USING SUPPORT VECTOR MACHINES TO CLASSIFY STUDENT ATTENTIVENESS FOR THE DEVELOPMENT OF PERSONALIZED LEARNING SYSTEMS

[1]V. Sai Charan Rajeev Varma, [2]D. Lakshmi Narayana Raju, [3]S. Lakshman Sai Nadh, [4]Dr.AMUTHA K

[1,2,3]Students, [4]Associate Professor
Department of CSE
Bharath Institute of Higher Education and Research
Chennai 600 073, Tamilnadu, India

Abstract- **There have been many studies in which researchers have attempted to classify student attentiveness. Many of these approaches depended on a qualitative analysis and lacked any quantitative analysis. Therefore, this work is focused on bridging the gap between qualitative and quantitative approaches to classify student attentiveness. Thus, this research applies machine learning algorithms (K-means and SVM) to automatically classify students as attentive or inattentive using data from a consumer RGB-D sensor. Results of this research can be used to improve teaching strategies for instructors at all levels and can aid instructors in implementing personalized learning systems, which is a National Academy of Engineering Grand Challenge. This research applies machine learning algorithms to an educational setting. Data from these algorithms can be used by instructors to provide valuable feedback on the effectiveness of their instructional strategies and pedagogies. Instructors can use this feedback to improve their instructional strategies, and students will benefit by achieving improved learning and subject mastery. Ultimately, this will result in the students' increased ability to do work in their respective areas. Broadly, this work can help advance efforts in many areas of education and instruction. It is expected that improving instructional strategies and implementing personalized learning will help create more competent, capable, and prepared persons available for the future workforce.**

1.INTRODUCTION

Student's learning attention in an offline class is a quantitative index to measure students' learning engagement in class, which includes attention depth and attention duration. The deeper and longer students' attention is in the class, the better the learning results will be. The famous Russian educator formerly said that "learning attentiveness is the only gateway to our soul, and everything in consciousness must pass through it to enter". There is a great similarity between attentive learning and immersive learning. A high learning attention state easily leads to an immersive learning state. Studies show that the learning outcomes of immersive learning can reach five times that of ordinary learning. Learning attention is related to a learner's learning motivation and learning interests. Strong learning objectives and intensive learning interests, such as happy learning or inquiry learning, can easily drive learners to enter a high-focus learning state. Well and carefully designed teaching methods can stimulate students' learning interest in class and improve their learning attention. The characteristics of learning attention vary among students and depend on the special characteristics of each student [1,2]. One student may easily enter a high learning attention state but cannot keep it for long. Another student may find it difficult to enter the state of high attention, but once entered, they can maintain this state for a long time. Some students struggle to stay engaged in their studies and are easily affected by external factors. We believe that studying the students' whole learning attention in class can measure the Electronics 2022, 11, 2663. teaching effect of the teacher and make a quantitative index to measure the effect of class teaching. Students' learning attention can also help teachers to improve their teaching methods. Studying the personal learning attention of each student can help customize personalized learning programs for each student. The research on online learning attention has attracted researchers and achieved valuable results. In practice, the degree of students' class attention is difficult to measure. It is a challenging job to accurately detect the learning status of students in the class. Some scholars have conducted research [3,4] on how to detect learning attention and achieved valuable results. Sun et al. [5] reviewed the existing progress in learning attention at home and abroad, divided the focus research into two aspects, attention recognition based on facial expression and attention recognition based on behavior, and further discussed the development trend of attention recognition. Focus research scenarios mainly include the online education environment and offline class teaching environments. For the convenience of image and voice acquisition in online classes, researchers have studied attention recognition in online classes and made good progress. Wang et al. [6] proposed a method of building a structural equation model based on the Triadic Theory of Learning, which studies the efficiency of online learning, divides the teaching quality into deep learning and shallow learning states, and tests the online teaching quality by taking 636 students majoring in economics and management in national universities as samples. This method puts forward that learning attention heavily affected students' learning vigor and was heavily affected by teachers' teaching contents and teaching method, further optimizing the configuration of the online teaching platform to improve the efficiency of online teaching. Chang et al. [7] proposed a method that consists of 3 sub-modules. The first module is for head pose detection, which mainly detects the deviation angle of the head of each student. The second module scores fatigue by eye and mouth closure. The third module detects facial expressions and scores emotion. Then, by

merging the above information, a fuzzy comprehensive evaluation method is used to quantitatively evaluate their learning attention. The learning attention detection system of the online education platform designed by this algorithm has been tested in a simulated scene. The experiment shows that this method can effectively evaluate students' class focus and improve the class quality and students' learning outcomes. Deng et al. [8] proposed a machine learningbased approach to measuring students' learning attention. A Gabor wavelet technique is used to extract features of eye states, and a support-vector machine (SVM) model is trained to classify students' eye states. Experiments show that the methods have good performance and have value in real applications. Zhong et al. [9] proposed the analysis model of college students' class behavior based on deep learning technology and constructed an analysis system for the college students' class behavior to realize the two core functions of learning attention analysis, providing an intelligent and efficient way of College Students' behaviors analysis and supervision and supporting college students' behavior research and management. Currently, IoT technologies and deep learning methods are used in smart classrooms, continuously improving teaching methods and obtaining more accurate evaluation results [10–12]. To test the effectiveness of a machine vision-based approach, Goldberg et al. [13] proposed a new validated manual rating method and provided a method for a machine vision-based approach to evaluate students learning attention. The experiment results show that the manual rating system was significantly correlated with self-reported study engagement. Zaletelj et al. [14] proposed a novel approach to automatically estimate students' learning attention in the classroom. A Kinect One sensor is used to acquire 2D and 3D data and build a feature set of both facial and body properties of students. Machine learning algorithms are used to estimate the time-varying attention levels of each student. The experiment results show that this method can detect students' attention and average attention levels, and the system has potential practical application for non-intrusive analysis of the student learning attention. Leelavathy et al. [15] proposed a novelty method to use biometric features such as eye gaze movements, head movements and facial emotion to detect students' study Electronics 2022, 11, 2663 3 of 19 engagement with many techniques, including principal component analysis (PCA), Haar cascade, local binary patterns and OpenCV, which are used for facial emotion recognition, pupil detection, head movements recognition and machine learning. Ling et al. [16] proposed a facial expression recognition and a head pose estimation system for smart learning in the classroom by using the YOLO model to detect student faces in high-resolution video and using a self-attention-based ViT model to recognize facial expressions. The classified facial expression is used to assist the teacher in analyzing students' learning status to provide suggestions for improving the teaching effect. As the traditional offline class is still the most important teaching place, some researchers have focused on the student's attention in the traditional class. He et al. [17] proposed a learning expression automatic recognition method integrating local and global features, which extracts and integrates the local geometric features of an expression image; it uses CLBP (complete local binary pattern) global shallow texture features reduced by KPCA (kernel principal component analysis) and CNN global depth network features. In addition, a new spontaneous learning expression database is also constructed, which divides the emotions in classroom learning into five types: confusion, happiness, fatigue, surprise, and neutrality, which are used for the training of CNN model. Comparative experiments show that this method is not only better than the traditional facial expression recognition method but effectively obtains the emotional changes of middle school students in the classroom, helps teachers grasp the overall situation of class students accurately and comprehensively, and promotes the improvement of classroom teaching quality. Sun [18] proposed a method that takes students' heads up and down as a method to identify learning attention. The situation of students' head position in the classroom, whether up or down, is detected every 50 frames. Combined with the comprehensive consideration of students' grades, Sun studied the difference in students' attention and the distribution of time periods with high attention in the classroom and obtained a positive correlation between attention and grades. It was also concluded that the peak attention of each class is in the first 10 min, 21–30 min, and 5 min before the end of the class. Teachers can teach students at different levels according to the above data to improve the effectiveness of learning. Duan et al. [19] proposed a method to detect the learning attention of each student by eye-opening and closing based on raising and lowering the head. When the experimental sample was 60 people in a class, the accuracy of this algorithm reached 92%, which improves the accuracy of recognition compared with the traditional learning attention algorithm. Yin et al. [20] proposed an intelligent teacher management system that can improve students' learning efficiency and make students more focused on learning. The system can monitor the state of students through wireless sensors. When the students are not engaged in a learning state, the system can stimulate the enthusiasm of students. At the same time, the state of students and the state of the class will be recorded in the internal background system. The system managers can monitor the state of students and teachers in a timely fashion and give some guidance and feedback to students and teachers. Scholars have applied new technologies in class to improve learning effects and have obtained good results [21,22]. We believe that traditional offline class teaching has irreplaceable advantages such as face-to-face communication, a unified learning environment and an immersive learning scene. It is of great significance to detect students' learning attention in traditional offline classes. Compared with online education, we use cameras and recording equipment to collect video and audio in the traditional class. The taken video is easily disturbed by different light conditions, ambient noise and so on. We believe that the use of eye-opening and closing detection, facial expression analysis, fatigue analysis [23] and other methods needs higher quality video and audio that the traditional class cannot provide. It is difficult to collect such high-quality videos in practice, and the deep learning model also faces difficulty in analyzing such high-resolution images. Due to the limitations of data collection equipment in traditional classrooms, this paper proposes a deep learning method based on the self-attention mechanism. According to the traditional class teaching scene, the students' head pose parameters combined with the class states can identify the Electronics 2022, 11, 2663 4 of 19 students' learning attention. There are face-detecting tools in our toolbox to detect faces. There is a head pose recognition model for obtaining head pose parameters. With speech recognition tools, a class can be divided into lecture, interaction, practice, and transcription states. Finally, we use multi-modal analysis methods to analyze the rise of students in different class states to achieve a more accurate analysis of students' attention. The proposed method in this paper is based on the following assumptions. (1) When a teacher is giving a lecture or interacting with students, such as asking a question and waiting for the students to answer, we think that students should look at the teacher. If the students are not looking at the teacher, but lowering their heads or faces to the left or right, then we do not think they are focused. (2) When the class is in the practice state,

