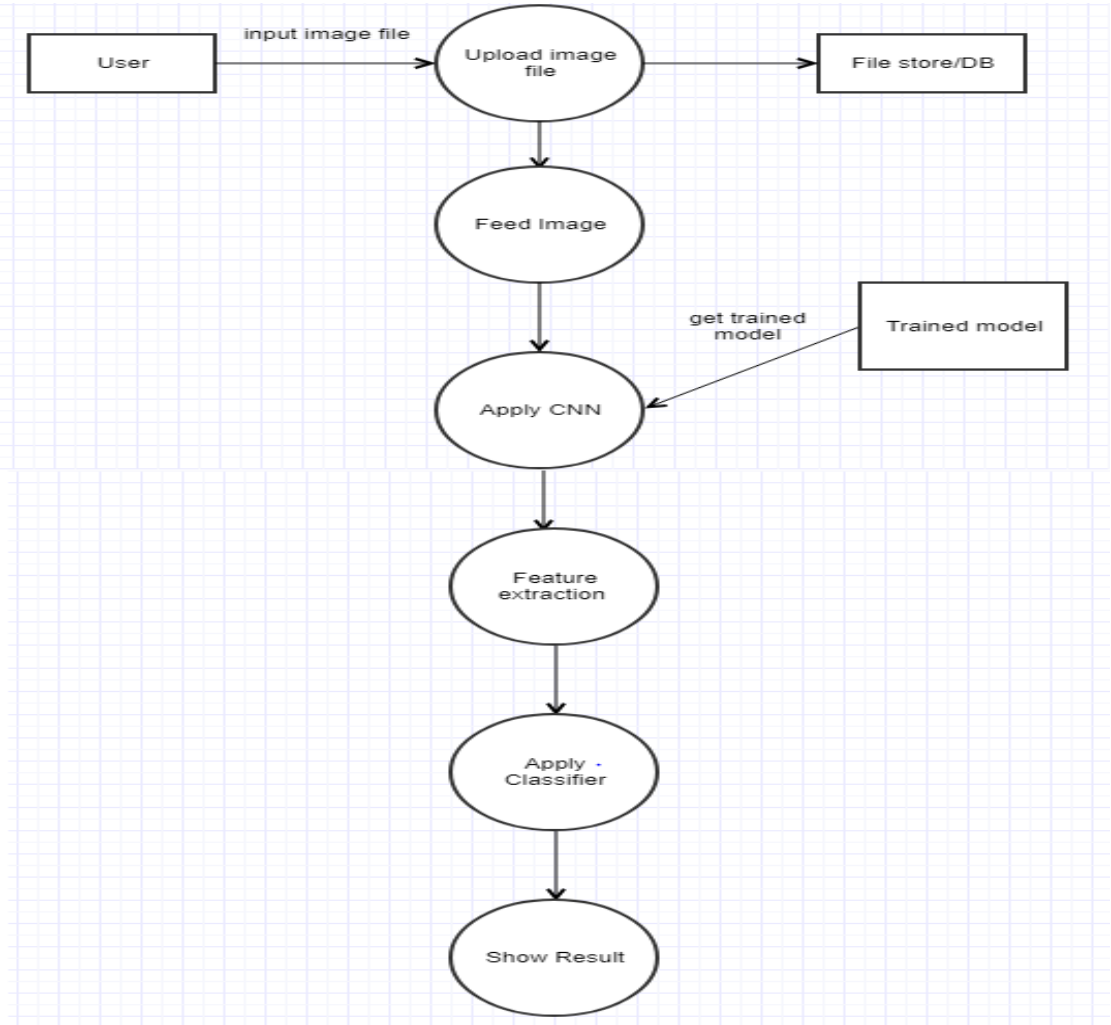
Proposed system:

- Our system employes CNN (Efficientnet b3) algorithm to identify the bird species from image and before feeding the data set into the program we use a preprocessing technique called Image data generator for increasing the accuracy of the program.

Advantages: -

- It can identify upto 325 bird species.
- Accuracy is above 90%.

ARCHITECTURE DIAGRAM:



REQUIREMENTS:

System: windows 10, 64bit
Software tool: Anaconda Navigator, Spyder
Programming language: Python

CHAPTER 4

Image Processing in Python: Algorithms Tools, and Methods You Should Know

Images define the world, each image has its own story, it contains a lot of crucial information that can be useful in many ways. This information can be obtained with the help of the technique known as **Image Processing**.

It is the core part of computer vision which plays a crucial role in many real-world examples like robotics, self-driving cars, and object detection. Image processing allows us to transform and manipulate thousands of images at a time and extract useful insights from them. It has a wide range of applications in almost every field.

Python is one of the widely used programming languages for this purpose. Its amazing libraries and tools help in achieving the task of image processing very efficiently.

Through this article, you will learn about classical algorithms, techniques, and tools to process the image and get the desired output. Let's get into it!

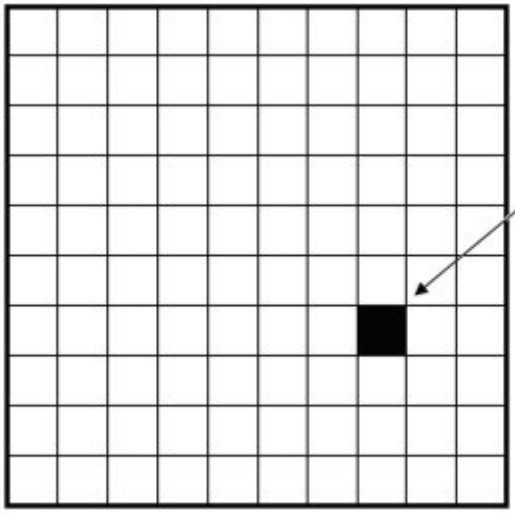
What is image processing?

As the name says, image processing means processing the image and this may include many different techniques until we reach our goal.

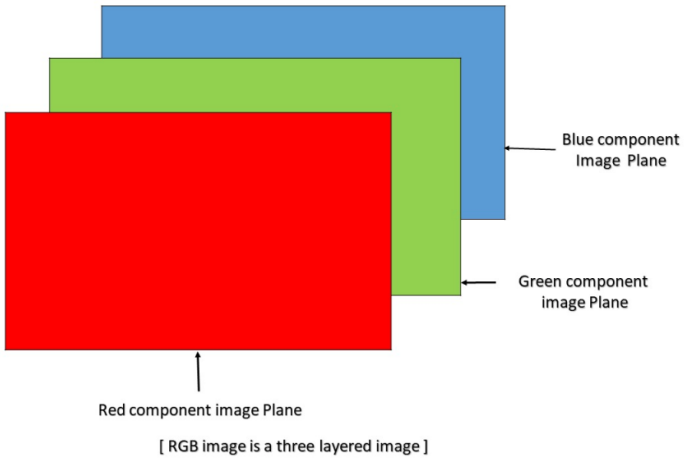
The final output can be either in the form of an image or a corresponding feature of that image. This can be used for further analysis and decision making.

But what is an image?

An image can be represented as a 2D function F(x,y) where x and y are spatial coordinates. The amplitude of F at a particular value of x,y is known as the intensity of an image at that point. If x,y, and the amplitude value is finite then we call it a digital image. It is an array of pixels arranged in columns and rows. Pixels are the elements of an image that contain information about intensity and color. An image can also be represented in 3D where x,y, and z become spatial coordinates. Pixels are arranged in the form of a matrix. This is known as an **RGB image**.



Source



Source

There are various types of images:

- RGB image: It contains three layers of 2D image, these layers are Red, Green, and Blue channels.
- Grayscale image: These images contain shades of black and white and contain only a single channel.

Classic image processing algorithms

1. Morphological Image Processing

Morphological image processing tries to remove the imperfections from the binary images because binary regions produced by simple thresholding can be distorted by noise. It also helps in smoothing the image using opening and closing operations.