students should lower their heads and write; otherwise, we do not think they are focused. (3) When a student is in the state of taking notes, he should look up at the blackboard and then lower his head to copy. If he is not in this state, we also think he is not in the state of focus. The key contributions of the proposed method include the following. • Different from many existing studies on online student learning attention, our study aims to analyze student learning attention in a traditional offline class. The input data, such as face images, face expressions, and voices, are relatively difficult to acquire due to occlusion, noises and light conditions and other factors. • Different from the traditional questionnaire survey method, this paper uses the observation and analysis method based on artificial intelligence to carry out the research on smart education. It collects videos and sounds in class through Internet of Things (IoT) technology and uses machines instead of people to observe students. It can capture the most real scene in class and uses artificial intelligence technology to compute and analyze students' learning attention. • This paper initially divides a class into multiple time periods, each of which can be categorized into four states of lecturing, interaction, practice, and transcription and obtains the differences in students' learning attention features in the four different class states. • Since the image acquisition method in the offline class is limited, the image patch of students' faces in the video is small enough that we cannot analyze face expressions to obtain student learning attention. We use many deep learning models, such as Retinaface [24], ViT [25–27], and ASR [28–30], for face detection and location, head pose estimation [31,32], and speech recognition to accurately extract the learning attention of each student. • This paper analyzed each student's learning attention and carried out a total statistical analysis so that we can help each student improve his/her learning outcome and help teachers improve their teaching effects. • The proposed method has a high application value and can be deployed in classrooms or laboratories in high schools and universities.

2.LITERATURE REVIEW

1. Cebrián, G.; Palau, R.; Mogas, J. The Smart Classroom as a Means to the Development of ESD Methodologies. Sustainability 2020, 12, 3010. [CrossRef].

Educational institutions are envisioned as principal agents for addressing the current sustainability challenge that society is facing. Education for Sustainable Development (ESD) is transformational and concerns learning content and outcomes, pedagogy and the learning environment in itself. ESD entails rethinking the learning environment (physical and virtual) in line with sustainable development, which implies classrooms' transformation towards learner engagement, formative assessments and active methodologies. This paper responds to this need through exploring the relationship between Smart Classrooms and four widely used ESD methodologies (project or problem-based learning, case study, simulation and cooperative inquiry), identifying how the dimensions and categories of the characteristics of Smart Classrooms can contribute and lead to the implementation of ESD methodologies in real teaching practice in an effective way. The method used in this study consisted of a literature review of both theoretical frameworks separately, ESD and Smart Classrooms, and a subsequent expert analysis to identify the interrelation between both. The Smart Classroom shows a high level of adequacy for using problem and project-based learning, case study and cooperative inquiry methods because of its characteristics in terms of technology developments, environmental conditions and processes. Simulation is the ESD methodology with the lowest level of adequacy in a Smart Classroom, because it is primarily held online rather than through face-to-face teaching. Smart Education facilitates the putting in practice of ESD processes as it enables the creation of intelligent, sustainable, resource-efficient, personalised and adaptive learning environments. Further empirical research is needed to explore the influence that the Smart Classroom has in enabling ESD processes and practices, and to identify students' and teachers' needs at different education levels. Additionally, teacher training programmes focused on the correct use of Smart Classrooms and on the digital competence of teachers are critical to its successful implementation

2. Xiao, N.; Thor, D.; Zheng, M. Student Preferences Impact Outcome of Flipped Classroom in Dental Education: Students Favoring Flipped Classroom Benefited More. Educ. Sci. 2021, 11, 150. [CrossRef].

Many reports in dental education showed that student learning improved with the flipped classroom method. However, there are few reports that describe how different subsets of students may benefit from the flipped classroom. In this study, we investigated how students' preference for the flipped classroom impacted their learning outcome. We used a flipped classroom module on the physiology of the autonomic nervous system taught to year one Doctor of Dental Surgery students to test the hypothesis that students who favored the flipped classroom performed better on assessment quizzes. The module was composed of pre-class activity, out-of-class assignment, in-class discussion, and two in-class quizzes. Quiz 1 was given after students self-studied the foundational content online through the pre-class activity, and Quiz 2 was at the end of the module. Students filled out a survey to report learning experiences and preferences. Fewer students scored below 75% on Quiz 2 than on Quiz 1. Students' self-evaluated understanding of content significantly improved after finishing the assignment and discussion compared to finishing the pre-class activity alone. Moreover, students who preferred to learn through the flipped classroom scored higher in Quiz 2. Students with higher overall grades in the course preferred the flipped classroom more than low performers. Our results indicated that students favoring the flipped classroom method spent more time on the assignment, understood the content better, and performed better on assessments than students who prefer traditional lectures

3. Huang, L.; Su, J.; Pao, T. A Context Aware Smart Classroom Architecture for Smart Campuses. Appl. Sci. 2019, 9, 1837. [CrossRef].

The Smart campus is a concept of an education institute using technologies, such as information systems, internet of things (IoT), and context-aware computing, to support learning, teaching, and administrative activities. Classrooms are important building blocks of a school campus. Therefore, a feasible architecture for building and running smart classrooms is essential for a smart campus. However, most studies related to the smart classroom are focused on studying or addressing particular technical or educational issues, such as networking, AI applications, lecture quality, and user responses to technology. In this study, an architecture for building and running context-aware smart classrooms is proposed. The proposed architecture consists of three parts including a

prototype of a context-aware smart classroom, a model for technology integration, and supporting measures for the operation of smart classrooms in this architecture. The classroom prototype was designed based on our study results and a smart classroom project in Ming Chuan University (MCU). The integration model was a layered model uses Raspberry Pi in the bottom layer of the model to integrate underlying technologies and provide application interfaces to the higher layer applications for the ease of building context-aware smart classroom applications. As a result, application interfaces were implemented using Raspberry Pi based on the proposed technology integration model, and a context-aware energy-saving smart classroom application was implemented based on the proposed classroom prototype and the implemented web application interface. The result shows that, in terms of technology, the proposed architecture is feasible for building context-aware smart classrooms in smart campuses

4. , J.; Balogh, Z.; Reichel, J.; Magdin, M.; Koprda, Š.; Molnár, G. Application Experiences Using IoT Devices in Education. Appl. Sci. 2020, 10, 7286.

The Internet of Things (IoT) is becoming a regular part of our lives. The devices can be used in many sectors, such as education and in the learning process. The article describes the possibilities of using commonly available devices such as smart wristbands (watches) and eye tracking technology, i.e., using existing technical solutions and methods that rely on the application of sensors while maintaining non-invasiveness. By comparing the data from these devices, we observed how the students' attention affects their results. We looked for a correlation between eye tracking, heart rate, and student attention and how it all impacts their learning outcomes. We evaluate the obtained data in order to determine whether there is a degree of dependence between concentration and heart rate of students.

3 . SYSTEM ANALYSIS

3.1 Existing System:

- In an existing system a The learner's behaviors such as login, exit and jump between web pages in the system are trajectory behaviors.
- Trajectory behaviors are the most extensive type of behavior and the most basic type of learning behaviors. When a learner retrieves a certain type of resource, this behavior belongs to both the resource learning behavior and the trajectory behavior.
- The difference between the trajectory behavior and the resource learning behavior is that the resource learning behavior contains the body of the search, and the track behavior is only to retrieve the action, not including the main content of the retrievation.

3.2 Proposed System :

- This article describes a system that uses a commercial camera to monitor, count, and record student gestures, postures, facial expressions, and verbalizations in order to produce data for determining student attentiveness.
- Machine learning algorithms are then used to cluster, label, and classify the data for the purpose of classifying subsequent students as attentive or inattentive.
- This system is an imperative step towards developing the proposed personalized learning system described in this article.
- The investigated as an indicator of student attentiveness. Eye and head pose tracking have also been used to determine student attentiveness. Facial expressions have been used to infer student attentiveness for computer network courses.