Morphological operations can be extended to grayscale images. It consists of non-linear operations related to the structure of features of an image. It depends on the related ordering of pixels but on their numerical values. This technique analyzes an image using a small template known as **structuring element** which is placed on different possible locations in the image and is compared with the corresponding neighbourhood pixels. A structuring element is a small matrix with 0 and 1 values.

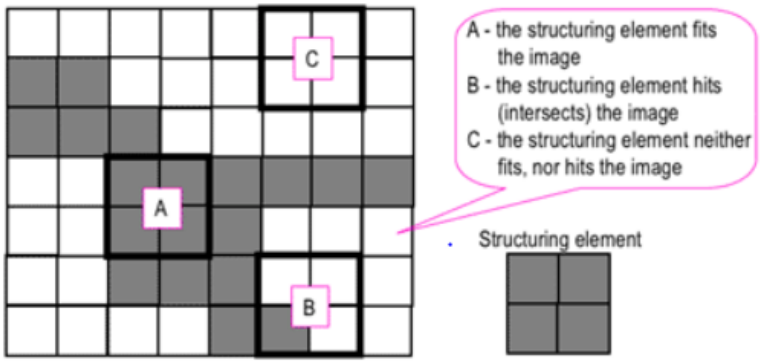
Let's see the two fundamental operations of morphological image processing, **Dilation and Erosion:**

- **dilation** operation adds pixels to the boundaries of the object in an image
- **erosion** operation removes the pixels from the object boundaries.

The number of pixels removed or added to the original image depends on the size of the structuring element.

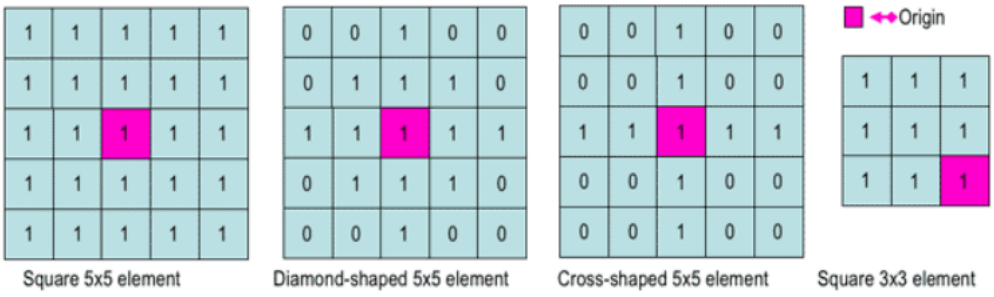
At this point you may be thinking "what is a structuring element?" Let me explain:

Structuring element is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size. It is positioned at all possible locations in the image and it is compared with the corresponding neighbourhood of pixels.

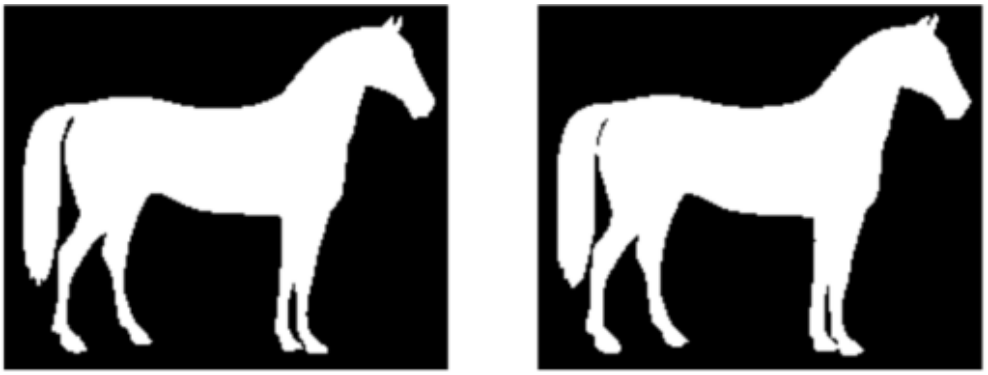


Source

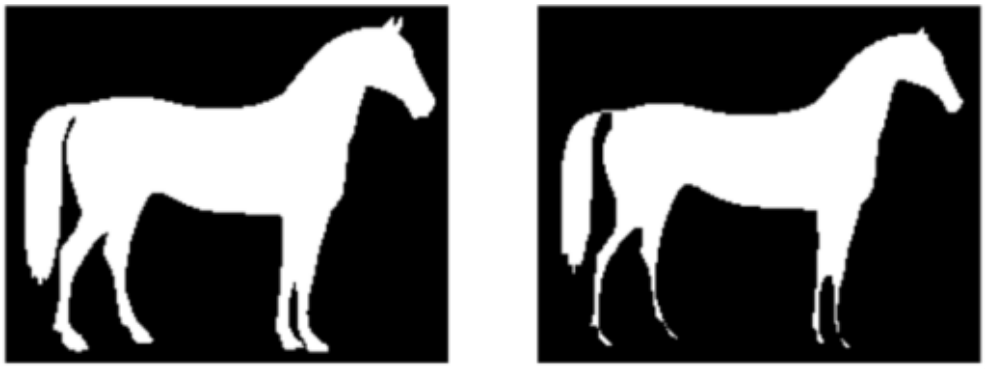
The square structuring element 'A' fits in the object we want to select, the 'B' intersects the object and 'C' is out of the object. The zero-one pattern defines the configuration of the structuring element. It's according to the shape of the object we want to select. The center of the structuring element identifies the pixel being processed.



Source



Dilation | Source



Erosion | Source

2. Gaussian Image Processing

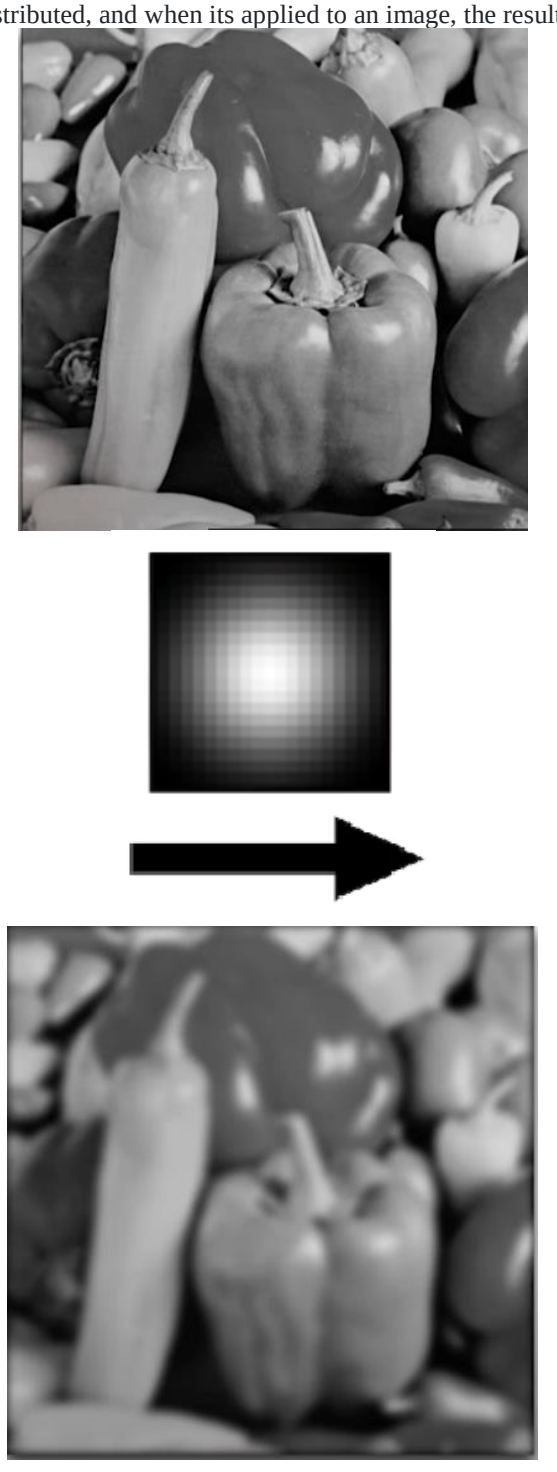
Gaussian blur which is also known as gaussian smoothing, is the result of blurring an **image** by a **Gaussian** function.

It is **used to reduce image noise and reduce details**. The visual effect of this blurring technique is similar to looking at an image through the translucent screen. It is sometimes used in computer vision for image enhancement at different scales or as a data augmentation technique in deep learning.

The basic gaussian function looks like:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

In practice, it is best to take advantage of the Gaussian blur's separable property by dividing the process into two passes. In the first pass, a one-dimensional kernel is used to blur the image in only the horizontal or vertical direction. In the second pass, the same one-dimensional kernel is used to blur in the remaining direction. The resulting effect is the same as convolving with a two-dimensional kernel in a single pass. Let's see an example to understand what gaussian filters do to an image.
If we have a filter which is normally distributed, and when its applied to an image, the results look like this:

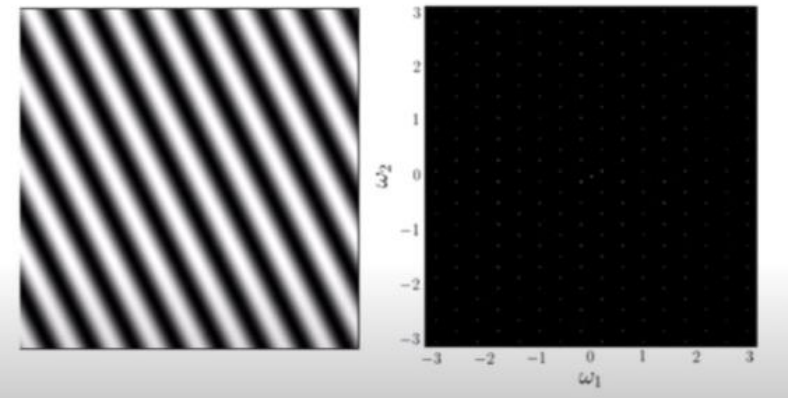


Original
Filter
Result
Source
You can see that some of the edges have little less detail. The filter is giving more weight to the pixels at the center than the pixels away from the center. Gaussian filters are low-pass filters i.e. weakens the high frequencies. It is commonly used in edge detection.

3. Fourier Transform in image processing
Fourier transform breaks down an image into sine and cosine components.
It has multiple applications like image reconstruction, image compression, or image filtering.
Since we are talking about images, we will take discrete fourier transform into consideration.
Let's consider a sinusoid, it comprises of three things:

- Magnitude – related to contrast
- Spatial frequency – related to brightness
- Phase – related to color information

The image in the frequency domain looks like this:



Source

The formula for 2D discrete fourier transform is:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

In the above formula, f(x,y) denotes the image.

The inverse fourier transform converts the transform back to image. The formula for 2D inverse discrete fourier transform is:

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

4. Edge Detection in image processing

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness.

This could be very beneficial in extracting useful information from the image because most of the shape information is enclosed in the edges. Classic edge detection methods work by detecting discontinuities in the brightness.

It can rapidly react if some noise is detected in the image while detecting the variations of grey levels. Edges are defined as the local maxima of the gradient.

The most common edge detection algorithm is **sobel edge detection algorithm**. Sobel detection operator is made up of 3*3 convolutional kernels. A simple kernel Gx and a 90 degree rotated kernel Gy. Separate measurements are made by applying both the kernel separately to the image.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * Image\ matrix$$

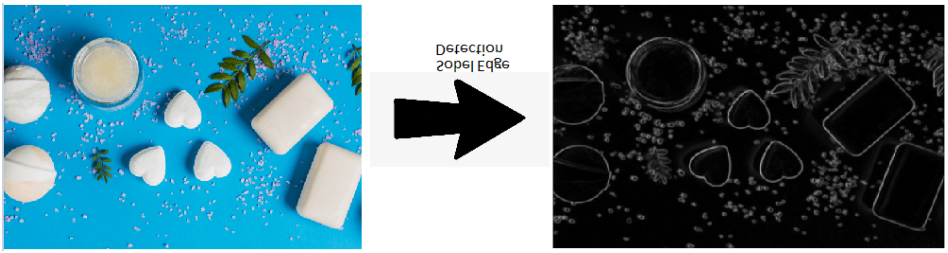
And,

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * Image\ matrix$$

* denotes the 2D signal processing convolution operation.

Resulting gradient can be calculated as:

$$G = sqrt(Gx^2 + Gy^2)$$



Source

5. Wavelet Image Processing

We saw a Fourier transform but it is only limited to the frequency. Wavelets take both time and frequency into the consideration. This transform is apt for non-stationary signals.

We know that edges are one of the important parts of the image, while applying the traditional filters it's been noticed that noise gets removed but image gets blurry. The wavelet transform is designed in such a way that we get good frequency resolution for low frequency components. Below is the 2D wavelet transform example:

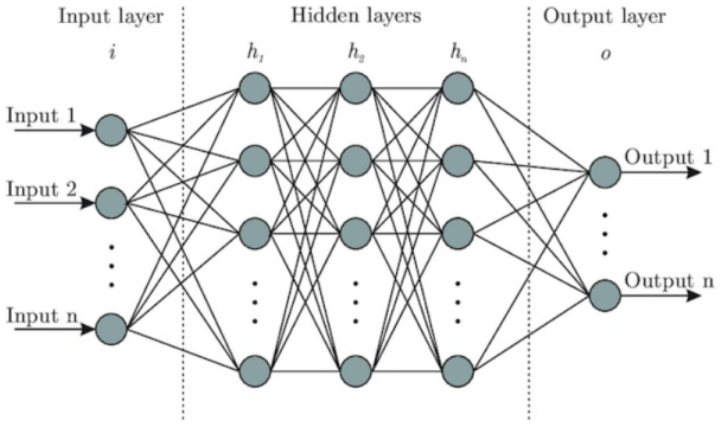
Source

Image processing using Neural Networks

Neural Networks are multi-layered networks consisting of neurons or nodes. These neurons are the core processing units of the neural network. They are designed to act like human brains. They take in data, train themselves to recognize the patterns in the data and then predict the output.

A basic neural network has three layers:

1. Input layer
2. Hidden layer
3. Output layer



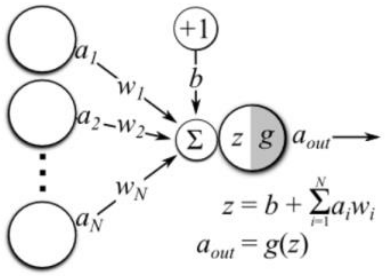
Basic neural network | Source

The input layers receive the input, the output layer predicts the output and the hidden layers do most of the calculations. The number of hidden layers can be modified according to the requirements. There should be atleast one hidden layer in a neural network.

The basic working of the neural network is as follows:

1. Let's consider an image, each pixel is fed as input to each neuron of the first layer, neurons of one layer are connected to neurons of the next layer through channels.
2. Each of these channels is assigned a numerical value known as weight.
3. The inputs are multiplied by the corresponding weights and this weighted sum is then fed as input to the hidden layers.
4. The output from the hidden layers is passed through an activation function which will determine whether the particular neuron will be activated or not.
5. The activated neurons transmits data to the next hidden layers. In this manner, data is propagated through the network, this is known as Forward Propagation.
6. In the output layer, the neuron with the highest value predicts the output. These outputs are the probability values.
7. The predicted output is compared with the actual output to obtain the error. This information is then transferred back through the network, the process is known as Backpropagation.

8. Based on this information, the weights are adjusted. This cycle of forward and backward propagation is done several times on multiple inputs until the network predicts the output correctly in most of the cases.
9. This ends the training process of the neural network. The time taken to train the neural network may get high in some cases.
In the below image, **ai's** is the set of inputs, **wi's** are the weights, **z** is the output and **g** is any activation function.