3.3 Advantages:

- The results from the system can be used to determine the learning style of a particular student.
- An instructor can teach similar material with various teaching styles and record which particular student reacts best to a given style

4. SYSTEM SPECIFICATIONS

4.1 HARDWARE SPECIFICATION

- System - Pentium-IV
- Speed - 2.4GHZ
- Hard disk - 40GB
- Monitor - 15VGA color
- RAM - 512MB

4.2 SOFTWARE SPECIFICATION

- Operating System - Windows XP
- Coding language – Python

5. SYSTEM DESIGN

5.1 Architecture diagram

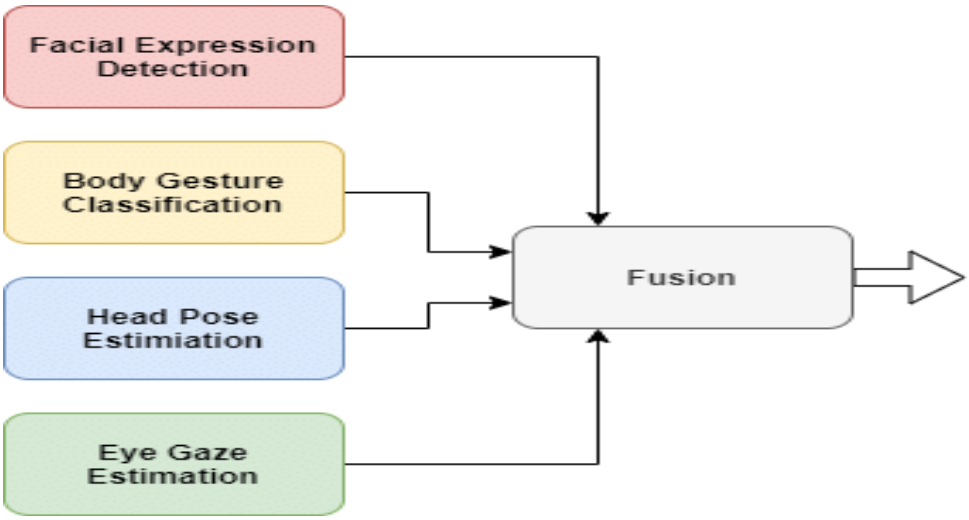


Fig 5.1 Architecture Diagram

5.2 BLOCK DIAGRAM

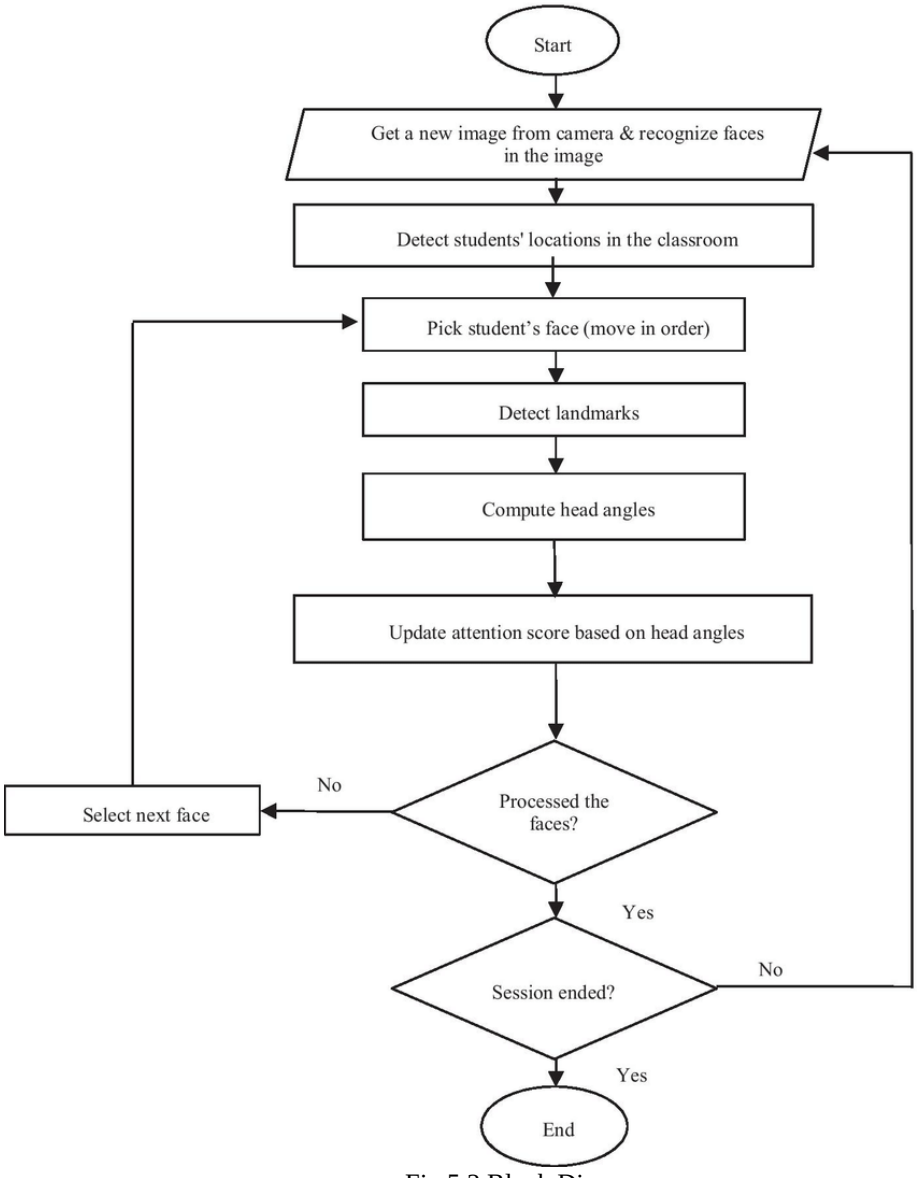


Fig 5.2 Block Diagram

6.MODULES
Modules

1. **Image acquisition**
2. **Preprocessing**
3. **Feature extraction**
4. **Segmentation**

Image acquisition

Image acquisition can be defined as the act of procuring an image from sources. This can be done by hardware system such as cameras and datasets and also some encoders sensors also take place in this process

Preprocessing

The main aim of an image pre-processing is an improvement of data such as image that reduces the unwilling distortions or enhances some features, simply we can said that remove the unwanted disturbance from the image.

Feature extraction

It is a part of the reduction process in dimensionally in which an initial set of the raw data is divided and reduced to more manageable groups.

Segmentation

It is a process of conversion of pixel into labelled image from the image. By this process you can process the important segments not an entire image.

Classification

The task of identifying what exactly in the image. That process is going to happen by the model is trained to recognize various classes. For eg: you may trained a modle to recognize the three different animals in the image.

Algorithm used

- In this we are going to use the CNN algorithm
- CNN (convolutional neural network) is a form of deep learning neural network. In a nutshell, consider CNN to be a machine learning algorithm that can take an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and distinguish one from the other. CNN operates by extracting features from videos
- Gray scale method use in image preprocessing for frames conversion. After frame conversion each frame compared with train dataset and detects the suspicious activity.

Why CNN is used for image processing

- The Convolutional Neural Network (CNN)is a subtype of Neural Networks that is mainly used for applications in image and speech recognition. Its built-in convolutional layer reduces the high dimensionality of images without losing its information. That is why CNNs are especially suited for this use case.

Where is CNN algorithm is used ?

- They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

What is CNNalgorithm ?

- A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice.

CNN algorithm flow chart

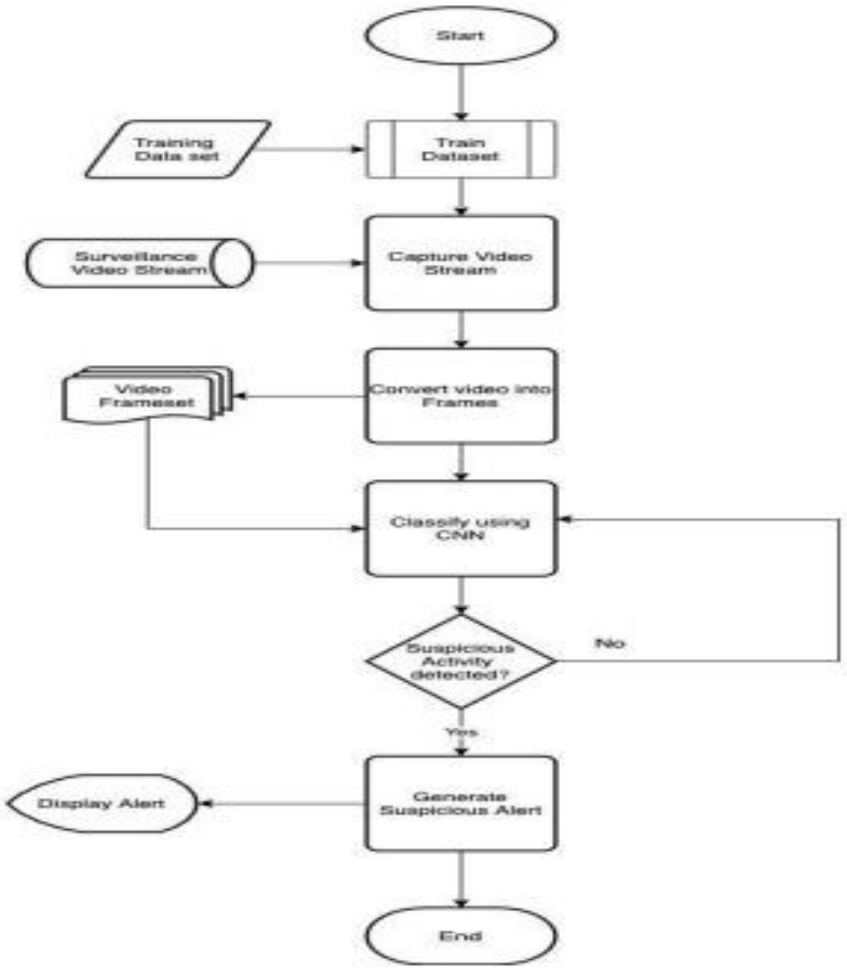


Fig 6.1 CNN Algorithm Flow Chart

7.REQUIREMENT SPECIFICATION

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!

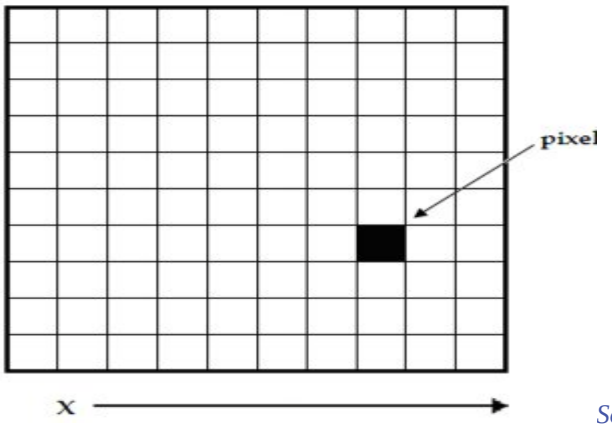
What is image processing?

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

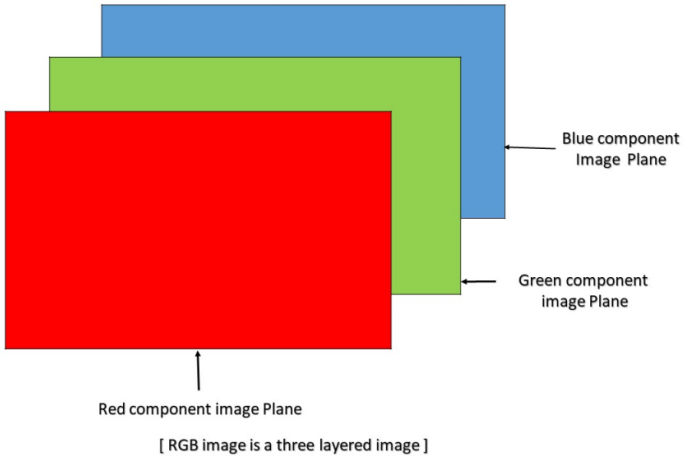
The final output can be either in the form of an image or a corresponding feature of that image. This can be used for further analysis and decision making.

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 Fig 7.1 RGB image



Source

Fig 7.2 RGB three layered image

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.

8.SYSTEM DESIGN

IMAGE PROCESSING

What is Image Processing?

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 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.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

· Importing the image with optical scanner or by digital photography.

· Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.

· Output is the last stage in which result can be altered image or report that is based on image analysis.

Purpose of Image processing

The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.
2. Image sharpening and restoration - To create a better image.
3. Image retrieval - Seek for the image of interest.
4. Measurement of pattern – Measures various objects in an image.
5. Image Recognition – Distinguish the objects in an image.

Types

The two types of **methods used for Image Processing** are **Analog and Digital** Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of

interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

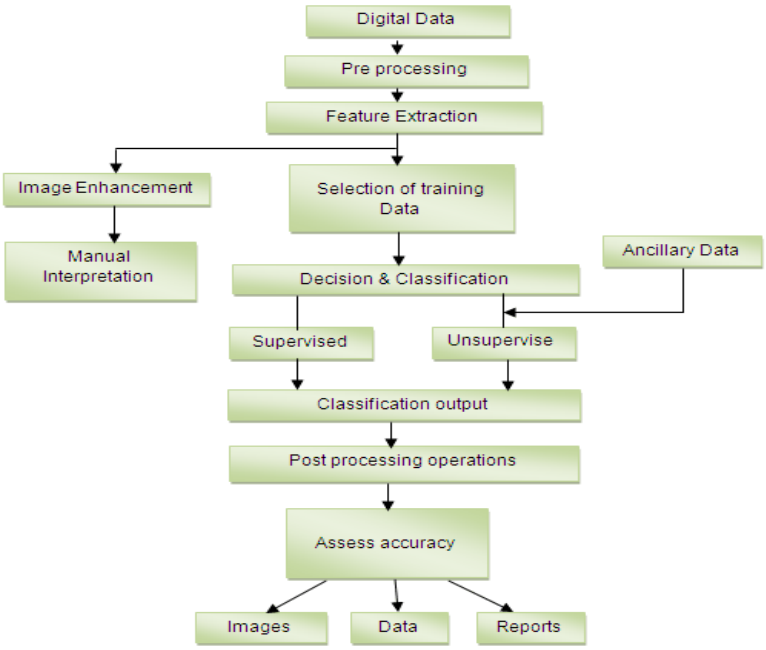


Fig 7.3 Block Diagram

IMAGE PROCESSING CONCEPTS

Binary Images

Binary images are images whose pixels have only two possible intensity values. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white.

Binary images are often produced by thresholding a grayscale or color image, in order to separate an object in the image from the background. The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background color. However, depending on the image which is to be thresholded, this polarity might be inverted, in which case the object is displayed with 0 and the background is with a non-zero value.

Some morphological operators assume a certain polarity of the binary input image so that if we process an image with inverse polarity the operator will have the opposite effect. For example, if we apply a closing operator to a black text on white background, the text will be opened.

Color Images

It is possible to construct (almost) all visible colors by combining the three primary colors red, green and blue, because the human eye has only three different color receptors, each of them sensible to one of the three colors. Different combinations in the stimulation of the receptors enable the human eye to distinguish approximately 350000 colors. A RGB color image is a multi-spectral image with one band for each color red, green and blue, thus producing a weighted combination of the three primary colors for each pixel.

A full 24-bit color image contains one 8-bit value for each color, thus being able to display $2^{24} = 16777216$ different colors. However, it is computationally expensive and often not necessary to use the full 24-bit image to store the color for each pixel. Therefore, the color for each pixel is often encoded in a single byte, resulting in an8-bit color image. The process of reducing the color representation from 24-bits to 8-bits, known as color quantization, restricts the number of possible colors to 256. However, there is normally no visible difference between a 24-color image and the same image displayed with 8 bits. An 8-bit color images are based on colormaps, which are look-up tables taking the 8-bit pixel value as index and providing an output value for each color.

8-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. If this is done, then the image is known as a 24-bit color image. However there are two problems with this approach:

- Storing 24 bits for every pixel leads to very large image files that with current technology are cumbersome to store and manipulate. For instance a 24-bit 512×512 image takes up 750KB in uncompressed form.

- Many monitor displays use colormaps with 8-bit index numbers, meaning that they can only display 256 different colors at any one time. Thus it is often wasteful to store more than 256 different colors in an image anyway, since it will not be possible to display them all on screen.

Because of this, many image formats (e.g. 8-bit GIF and TIFF) use 8-bit colormaps to restrict the maximum number of different colors to 256. Using this method, it is only necessary to store an 8-bit index into the colormap for each pixel, rather than the full 24-bit color value. Thus 8-bit image formats consist of two parts: a colormap describing what colors are present in the image, and the array of index values for each pixel in the image.

When a 24-bit full color image is turned into an 8-bit image, it is usually necessary to throw away some of the colors, a process known as color quantization. This leads to some degradation in image quality, but in practice the observable effect can be quite small, and in any case, such degradation is inevitable if the image output device (e.g. screen or printer) is only capable of displaying 256 colors or less.

The use of 8-bit images with colormaps does lead to some problems in image processing. First of all, each image has to have its own colormap, and there is usually no guarantee that each image will have exactly the same colormap. Thus on 8-bit displays it is frequently impossible to correctly display two different color images that have different colormaps at the same time. Note that in practice 8-bit images often use reduced size colormaps with less than 256 colors in order to avoid this problem.

Another problem occurs when the output image from an image processing operation contains different colors to the input image or images. This can occur very easily, as for instance when two color images are added together pixel-by-pixel. Since the output image contains different colors from the input images, it ideally needs a new colormap, different from those of the input images, and this involves further color quantization which will degrade the image quality. Hence the resulting output is usually only an approximation of the desired output. Repeated image processing operations will continually degrade the image colors. And of course we still have the problem that it is not possible to display the images simultaneously with each other on the same 8-bit display.

Because of these problems it is to be expected that as computer storage and processing power become cheaper, there will be a shift away from 8-bit images and towards full 24-bit image processing.

24-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. Image formats that store a full 24 bits to describe the color of each and every pixel are therefore known as 24-bit color images.

Using 24 bits to encode color information allows $2^{24} = 16777216$ different colors to be represented, and this is sufficient to cover the full range of human color perception fairly well.

The term 24-bit is also used to describe monitor displays that use 24 bits per pixel in their display memories, and which are hence capable of displaying a full range of colors.

There are also some disadvantages to using 24-bit images. Perhaps the main one is that it requires three times as much memory, disk space and processing time to store and manipulate 24-bit color images as compared to 8-bit color images. In addition, there is often not much point in being able to store all those different colors if the final output device (e.g. screen or printer) can only actually produce a fraction of them. Since it is possible to use colormaps to produce 8-bit color images that look almost as good, at the time of writing 24-bit displays are relatively little used. However it is to be expected that as the technology becomes

Color Quantization

Color quantization is applied when the color information of an image is to be reduced. The most common case is when a 24-bit color image is transformed into an 8-bit color image.

Two decisions have to be made:

1. which colors of the larger color set remain in the new image, and
2. how are the discarded colors mapped to the remaining ones.

The simplest way to transform a 24-bit color image into 8 bits is to assign 3 bits to red and green and 2 bits to blue (blue has only 2 bits, because of the eye's lower sensitivity to this color). This enables us to display 8 different shades of red and green and 4 of blue. However, this method can yield only poor results. For example, an image might contain different shades of blue which are all clustered around a certain value such that only one shade of blue is used in the 8-bit image and the remaining three blues are not used.