Operations in a single neuron | Source

Here are some guidelines to prepare data for image processing.

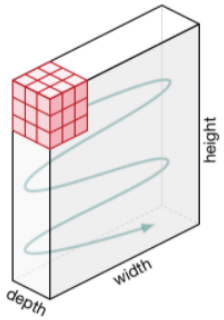
- More data needs to be fed to the model to get the better results.
- Image dataset should be of high quality to get more clear information, but to process them you may require deeper neural networks.
- In many cases RGB images are converted to grayscale before feeding them into a neural network.

Types of Neural Network

Convolutional Neural Network

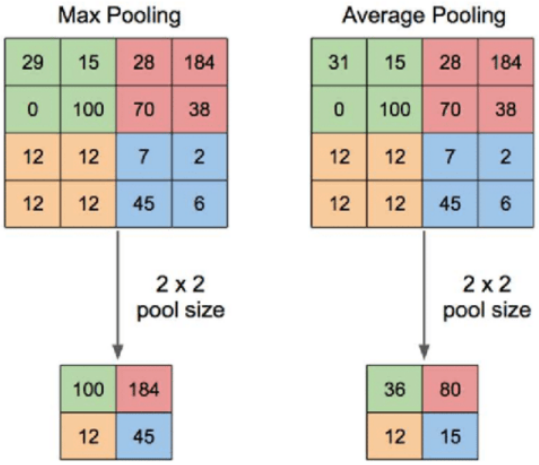
A convolutional neural network, ConvNets in short has three layers:

- **Convolutional Layer** (*CONV*): They are the core building block of CNN, it is responsible for performing convolution operation.The element involved in carrying out the convolution operation in this layer is called the **Kernel/Filter (matrix).** The kernel makes horizontal and vertical shifts based on the **stride rate** until the full image is traversed.



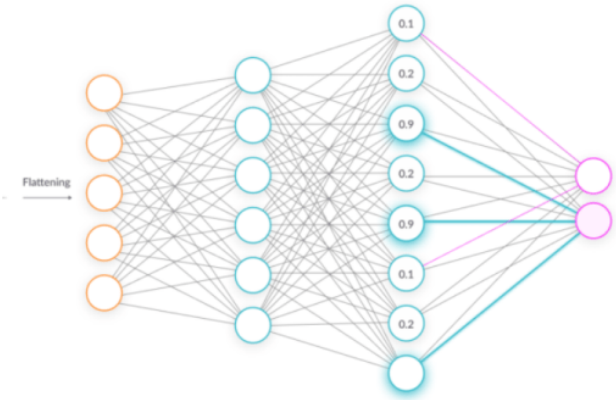
Movement of the kernel | Source

- **Pooling Layer** (*POOL*): This layer is responsible for dimensionality reduction. It helps to decrease the computational power required to process the data. There are two types of Pooling: Max Pooling and Average Pooling. Max pooling returns the maximum value from the area covered by the kernel on the image. Average pooling returns the average of all the values in the part of the image covered by the kernel.



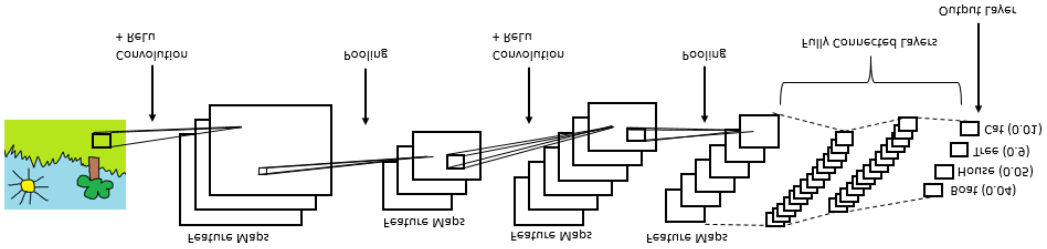
Pooling operation | Source

- **Fully Connected Layer** (*FC*): The **fully connected layer** (**FC**) operates on a flattened input where each input is connected to all neurons. If present, **FC layers** are usually found towards the end of **CNN** architectures.



Fully connected layers | *Source*

CNN is mainly used in extracting features from the image with help of its layers. CNNs are widely used in image classification where each input image is passed through the series of layers to get a probabilistic value between 0 and 1.



Source

Generative Adversarial Networks

Generative models use an unsupervised learning approach (there are images but there are no labels provided).

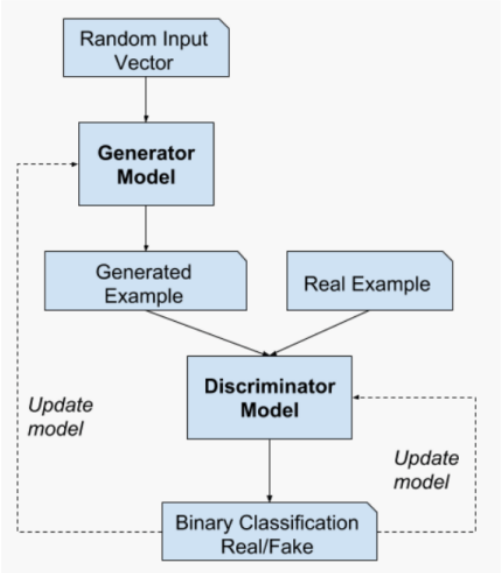
GANs are composed of two models **Generator** and **Discriminator.** *Generator* learns to make fake images that look realistic so as to fool the discriminator and *Discriminator* learns to distinguish fake from real images (it tries not to get fooled).

Generator is not allowed to see the real images, so it may produce poor results in the starting phase while the discriminator is allowed to look at real images but they are jumbled with the fake ones produced by the generator which it has to classify as real or fake.

Some noise is fed as input to the generator so that it's able to produce different examples every single time and not the same type image. Based on the scores predicted by the discriminator, the generator tries to improve its results, after a certain point of time, the generator will be able to produce images that will be harder to distinguish, at that point of time, the user gets satisfied with its results. Discriminator also improves itself as it gets more and more realistic images at each round from the generator.

Popular types of GANs are Deep Convolutional GANs(DCGANs), Conditional GANs(cGANs), StyleGANs, CycleGAN, DiscoGAN, GauGAN and so on.

GANs are great for image generation and manipulation. Some applications of GANs include : Face Aging, Photo Blending, Super Resolution, Photo Inpainting, Clothing Translation.



Source

Image processing tools

1. OpenCV

It stands for Open Source Computer Vision Library. This library consists of around 2000+ optimised algorithms that are useful for computer vision and machine learning. There are several ways you can use opencv in image processing, a few are listed below:

- Converting images from one color space to another i.e. like between BGR and HSV, BGR and gray etc.
- Performing thresholding on images, like, simple thresholding, adaptive thresholding etc.
- Smoothing of images, like, applying custom filters to images and blurring of images.
- Performing morphological operations on images.
- Building image pyramids.
- Extracting foreground from images using GrabCut algorithm.
- Image segmentation using watershed algorithm.

Refer to this link for more details.

2. Scikit-image

It is an open-source library used for image preprocessing. It makes use of machine learning with built-in functions and can perform complex operations on images with just a few functions.

It works with numpy arrays and is a fairly simple library even for those who are new to python. Some operations that can be done using scikit image are :

- To implement thresholding operations use **try_all_threshold()** method on the image. It will use seven global thresholding algorithms. This is in the **filters** module.
- To implement edge detection use **sobel()** method in the **filters** module. This method requires a 2D grayscale image as an input, so we need to convert the image to grayscale.
- To implement gaussian smoothing use **gaussian()** method in the **filters** module.
- To apply histogram equalization, use **exposure** module, to apply normal histogram equalization to the original image, use **equalize_hist()** method and to apply adaptive equalization, use **equalize_adapthist()** method.
- To rotate the image use **rotate()** function under the **transform** module.
- To rescale the image use **rescale()** function from the **transform** module.
- To apply morphological operations use **binary_erosion()** and **binary_dilation()** function under the **morphology** module.