Alternatively, since 8-bit color images are displayed using a colormap, we can assign any arbitrary color to each of the 256 8-bit values and we can define a separate colormap for each image. This enables us perform a color quantization adjusted to the data contained in the image. One common approach is the popularity algorithm, which creates a histogram of all colors and retains the 256 most frequent ones. Another approach, known as the median-cut algorithm, yields even better results but also needs more computation time. This technique recursively fits a box around all colors used in the RGB colorspacewhich it splits at the median value of its longest side. The algorithm stops after 255 recursions. All colors in one box are mapped to the centroid of this box.

All above techniques restrict the number of displayed colors to 256. A technique of achieving additional colors is to apply a variation of half-toning used for gray scale images, thus increasing the color resolution at the cost of spatial resolution. The 256 values of the colormap are divided into four sections containing 64 different values of red, green, blue and white. As can be seen in Figure 1, a 2×2 pixel area is grouped together to represent one composite color, each of the four pixels displays either one of the primary colors or white. In this way, the number of possible colors is increased from 256 to 64^4.

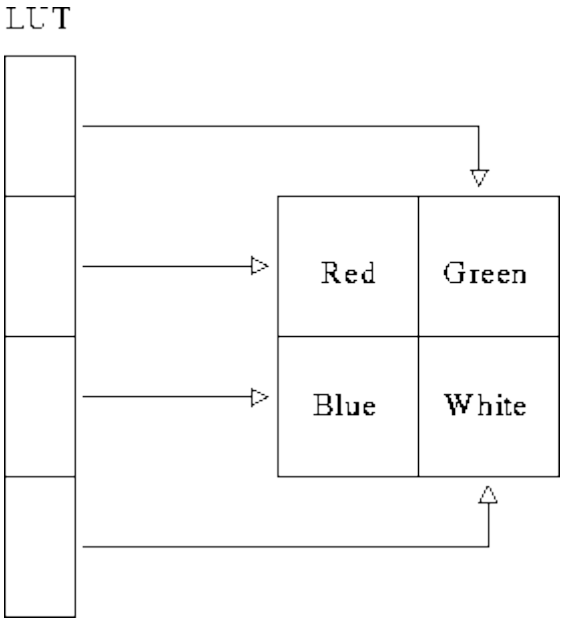


Fig 7.4 colour quantization

A 2×2-pixel area displaying one composite color

cheaper, their use in image processing will grow.

Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a gray level image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel. Figure 1 shows an example image and kernel that we will use to illustrate convolution.

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Fig 7.5 Convolution diagram

Figure 1 An example small image (left) and kernel (right) to illustrate convolution. The labels within each grid square are used to identify each square.The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. (Note that implementations differ in what they do at the edges of images, as explained below.) Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

So, in our example, the value of the bottom right pixel in the output image will be given by:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will have M - m + 1 rows, and N - n + 1 columns.

Mathematically we can write the convolution as:

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1)K(k,l)$$

where i runs from 1 to M - m + 1 and j runs from 1 to N - n + 1.

Note that many implementations of convolution produce a larger output image than this because they relax the constraint that the kernel can only be moved to positions where it fits entirely within the image. Instead, these implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel `overlaps' the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the image, it is necessary to invent input pixel values for places where the kernel extends off the end of the image. Typically pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. Therefore in general if you are using a convolution implementation that does this, it is better to clip the image to remove these spurious regions. Removing n - 1 pixels from the right hand side and m - 1 pixels from the bottom will fix things.

Convolution can be used to implement many different operators, particularly spatial filters and feature detectors. Examples include Gaussian smoothing and the Sobel edge detector.

Distance Metrics

It is often useful in image processing to be able to calculate the distance between two pixels in an image, but this is not as straightforward as it seems. The presence of the pixel grid makes several so-called distance metrics possible which often give different answers to each other for the distance between the same pair of points. We consider the three most important ones.

Euclidean Distance

This is the familiar straight line distance that most people are familiar with. If the two pixels that we are considering have coordinates (x_1, y_1) and (x_2, y_2), then the Euclidean distance is given by:

$$D_{Euclid} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

City Block Distance

Also known as the Manhattan distance. This metric assumes that in going from one pixel to the other it is only possible to travel directly along pixel grid lines. Diagonal moves are not allowed. Therefore the `city block' distance is given by:

$$D_{City} = |x_2 - x_1| + |y_2 - y_1|$$

Chessboard Distance

This metric assumes that you can make moves on the pixel grid as if you were a King making moves in chess, *i.e.* a diagonal move counts the same as a horizontal move. This means that the metric is given by:

$$D_{Chess} = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Note that the last two metrics are usually much faster to compute than the Euclidean metric and so are sometimes used where speed is critical but accuracy is not too important.

Dithering

Dithering is an image display technique that is useful for overcoming limited display resources. The word dither refers to a random or semi-random perturbation of the pixel values.

Two applications of this techniques are particularly useful:

- Low quantization display: When images are quantized to a few bits (e.g. 3) then only a limited number of graylevels are used in the display of the image. If the scene is smoothly shaded, then the image display will generate rather distinct boundaries around the edges of image regions when the original scene intensity moves from one quantization level to the next. To eliminate this effect, one dithering technique adds random noise (with a small range of values) to the input signal before quantization into the output range. This randomizes the quantization of the pixels at the original quantization boundary, and thus pixels make a more gradual transition from neighborhoods containing 100% of the first quantization level to neighborhoods containing 100% of the second quantization level.
- Limited color display: When fewer colors are able to be displayed (e.g. 256) than are present in the input image (e.g. 24 bit color), then patterns of adjacent pixels are used to simulate the appearance of the unrepresented colors.

Edge Detectors

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information.

Since edges consist of mainly high frequencies, we can, in theory, detect edges by applying a highpass frequency filter in the Fourier domain or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results.

Since edges correspond to strong illumination gradients, we can highlight them by calculating the derivatives of the image. This is illustrated for the one-dimensional case in Figure 1.

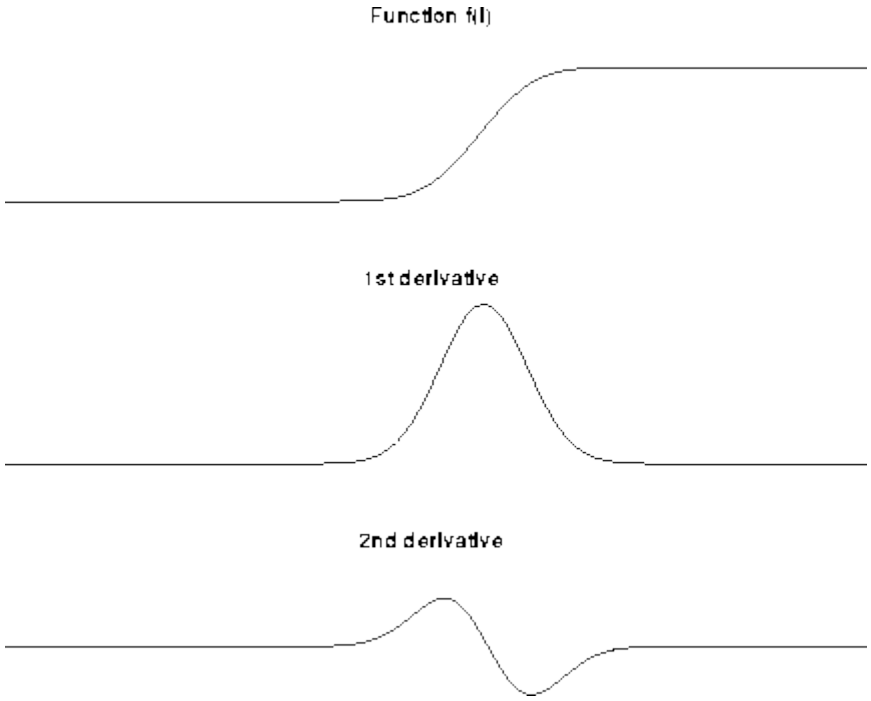


Fig 7.6 Edge Detectors Diagram

Figure 1 1st and 2nd derivative of an edge illustrated in one dimension.

We can see that the position of the edge can be estimated with the maximum of the 1st derivative or with the zero-crossing of the 2nd derivative. Therefore we want to find a technique to calculate the derivative of a two-dimensional image. For a discrete one-dimensional function f(i), the first derivative can be approximated by

$$\frac{d\,f(i)}{d(i)} = f(i+1) - f(i)$$

Calculating this formula is equivalent to convolving the function with [-1 1]. Similarly the 2nd derivative can be estimated by convolving f(i) with [1 -2 1].

Different edge detection kernels which are based on the above formula enable us to calculate either the 1st or the 2nd derivative of a two-dimensional image. There are two common approaches to estimate the 1st derivative in a two-dimensional image, Prewitt compass edge detection and gradient edge detection.

Prewitt compass edge detection involves convolving the image with a set of (usually 8) kernels, each of which is sensitive to a different edge orientation. The kernel producing the maximum response at a pixel location determines the edge magnitude and orientation. Different sets of kernels might be used: examples include Prewitt, Sobel, Kirsch and Robinson kernels.