3. PIL/pillow

PIL stands for Python Image Library and **Pillow** is the friendly PIL fork by Alex Clark and Contributors. It's one of the powerful libraries. It supports a wide range of image formats like PPM, JPEG, TIFF, GIF, PNG, and BMP.

It can help you perform several operations on images like rotating, resizing, cropping, grayscaling etc. Let's go through some of those operations

To carry out manipulation operations there is a module in this library called **Image.**

- To load an image use the **open()** method.
- To display an image use **show()** method.
- To know the file format use **format** attribute
- To know the size of the image use **size** attribute
- To know about the pixel format use **mode** attribute.
- To save the image file after desired processing, use **save()** method. Pillow saves the image file in *png* format.
- To resize the image use **resize()** method that takes two arguments as width and height.
- To crop the image, use **crop()** method that takes one argument as a box tuple that defines position and size of the cropped region.
- To rotate the image use **rotate()** method that takes one argument as an integer or float number representing the degree of rotation.
- To flip the image use **transform()** method that take one argument among the following: Image.FLIP_LEFT_RIGHT, Image.FLIP_TOP_BOTTOM, Image.ROTATE_90, Image.ROTATE_180, Image.ROTATE_270.

READ ALSO

4. NumPy

With this library you can also perform simple image techniques, such as flipping images, extracting features, and analyzing them.

Images can be represented by numpy multi-dimensional arrays and so their type is **NdArrays**. A color image is a numpy array with 3 dimensions. By slicing the multi-dimensional array the RGB channels can be separated.

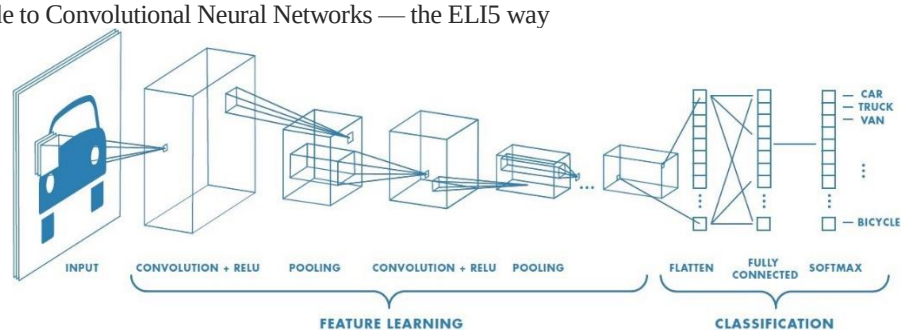
Below are some of the operations that can be performed using NumPy on the image (image is loaded in a variable named **test_img** using imread).

- To flip the image in a vertical direction, use **np.flipud(test_img).**
- To flip the image in a horizontal direction, use **np.fliplr(test_img).**
- To reverse the image, use **test_img[::-1]** (the image after storing it as the numpy array is named as <img_name>).
- To add filter to the image you can do this:

  Example: **np.where(test_img > 150, 255, 0)**, this says that in this picture if you find anything with 150, then replace it with 255, else 0.
- You can also display the RGB channels separately. It can be done using this code snippet:

  To obtain a red channel, do **test_img[:,:,0]**, to obtain a green channel, do **test_img[:,:,1]** and to obtain a blue channel, do **test_img[:,:,2].**

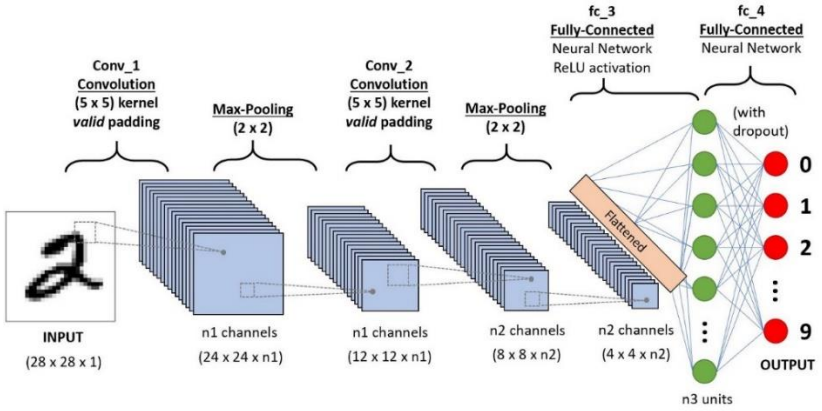
A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way



Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision.

The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a **Convolutional Neural Network**.

Introduction

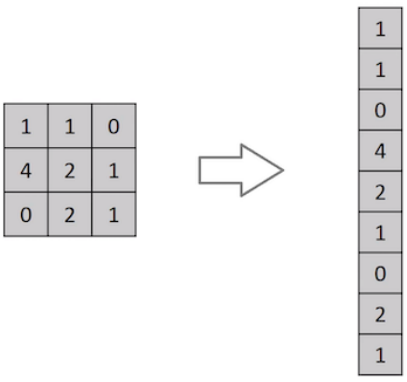


A CNN sequence to classify handwritten digits

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Why ConvNets over Feed-Forward Neural Nets?



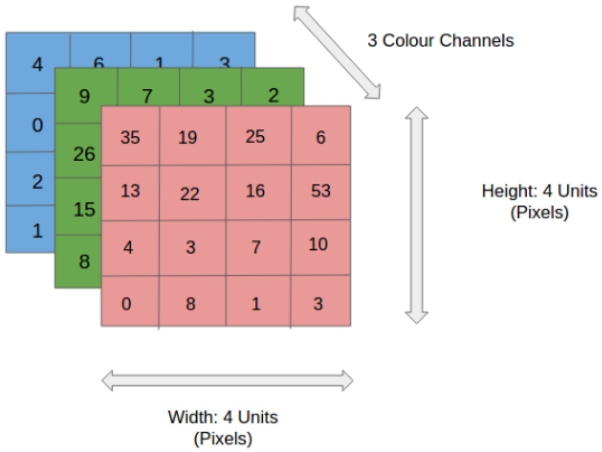
Flattening of a 3x3 image matrix into a 9x1 vector

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? Uh.. not really.

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Input Image

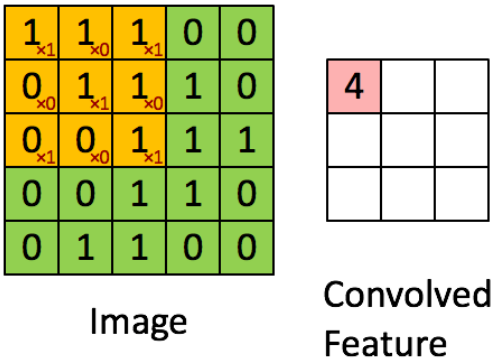


4x4x3 RGB Image

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Convolution Layer — The Kernel



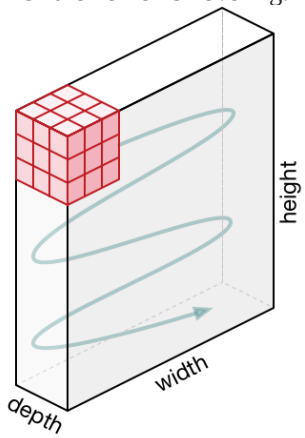
Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

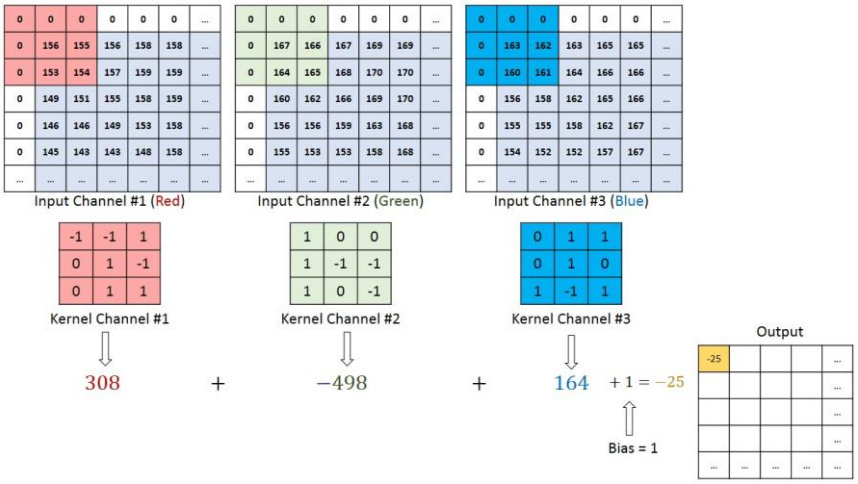
Kernel/Filter, K = 1 0 1
0 1 0
1 0 1

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.