Gradient edge detection is the second and more widely used technique. Here, the image is convolved with only two kernels, one estimating the gradient in the x-direction, Gx, the other the gradient in the y-direction, Gy. The absolute gradient magnitude is then given by

$$|G| = \sqrt{Gx^2 + Gy^2}$$

and is often approximated with

$$|G| = |Gx| + |Gy|$$

In many implementations, the gradient magnitude is the only output of a gradient edge detector, however the edge orientation might be calculated with

The most common kernels used for the gradient edge detector are the Sobel, Roberts Cross and Prewitt operators.

After having calculated the magnitude of the 1st derivative, we now have to identify those pixels corresponding to an edge. The easiest way is to threshold the gradient image, assuming that all pixels having a local gradient above the threshold must represent an edge. An alternative technique is to look for local maxima in the gradient image, thus producing one pixel wide edges. A more sophisticated technique is used by the Canny edge detector. It first applies a gradient edge detector to the image and then finds the edge pixels using non-maximal suppression and hysteresis tracking.

An operator based on the 2nd derivative of an image is the Marr edge detector, also known as zero crossing detector. Here, the 2nd derivative is calculated using a Laplacian of Gaussian (LoG) filter. The Laplacian has the advantage that it is an isotropic measure of the 2nd derivative of an image, i.e. the edge magnitude is obtained independently from the edge orientation by convolving the image with only one kernel. The edge positions are then given by the zero-crossings in the LoG image. The scale of the edges which are to be detected can be controlled by changing the variance of the Gaussian.

A general problem for edge detection is its sensitivity to noise, the reason being that calculating the derivative in the spatial domain corresponds to accentuating high frequencies and hence magnifying noise. This problem is addressed in the Canny and Marr operators by convolving the image with a smoothing operator (Gaussian) before calculating the derivative.

Frequency Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The frequency domain is a space in which each image value at image position F represents the amount that the intensity values in image I vary over a specific distance related to F. In the frequency domain, changes in image position correspond to changes in the spatial frequency, (or the rate at which image intensity values) are changing in the spatial domain image I.

For example, suppose that there is the value 20 at the point that represents the frequency 0.1 (or 1 period every 10 pixels). This means that in the corresponding spatial domain image I the intensity values vary from dark to light and back to dark over a distance of 10 pixels, and that the contrast between the lightest and darkest is 40 gray levels (2 times 20).

The spatial frequency domain is interesting because: 1) it may make explicit periodic relationships in the spatial domain, and 2) some image processing operators are more efficient or indeed only practical when applied in the frequency domain.

In most cases, the Fourier Transform is used to convert images from the spatial domain into the frequency domain and vice-versa.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

Grayscale Images

A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. In fact a `gray' color is one in which the red, green and blue components all have equal intensity in RGB space, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image.

Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. If the levels are evenly spaced then the difference between successive graylevels is significantly better than the graylevel resolving power of the human eye.

Grayscale images are very common, in part because much of today's display and image capture hardware can only support 8-bit images. In addition, grayscale images are entirely sufficient for many tasks and so there is no need to use more complicated and harder-to-process color images.

Image Editing Software

There is a huge variety of software for manipulating images in various ways. Much of this software can be grouped under the heading image processing software, and the bulk of this reference is concerned with that group.

Another very important category is what we call image editing software. This group includes painting programs, graphic art packages and so on. They are often useful in conjunction with image processing software packages, in situations where direct immediate interaction with an image is the easiest way of achieving something. For instance, if a region of an image is to be masked out for subsequent image processing, it may be easiest to create the mask using an art package by directly drawing on top of the original image. The mask used in the description of the AND operator was created this way for instance. Art packages also often allow the user to move sections of the images around and brighten or darken selected regions interactively. Few dedicated image processing packages offer the same flexibility and ease of use in this respect.

Idempotence

Some operators have the special property that applying them more than once to the same image produces no further change after the first application. Such operators are said to be idempotent. Examples include the morphological operators opening and closing.

Isotropic Operators

An isotropic operator in an image processing context is one which applies equally well in all directions in an image, with no particular sensitivity or bias towards one particular set of directions (e.g. compass directions). A typical example is the zero crossing edge detector which responds equally well to edges in any orientation. Another example is Gaussian smoothing. It should be borne in mind that although an operator might be isotropic in theory, the actual implementation of it for use on a discrete pixel grid may not be perfectly isotropic. An example of this is a Gaussian smoothing filter with very small standard deviation on a square grid.

Kernel

A kernel is a (usually) smallish matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure 1 shows a 3×3 kernel that implements a mean filter.

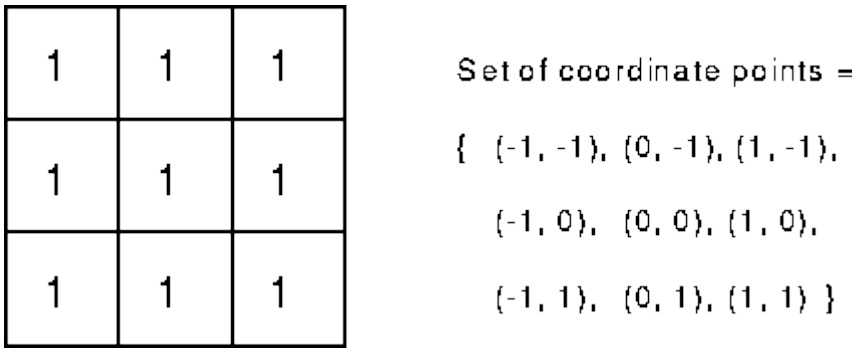


Fig 7.7 convolutional kernel

Convolution kernel for a mean filter with 3×3 neighborhood. The word `kernel' is also commonly used as a synonym for `structuring element', which is a similar object used in mathematical morphology. A structuring element differs from a kernel in that it also has a specified origin. This sense of the word `kernel' is not used in HIPR.

Logical Operators

Logical operators are generally derived from Boolean algebra, which is a mathematical way of manipulating the truth values of concepts in an abstract way without bothering about what the concepts actually mean. The truth value of a concept in Boolean value can have just one of two possible values: true or false. Boolean algebra allows you to represent things like:

The block is both red and large

by something like:

A AND *B*

where A represents `The block is red', and B represents `The block is large'. Now each of these sub-phrases has its own truth value in any given situation: each sub-phrase is either true or false. Moreover, the entire composite phrase also has a truth value: it is true if both of the sub-phrases are true, and false in any other case. We can write this AND combination rule (and its dual operation NAND) using a truth-table as shown in Figure 1, in which we conventionally represent true by 1, and false by zero.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Fig 7.8 truth tables for AND and NAND

The left hand table shows each of the possible combinations of truth values of A and B, and the the resulting truth value of A AND B. Similar truth-tables can be set up for the other logical operators: NAND, OR, NOR, XOR, XNOR and NOT.

Turning now to an image processing context, the pixel values in a binary image, which are either 0 or 1, can be interpreted as truth values as above. Using this convention we can carry out logical operations on images simply by applying the truth-table combination rules to the pixel values from a pair of input images (or a single input image in the case of NOT). Normally, corresponding pixels from each of two identically sized binary input images are compared to produce the output image, which is another binary image of the same size. As with other image arithmetic operations, it is also possible to logically combine a single input image with a constant logical value, in which case each pixel in the input image is compared to the same constant in order to produce the corresponding output pixel. See the individual logical operator descriptions for examples of these operations.

Logical operations can also be carried out on images with integer pixel values. In this extension the logical operations are normally carried out in bitwise fashion on binary representations of those integers, comparing corresponding bits with corresponding bits to produce the output pixel value. For instance, suppose that we wish to XOR the integers 47 and 255 together using 8-bit integers. 47 is 00101111 in binary and 255 is 11111111. XORing these together in bitwise fashion, we have 11010000 in binary or 208 in decimal.

Note that not all implementations of logical operators work in such bitwise fashion. For instance some will treat zero as false and any non-zero value as true and will then apply the conventional 1-bit logical functions to derive the output image. The output may be a simple binary image itself, or it may be a graylevel image formed perhaps by multiplying what would be the binary output image (containing 0's and 1's) with one of the input images.

Look-up Tables and Colormaps

Look-Up Tables or LUTs are fundamental to many aspects of image processing. An LUT is simply a table of cross-references linking index numbers to output values. The most common use is to determine the colors and intensity values with which a particular image will be displayed, and in this context the LUT is often called simply a colormap.

The idea behind the colormap is that instead of storing a definite color for each pixel in an image, for instance in 24-bit RGB format, each pixel's value is instead treated as an index number into the colormap. When the image is to be displayed or otherwise processed,

the colormap is used to look up the actual colors corresponding to each index number. Typically, the output values stored in the LUT would be RGB color values.

There are two main advantages to doing things this way. Firstly, the index number can be made to use fewer bits than the output value in order to save storage space. For instance an 8-bit index number can be used to look up a 24-bit RGB color value in the LUT. Since only the 8-bit index number needs to be stored for each pixel, such 8-bit color images take up less space than a full 24-bit image of the same size. Of course the image can only contain 256 different colors (the number of entries in an 8-bit LUT), but this is sufficient for many applications and usually the observable image degradation is small.

Secondly the use of a color table allows the user to experiment easily with different color labeling schemes for an image.

One disadvantage of using a colormap is that it introduces additional complexity into an image format. It is usually necessary for each image to carry around its own colormap, and this LUT must be continually consulted whenever the image is displayed or processed.

Another problem is that in order to convert from a full color image to (say) an 8-bit color image using a color image, it is usually necessary to throw away many of the original colors, a process known as color quantization. This process is lossy, and hence the image quality is degraded during the quantization process. Additionally, when performing further image processing on such images, it is frequently necessary to generate a new colormap for the new images, which involves further color quantization, and hence further image degradation.