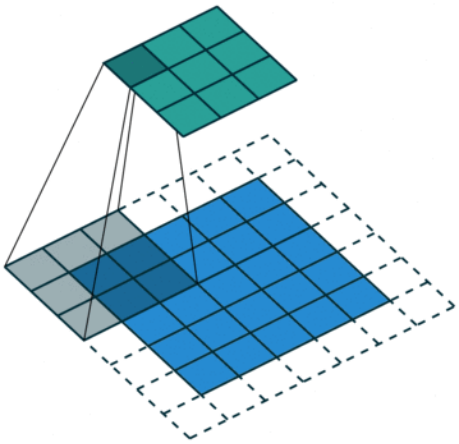
Movement of the Kernel

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.



Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

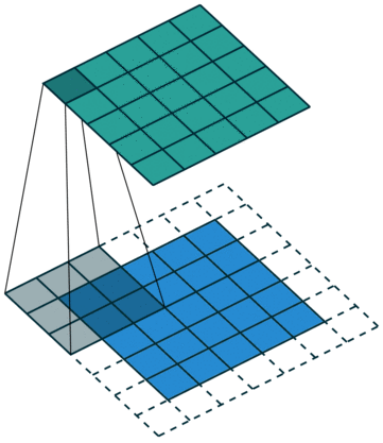
In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.



Convolution Operation with Stride Length = 2

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

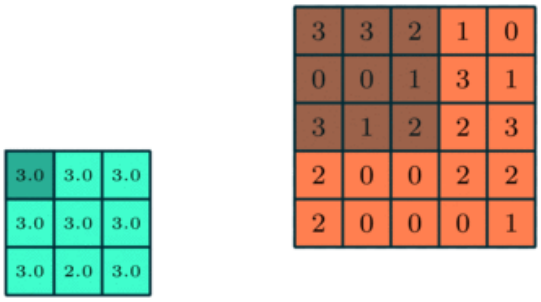


SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image

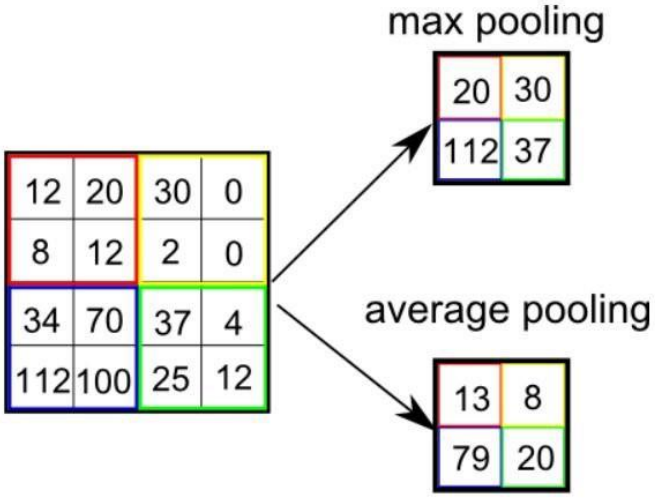
When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding**.
On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.
The following repository houses many such GIFs which would help you get a better understanding of how Padding and Stride Length work together to achieve results relevant to our needs.

vdumoulin/conv_arithmetic
A technical report on convolution arithmetic in the context of deep learning - vdumoulin/conv_arithmetic
github.com

Pooling Layer

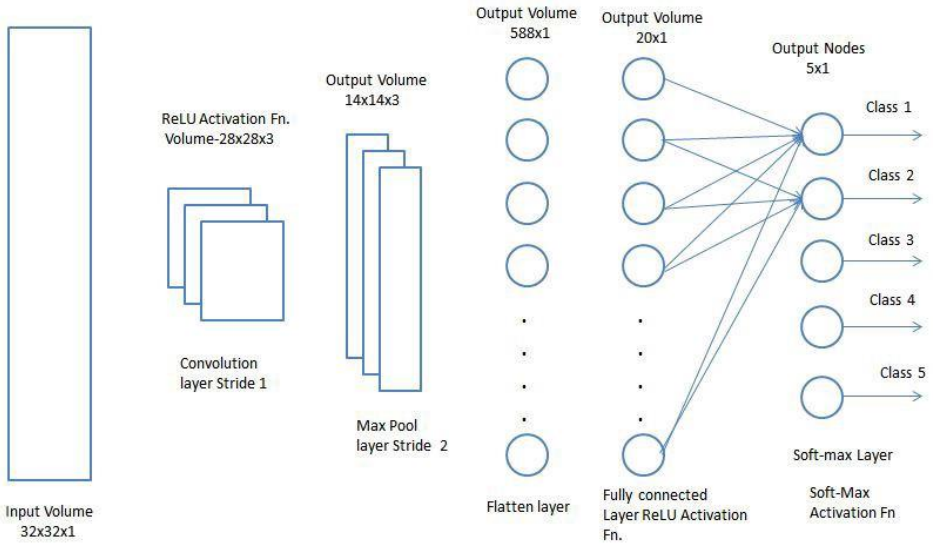


3x3 pooling over 5x5 convolved feature
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.
There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.
Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.



Types of Pooling
The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.
After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Classification — Fully Connected Layer (FC Layer)



Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.
Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

Python:

PYTHON (PROGRAMMING LANGUAGE)

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation

HISTORY

Python was conceived in the late 1980s, and its implementation began in December 1989 [28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL). About the origin of Python, Van Rossum wrote in 1996:Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.xand 2.7.x version series. The End Of Life date (EOL, sunset date) for Python 2.7 was initially set at 2015, then postponed to 2020 out of concern that a large body of existing code cannot easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life but Google cited performance under concurrent workloads as their only motivation. Features and philosophy Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other 6/13/2017 Python (programming language) paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers some support for functional

programming in the Lisp tradition. The language has map(), reduce() and filter() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML. The core philosophy of the language is summarized by the document The Zen of Python (PEP 20), which includes aphorisms such as: Beautiful is better than ugly Explicit is better than implicit Simple is better than complex Complex is better than complicated Readability counts Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it". Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity.When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar. A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example: List comprehensions vs. for-loops Conditional expressions vs. if blocks The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements. 6/13/2017 Python (programming language) - Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python. Methods Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Typing Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them. Python allows programmers to define their own types using classes, which are most often used for objectoriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta programming and reflection. Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0. The long term plan is to support gradual typing and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

Mathematics Python has the usual C arithmetic operators (+, -, *, /, %). It also has ** for exponentiation, e.g. 5**3 == 125 and 9**0.5 == 3.0, and a new matrix multiply @ operator is included in version 3.5. [71] Additionally, it has a unary operator (~), which essentially inverts all the bytes of its one argument. For integers, this means ~x=-x-1. Other operators include bitwise shift operators x << y, which shifts x to the left y places, the same as x*(2**y) , and x >> y, which shifts x to the right y places, the same as x/(2**y) . [73] The behavior of division has changed significantly over time: Python 2.1 and earlier use the C division behavior. The / operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. 7 / 3 == 2 and -7 / 3 == -2. Python 2.2 changes integer division to round towards negative infinity, e.g. 7 / 3 == 2 and -7 / 3 == -3. The floor division // operator is introduced. So 7 // 3 == 2, -7 // 3 == -3, 7.5 // 3 == 2.0 and 6/13/2017 Python (programming language) - Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 8/19 -7.5 // 3 == -3.0. Adding from __future__ import division causes a module to use Python 3.0 rules for division (see next). Python 3.0 changes / to be always floating-point division. In Python terms, the pre-3.0 / is classic division, the version-3.0 / is real division, and // is floor division. Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation (a+b) // b == a // b + 1 is always true. It also means that the equation b * (a // b) + a % b == a is valid for both positive and negative values of a. However, maintaining the validity of this equation means that while the result of a % b is, as expected, in the half-open interval [0, b), where b is a positive integer, it has to lie in the interval (b, 0] when b is negative. Python provides a round function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: round(0.5) is 1.0, round(-

0.5) is −1.0. Python 3 uses round to even: round(1.5) is 2, round(2.5) is 2. Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression a < b < c tests whether a is less than b and b is less than c. Cderived languages interpret this expression differently: in C, the expression would first evaluate a < b, resulting in 0 or 1, and that result would then be compared with c. Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type int, to arbitrary precision, belonging to the python type long, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the int and long types was eliminated; this behavior is now entirely contained by the int class.) The Decimal type/class in module decimal (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes.The Fraction type in module fractions (since version 2.6) provides arbitrary precision for rational numbers. Due to Python's extensive mathematics library, and the third-party library NumPy which further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation. Libraries Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it.

It is a type of signal dispensation in which input is an image, like video frame or photograph and output may be image or characteristics associated with that image.

Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

ADVANTAGES OF IMAGE PROCESSING:

The purpose of image processing is divided into 5 groups.

They are:

VISUALIZATION - Observe the objects that are not visible.

IMAGE SHARPENING AND RESTORATION - To create a better image.

IMAGE RETRIEVAL - Seek for the image of interest.

MEASUREMENT OF PATTERN – Measures various objects in an image.

IMAGE RECOGNITION – Distinguish the objects in an image.

FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:

1. IMAGE ACQUISITION:

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves pre-processing, such as scaling etc.

2. IMAGE ENHANCEMENT:

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. IMAGE RESTORATION:

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

4. COLOR IMAGE PROCESSING

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modelling and processing in a digital domain etc.

5. WAVELETS AND MULTIRESOLUTION PROCESSING

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

6. COMPRESSION

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

7. MORPHOLOGICAL PROCESSING

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. SEGMENTATION

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

9. REPRESENTATION AND DESCRIPTION

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

10. OBJECT RECOGNITION

Recognition is the process that assigns a label, such as, "vehicle" to an object based on its descriptors

How do I run a Python program under Windows?

This is not necessarily a straightforward question. If you are already familiar with running programs from the Windows command line then everything will seem obvious; otherwise, you might need a little more guidance.

Unless you use some sort of integrated development environment, you will end up *typing* Windows commands into what is variously referred to as a "DOS window" or "Command prompt window". Usually you can create such a window from your search bar by searching for cmd. You should be able to recognize when you have started such a window because you will see a Windows "command prompt", which usually looks like this:

```
C:\>
```

The letter may be different, and there might be other things after it, so you might just as easily see something like:

```
D:\YourName\Projects\Python>
```

depending on how your computer has been set up and what else you have recently done with it. Once you have started such a window, you are well on the way to running Python programs.

You need to realize that your Python scripts have to be processed by another program called the Python *interpreter*. The interpreter reads your script, compiles it into bytecodes, and then executes the bytecodes to run your program. So, how do you arrange for the interpreter to handle your Python?

First, you need to make sure that your command window recognises the word "py" as an instruction to start the interpreter. If you have opened a command window, you should try entering the command py and hitting return:

```
C:\Users\YourName> py
```

You should then see something like:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You have started the interpreter in "interactive mode". That means you can enter Python statements or expressions interactively and have them executed or evaluated while you wait. This is one of Python's strongest features. Check it by entering a few expressions of your choice and seeing the results:

```
>>>
```

```
>>> print("Hello")
Hello
>>> "Hello" * 3
'HelloHelloHello'
```

Many people use the interactive mode as a convenient yet highly programmable calculator. When you want to end your interactive Python session, call the exit() function or hold the Ctrl key down while you enter a Z, then hit the "Enter" key to get back to your Windows command prompt.

You may also find that you have a Start-menu entry such as Start ‣ Programs ‣ Python 3.x ‣ Python (command line) that results in you seeing the >>> prompt in a new window. If so, the window will disappear after you call the exit() function or enter the Ctrl-Z character; Windows is running a single "python" command in the window, and closes it when you terminate the interpreter.

Now that we know the py command is recognized, you can give your Python script to it. You'll have to give either an absolute or a relative path to the Python script. Let's say your Python script is located in your desktop and is named hello.py, and your command prompt is nicely opened in your home directory so you're seeing something similar to:

```
C:\Users\YourName>
```

So now you'll ask the py command to give your script to Python by typing py followed by your script path:

```
C:\Users\YourName> py Desktop\hello.py
hello
```

How do I make Python scripts executable?

On Windows, the standard Python installer already associates the .py extension with a file type (Python.File) and gives that file type an open command that runs the interpreter (D:\Program Files\Python\python.exe "%1" %*). This is enough to make scripts executable from the command prompt as 'foo.py'. If you'd rather be able to execute the script by simple typing 'foo' with no extension you need to add .py to the PATHEXT environment variable.

Why does Python sometimes take so long to start?

Usually Python starts very quickly on Windows, but occasionally there are bug reports that Python suddenly begins to take a long time to start up. This is made even more puzzling because Python will work fine on other Windows systems which appear to be configured identically.

The problem may be caused by a misconfiguration of virus checking software on the problem machine. Some virus scanners have been known to introduce startup overhead of two orders of magnitude when the scanner is configured to monitor all reads from the filesystem. Try checking the configuration of virus scanning software on your systems to ensure that they are indeed configured identically. McAfee, when configured to scan all file system read activity, is a particular offender.

How do I make an executable from a Python script?

See How can I create a stand-alone binary from a Python script? for a list of tools that can be used to make executables.

Is a *.pyd file the same as a DLL?

Yes, .pyd files are dll's, but there are a few differences. If you have a DLL named foo.pyd, then it must have a function PyInit_foo(). You can then write Python "import foo", and Python will search for foo.pyd (as well as foo.py, foo.pyc) and if it finds it, will attempt to call PyInit_foo() to initialize it. You do not link your .exe with foo.lib, as that would cause Windows to require the DLL to be present.

Note that the search path for foo.pyd is PYTHONPATH, not the same as the path that Windows uses to search for foo.dll. Also, foo.pyd need not be present to run your program, whereas if you linked your program with a dll, the dll is required. Of course, foo.pyd is required if you want to say import foo. In a DLL, linkage is declared in the source code with __declspec(dllexport). In a .pyd, linkage is defined in a list of available functions.

How can I embed Python into a Windows application?

Embedding the Python interpreter in a Windows app can be summarized as follows:

1. Do _not_ build Python into your .exe file directly. On Windows, Python must be a DLL to handle importing modules that are themselves DLL's. (This is the first key undocumented fact.) Instead, link to python*NN*.dll; it is typically installed in C:\Windows\System. *NN* is the Python version, a number such as "33" for Python 3.3.

   You can link to Python in two different ways. Load-time linking means linking against python*NN*.lib, while run-time linking means linking against python*NN*.dll. (General note: python*NN*.lib is the so-called "import lib" corresponding to python*NN*.dll. It merely defines symbols for the linker.)