As well as their use in colormaps, LUTs are often used to remap the pixel values within an image. This is the basis of many common image processing point operations such as thresholding, gamma correction and contrast stretching. The process is often referred to as anamorphosis.

Masking

A mask is a binary image consisting of zero- and non-zero values. If a mask is applied to another binary or to a grayscale image of the same size, all pixels which are zero in the mask are set to zero in the output image. All others remain unchanged.

Masking can be implemented either using pixel multiplication or logical AND, the latter in general being faster.

Masking is often used to restrict a point or arithmetic operator to an area defined by the mask. We can, for example, accomplish this by first masking the desired area in the input image and processing it with the operator, then masking the original input image with the inverted mask to obtain the unprocessed area of the image and finally recombining the two partial images using image addition. An example can be seen in the worksheet on the logical AND operator. In some image processing packages, a mask can directly be defined as an optional input to a point operator, so that automatically the operator is only applied to the pixels defined by the mask .

Mathematical Morphology

The field of mathematical morphology contributes a wide range of operators to image processing, all based around a few simple mathematical concepts from set theory. The operators are particularly useful for the analysis of binary images and common usages include edge detection, noise removal, image enhancement and image segmentation.

The two most basic operations in mathematical morphology are erosion and dilation. Both of these operators take two pieces of data as input: an image to be eroded or dilated, and a structuring element (also known as a kernel). The two pieces of input data are each treated as representing sets of coordinates in a way that is slightly different for binary and grayscale images.

For a binary image, white pixels are normally taken to represent foreground regions, while black pixels denote background. (Note that in some implementations this convention is reversed, and so it is very important to set up input images with the correct polarity for the implementation being used). Then the set of coordinates corresponding to that image is simply the set of two-dimensional Euclidean coordinates of all the foreground pixels in the image, with an origin normally taken in one of the corners so that all coordinates have positive elements.

For a grayscale image, the intensity value is taken to represent height above a base plane, so that the grayscale image represents a surface in three-dimensional Euclidean space. Figure 1 shows such a surface. Then the set of coordinates associated with this image surface is simply the set of three-dimensional Euclidean coordinates of all the points within this surface and also all points below the surface, down to the base plane. Note that even when we are only considering points with integer coordinates, this is a lot of points, so usually algorithms are employed that do not need to consider all the points.

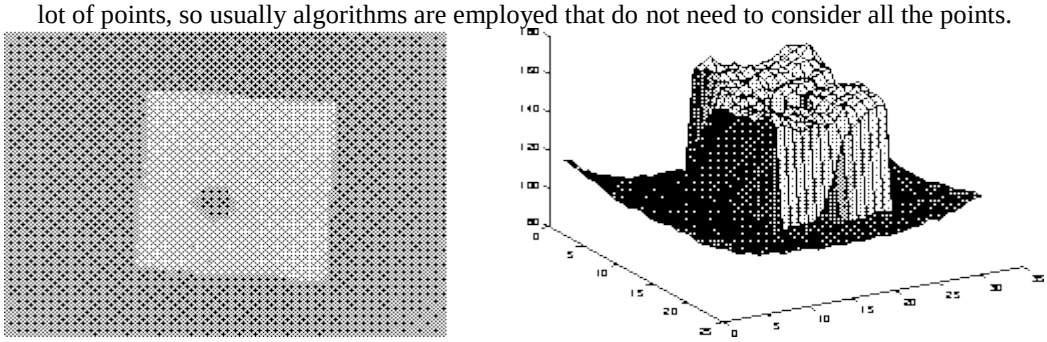


Fig 7.9 Gray level image

Simple graylevel image and the corresponding surface in image space The structuring element is already just a set of point coordinates (although it is often represented as a binary image). It differs from the input image coordinate set in that it is normally much smaller, and its coordinate origin is often not in a corner, so that some coordinate elements will have negative values. Note that in many implementations of morphological operators, the structuring element is assumed to be a particular shape (e.g. a 3×3 square) and so is hardwired into the algorithm.

Binary morphology can be seen as a special case of graylevel morphology in which the input image has only two graylevels at values 0 and 1.

Erosion and dilation work (at least conceptually) by translating the structuring element to various points in the input image, and examining the intersection between the translated kernel coordinates and the input image coordinates. For instance, in the case of erosion, the output coordinate set consists of just those points to which the origin of the structuring element can be translated, while the element still remains entirely `within' the input image.

Virtually all other mathematical morphology operators can be defined in terms of combinations of erosion and dilation along with set operators such as intersection and union. Some of the more important are opening, closing and skeletonization.

Multi-spectral Images

A multi-spectral image is a collection of several monochrome images of the same scene, each of them taken with a different sensor. Each image is referred to as a band. A well known multi-spectral (or multi-band image) is a RGB color image, consisting of a red, a green and a blue image, each of them taken with a sensor sensitive to a different wavelength. In image processing, multi-spectral images are most commonly used for Remote Sensing applications. Satellites usually take several images from frequency bands in the visual and non-visual range. Landsat 5, for example, produces 7 band images with the wavelength of the bands being between 450 and 1250 nm.

All the standard single-band image processing operators can also be applied to multi-spectral images by processing each band separately. For example, a multi-spectral image can be edge detected by finding the edges in each band and than ORing the three edge images together. However, we would obtain more reliable edges, if we associate a pixel with an edge based on its properties in all three bands and not only in one.

To fully exploit the additional information which is contained in the multiple bands, we should consider the images as one multi-spectral image rather than as a set of monochrome graylevel images. For an image with k bands, we can then describe the brightness of each pixel as a point in a k-dimensional space represented by a vector of length k.

Special techniques exist to process multi-spectral images. For example, to classify a pixel as belonging to one particular region, its intensities in the different bands are said to form a feature vector describing its location in the k-dimensional feature space. The simplest way to define a class is to choose a upper and lower threshold for each band, thus producing a k-dimensional `hyper-cube' in the feature space. Only if the feature vector of a pixel points to a location within this cube, is the pixel classified as belonging to this class. A more sophisticated classification method is described in the corresponding worksheet.

The disadvantage of multi-spectral images is that, since we have to process additional data, the required computation time and memory increase significantly. However, since the speed of the hardware will increase and the costs for memory will decrease in the future, it can be expected that multi-spectral images will become more important in many fields of computer vision.

Non-linear Filtering

Suppose that an image processing operator F acting on two input images A and B produces output images C and D respectively. If the operator F is linear, then

$$F(a \times \mathrm{A} + b \times \mathrm{B}) = a \times \mathrm{C} + b \times \mathrm{D}$$

where a and b are constants. In practice, this means that each pixel in the output of a linear operator is the weighted sum of a set of pixels in the input image.

By contrast, non-linear operators are all the other operators. For example, the threshold operator is non-linear, because individually, corresponding pixels in the two images A and B may be below the threshold, whereas the pixel obtained by adding A and B may be above threshold. Similarly, an absolute value operation is non-linear:

$$|-1+1| \neq |-1| + |1|$$

as is the exponential operator:

$$exp(1+1) \neq exp(1) + exp(1)$$

Pixels

In order for any digital computer processing to be carried out on an image, it must first be stored within the computer in a suitable form that can be manipulated by a computer program. The most practical way of doing this is to divide the image up into a collection of discrete (and usually small) cells, which are known as pixels. Most commonly, the image is divided up into a rectangular grid of pixels, so that each pixel is itself a small rectangle. Once this has been done, each pixel is given a pixel value that represents the color of that pixel. It is assumed that the whole pixel is the same color, and so any color variation that did exist within the area of the pixel before the image was discretized is lost. However, if the area of each pixel is very small, then the discrete nature of the image is often not visible to the human eye.

Other pixel shapes and formations can be used, most notably the hexagonal grid, in which each pixel is a small hexagon. This has some advantages in image processing, including the fact that pixel connectivityis less ambiguously defined than with a square grid, but hexagonal grids are not widely used. Part of the reason is that many image capture systems (e.g. most CCD cameras and scanners) intrinsically discretize the captured image into a rectangular grid in the first instance.

Pixel Connectivity

The notation of pixel connectivity describes a relation between two or more pixels. For two pixels to be connected they have to fulfill certain conditions on the pixel brightness and spatial adjacency.

First, in order for two pixels to be considered connected, their pixel values must both be from the same set of values V. For a grayscale image, V might be any range of graylevels, e.g. V={22,23,...40}, for a binary image we simple have V={1}.

To formulate the adjacency criterion for connectivity, we first introduce the notation of neighborhood. For a pixel p with the coordinates (x,y) the set of pixels given by:

$$N_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

is called its 4-neighbors. Its 8-neighbors are defined as

$$N_8(p) = N_4 \cup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$

From this we can infer the definition for 4- and 8-connectivity:

Two pixels p and q, both having values from a set V are 4-connected if q is from the set $N_4(p)$ and 8-connected if q is from $N_8(p)$.

General connectivity can either be based on 4- or 8-connectivity; for the following discussion we use 4-connectivity.

A pixel p is connected to a pixel q if p is 4-connected to q or if p is 4-connected to a third pixel which itself is connected to q. Or, in other words, two pixels q and p are connected if there is a path from p and q on which each pixel is 4-connected to the next one.

A set of pixels in an image which are all connected to each other is called a connected component. Finding all connected components in an image and marking each of them with a distinctive label is called connected component labeling.