   Run-time linking greatly simplifies link options; everything happens at run time. Your code must load python*NN*.dll using the Windows LoadLibraryEx() routine. The code must also use access routines and data in python*NN*.dll (that is, Python's C API's) using pointers obtained by the Windows GetProcAddress() routine. Macros can make using these pointers transparent to any C code that calls routines in Python's C API.

   Borland note: convert python*NN*.lib to OMF format using Coff2Omf.exe first.
2. If you use SWIG, it is easy to create a Python "extension module" that will make the app's data and methods available to Python. SWIG will handle just about all the grungy details for you. The result is C code that you link *into* your .exe file (!) You do _not_ have to create a DLL file, and this also simplifies linking.
3. SWIG will create an init function (a C function) whose name depends on the name of the extension module. For example, if the name of the module is leo, the init function will be called initleo(). If you use SWIG shadow classes, as you should, the init function will be called initleoc(). This initializes a mostly hidden helper class used by the shadow class.

   The reason you can link the C code in step 2 into your .exe file is that calling the initialization function is equivalent to importing the module into Python! (This is the second key undocumented fact.)
4. In short, you can use the following code to initialize the Python interpreter with your extension module.

```
#include "python.h"
...
Py_Initialize();  // Initialize Python.
initmyAppc();  // Initialize (import) the helper class.
PyRun_SimpleString("import myApp");  // Import the shadow class.
```

10. There are two problems with Python's C API which will become apparent if you use a compiler other than MSVC, the compiler used to build pythonNN.dll.

    Problem 1: The so-called "Very High Level" functions that take FILE * arguments will not work in a multi-compiler environment because each compiler's notion of a struct FILE will be different. From an implementation standpoint these are very _low_ level functions.

    Problem 2: SWIG generates the following code when generating wrappers to void functions:

```
Py_INCREF(Py_None);
_resultobj = Py_None;
return _resultobj;
```

    Alas, Py_None is a macro that expands to a reference to a complex data structure called _Py_NoneStruct inside pythonNN.dll. Again, this code will fail in a mult-compiler environment. Replace such code by:

```
return Py_BuildValue("");
```

    It may be possible to use SWIG's %typemap command to make the change automatically, though I have not been able to get this to work (I'm a complete SWIG newbie).
11. Using a Python shell script to put up a Python interpreter window from inside your Windows app is not a good idea; the resulting window will be independent of your app's windowing system. Rather, you (or the wxPythonWindow class) should create a "native" interpreter window. It is easy to connect that window to the Python interpreter. You can redirect Python's i/o to _any_ object that supports read and write, so all you need is a Python object (defined in your extension module) that contains read() and write() methods.

How do I keep editors from inserting tabs into my Python source?

The FAQ does not recommend using tabs, and the Python style guide, **PEP 8**, recommends 4 spaces for distributed Python code; this is also the Emacs python-mode default.

Under any editor, mixing tabs and spaces is a bad idea. MSVC is no different in this respect, and is easily configured to use spaces: Take Tools ▸ Options ▸ Tabs, and for file type "Default" set "Tab size" and "Indent size" to 4, and select the "Insert spaces" radio button.
Python raises IndentationError or TabError if mixed tabs and spaces are causing problems in leading whitespace. You may also run the tabnanny module to check a directory tree in batch mode.
How do I check for a keypress without blocking?
Use the msvcrt module. This is a standard Windows-specific extension module. It defines a function kbhit() which checks whether a keyboard hit is present, and getch() which gets one character without echoing it.

OPENCV
OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.
OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe

HISTORY:
Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.
The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months[1] and development is now done by an independent Russian team supported by commercial corporations.
In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.
On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV

APPLICATIONS:
OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)

- Deep neural networks (DNN)

PROGRAMMING LANGUAGE

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

Since version 3.4, **OpenCV.js** is a JavaScript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface

HARDWARE ACCELERATION

If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

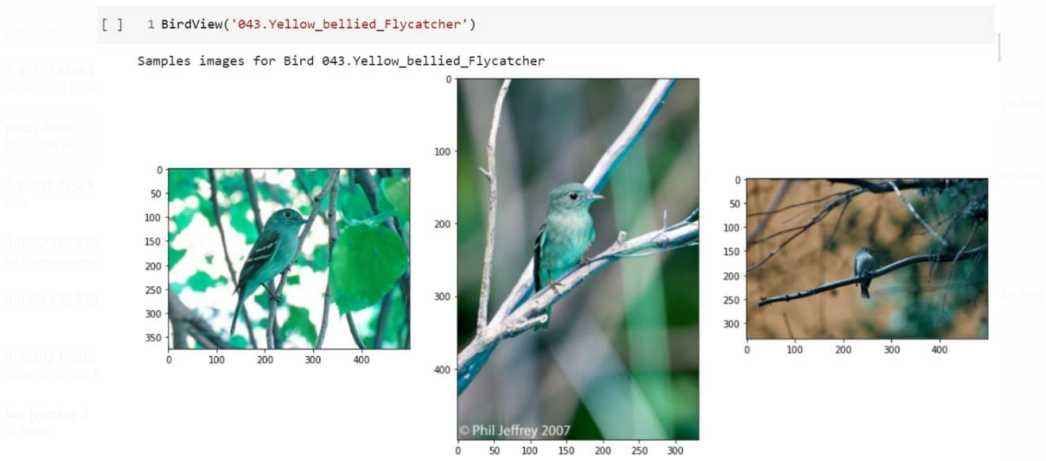
A CUDA-based GPU interface has been in progress since September 2010.

An OpenCL-based GPU interface has been in progress since October 2012, documentation for version 2.4.13.3 can be found at docs.opencv.org.

OS SUPPORT

OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo, BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake.

RESULT



CONCLUSION

- The main idea behind developing the identification software is to build awareness regarding bird-watching, bird and their identification, especially birds found in India. It also caters to need of simplifying bird identification process and thus making bird-watching easier. The technology used in the experimental setup is Transfer learning in MATLAB on a pretrained algorithm (AlexNet). It uses feature extraction for image recognition. The method used is good enough to extract features and classify images. This paper presents a summary of our project. The main purpose of the project is to identify the bird species from an image given as input by the user. The technology used is transfer learning and MATLAB. We used MATLAB because it is suitable for implementing advanced algorithm and gives good numerical precision accuracy. It is also general purpose and scientific. We achieved an accuracy of 80%-85%. We believe this project extends a great deal of scope as the purpose meets. In wildlife research and monitoring, this concept can be implemented in camera traps to maintain the record of wildlife movement in specific habitat and behavior of any species.

REFERENCES:

1. Tayal, Madhuri, Atharva Mangrulkar, Purvashree Waldey, and Chitra Dangra. 2018. “Bird Identification by Image Recognition.” Helix 8(6): 4349–4352.
2. Albustanji, Abeer. 2019. “Veiled-Face Recognition Using Deep Learning.” Mutah University. Alter, Anne L, and Karen M Wang. 2017. “An Exploration of Computer Vision Techniques for Bird Species Classification.”.
3. Atanbori, John et al. 2018. “Classification of Bird Species from Video Using Appearance and Motion Features” Ecological Informatics 48: 12– 23.
4. Brownlee, Jason. 2016. “How To Use Classification Machine Learning Algorithms in Weka.” Retrieved from https://machinelearningmastery. com/use-classification-machine-learning-algorithms-weka/.
5. Cai, J., Ee, D., Pham, B., Roe, P., & Zhang, J. (2007, December). Sensor network for the monitoring of ecosystem: Bird species recognition. In 2007 3rd international conference on intelligent sensors, sensor networks and information 293-298. IEEE.
6. Kumar, A., & Das, S. D. 2018. “Bird Species Classification Using Transfer Learning with Multistage Training”. In Workshop on Computer Vision Applications 28-38. Springer, Singapore.
7. Hassanat, A. (2018). “Furthest-pair-based binary search tree for speeding big data classification using k-nearest neighbors”. Big Data, 6(3): 225-235.