An example of a binary image with two connected components which are based on 4-connectivity can be seen in Figure 1. If the connectivity were based on 8-neighbors, the two connected components would merge into one.

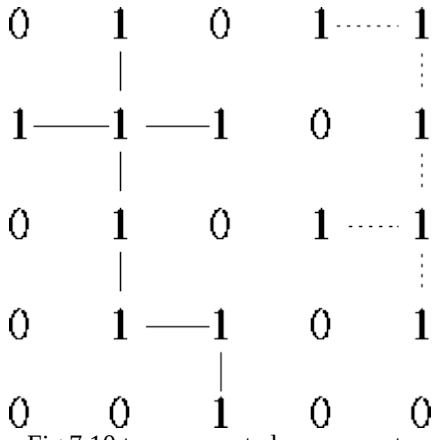


Fig 7.10 two connected components

Pixel Values

Each of the pixels that represents an image stored inside a computer has a pixel value which describes how bright that pixel is, and/or what color it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

To represent color images, separate red, green and blue components must be specified for each pixel (assuming an RGB colorspace), and so the pixel `value' is actually a vector of three numbers. Often the three different components are stored as three separate `grayscale' images known as color planes (one for each of red, green and blue), which have to be recombined when displaying or processing.

Multi-spectral images can contain even more than three components for each pixel, and by extension these are stored in the same kind of way, as a vector pixel value, or as separate color planes.

The actual grayscale or color component intensities for each pixel may not actually be stored explicitly. Often, all that is stored for each pixel is an index into a colormap in which the actual intensity or colors can be looked up.

Although simple 8-bit integers or vectors of 8-bit integers are the most common sorts of pixel values used, some image formats support different types of value, for instance 32-bit signed integers or floating point values. Such values are extremely useful in image processing as they allow processing to be carried out on the image where the resulting pixel values are not necessarily 8-bit integers. If this approach is used then it is usually necessary to set up a colormap which relates particular ranges of pixel values to particular displayed colors.

Primary Colors

It is a useful fact that the huge variety of colors that can be perceived by humans can all be produced simply by adding together appropriate amounts of red, blue and green colors. These colors are known as the primary colors. Thus in most image processing applications, colors are represented by specifying separate intensity values for red, green and blue components. This representation is commonly referred to as RGB.

The primary color phenomenon results from the fact that humans have three different sorts of color receptors in their retinas which are each most sensitive to different visible light wavelengths.

The primary colors used in painting (red, yellow and blue) are different. When paints are mixed, the `addition' of a new color paint actually subtracts wavelengths from the reflected visible light.

RGB and Colorspaces

A color perceived by the human eye can be defined by a linear combination of the three primary colors red, green and blue. These three colors form the basis for the RGB-colorspace. Hence, each perceivable color can be defined by a vector in the three-dimensional colorspace. The intensity is given by the length of the vector, and the actual color by the two angles describing the orientation of the vector in the colorspace.

The RGB-space can also be transformed into other coordinate systems, which might be more useful for some applications. One common basis for the color space is IHS. In this coordinate system, a color is described by its intensity, hue (average wavelength) and saturation (the amount of white in the color). This color space makes it easier to directly derive the intensity and color of perceived light and is therefore more likely to be used by human beings.

Spatial Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The spatial domain is the normal image space, in which a change in position in I directly projects to a change in position in S. Distances in I (in pixels) correspond to real distances (e.g. in meters) in S.

This concept is used most often when discussing the frequency with which image values change, that is, over how many pixels does a cycle of periodically repeating intensity variations occur. One would refer to the number of pixels over which a pattern repeats (its periodicity) in the spatial domain.

In most cases, the Fourier Transform will be used to convert images from the spatial domain into the frequency domain.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

Another term used in this context is spatial derivative, which refers to how much the image intensity values change per change in image position.

Structuring Elements

The field of mathematical morphology provides a number of important image processing operations, including erosion, dilation, opening and closing. All these morphological operators take two pieces of data as input. One is the input image, which may be either binary or grayscale for most of the operators. The other is the structuring element. It is this that determines the precise details of the effect of the operator on the image.

The structuring element is sometimes called the kernel, but we reserve that term for the similar objects used in convolutions.

The structuring element consists of a pattern specified as the coordinates of a number of discrete points relative to some origin. Normally cartesian coordinates are used and so a convenient way of representing the element is as a small image on a rectangular grid. Figure 1 shows a number of different structuring elements of various sizes. In each case the origin is marked by a ring around that point. The origin does not have to be in the center of the structuring element, but often it is. As suggested by the figure, structuring elements that fit into a 3×3 grid with its origin at the center are the most commonly seen type.

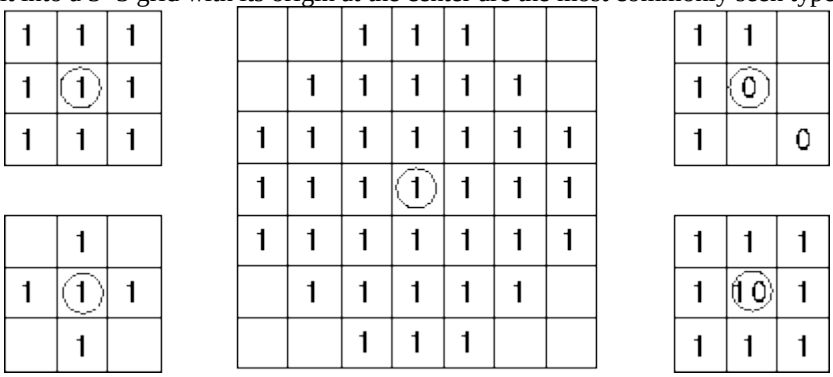


Fig 7.11 Structural elements

Note that each point in the structuring element may have a value. In the simplest structuring elements used with binary images for operations such as erosion, the elements only have one value, conveniently represented as a one. More complicated elements, such as those used with thinning or grayscale morphological operations, may have other pixel values.

An important point to note is that although a rectangular grid is used to represent the structuring element, not every point in that grid is part of the structuring element in general. Hence the elements shown in Figure 1 contain some blanks. In many texts, these blanks are represented as zeros, but this can be confusing and so we avoid it here.

When a morphological operation is carried out, the origin of the structuring element is typically translated to each pixel position in the image in turn, and then the points within the translated structuring element are compared with the underlying image pixel values. The details of this comparison, and the effect of the outcome depend on which morphological operator is being used.

Wrapping and Saturation

If an image is represented in a byte or integer pixel format, the maximum pixel value is limited by the number of bits used for the representation, e.g. the pixel values of a 8-bit image are limited to 255.

However, many image processing operations produce output values which are likely to exceed the given maximum value. In such cases, we have to decide how to handle this pixel overflow.

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) - Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 3/19

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. Syntax and semantics 6/13/2017 Python (programming language) - Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 4/19 Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, often using English keywords where other languages use punctuation. Python

doesn't have semicolons and curly brackets "{}" which is different compared to most of the programming language. Further, Python has fewer syntactic exceptions and special cases than C or Pascal. Indentation Python uses whitespace indentation to delimit blocks – rather than curly braces or keywords. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. This feature is also sometimes termed the off-side rule. Statements and control flow Python's statements include (among others): The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it.

However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing. The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if). The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block. The while statement, which executes a block of code as long as its condition is true. The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits. The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming. The def statement, which defines a function or method. The with statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior.

The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block. The assert statement, used during debugging to check for conditions that ought to apply. The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines. The import statement, which is used to import modules whose functions or variables can be used in the current program. The print statement was changed to the print() function in Python 3. Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out 6/13/2017 Python (programming language) – Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 5/19 of the generator.

As of Python 2.5, it is possible to pass information back into a generator function, and as of Python 3.3, the information can be passed through multiple stack levels. Expressions Some Python expressions are similar to languages such as C and Java, while some are not: Addition, subtraction, and multiplication are the same, but the behavior of division differs. Python also added the ** operator for exponentiation.

As of Python 3.5, it supports matrix multiplication directly with the @ operator, versus C and Java, which implement these as library functions. Earlier versions of Python also used methods instead of an infix operator. In Python, == compares by value, versus Java, which compares numerics by value and objects by reference.
(Value comparisons in Java on objects can be performed with the equals()method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c. Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C. Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression. Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.

Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages). Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The parentheses around the tuple are optional in some contexts. Tuples can appear on the left side of an equal sign; hence a statement like x, y = y, x can be used to swap two variables. Python has a "string format" operator %. This functions analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python has various kinds of string literals: Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl- influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation (done as "$spam" in Unix shells and Perl-influenced languages) became available in Python 3.6 as "formatted string literals". Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby. Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@- quoting" in C#. Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes

are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. 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. For Internet-facing applications, many standard formats and protocols (such as MIME and HTTP) are supported. Modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and doing unit testing are also included. Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of

the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations. As of May, 2017, the Python Package Index, the official repository containing third-party software for Python, contains over 107,000 packages offering a wide range of functionality, including: graphical user interfaces, web frameworks, multimedia,databases, networking and communications test frameworks, automation and web scraping, documentation tools, system administration 6/13/2017 Python (programming language) – Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 9/19 scientific computing, text processing, image processing Development environments Most Python implementations (including CPython) include a read–eval–print loop (REPL), meaning they can function as a command line interpreter, for which the user enters statements sequentially and receives the results immediately.

Other shells add abilities beyond those in the basic interpreter, including IDLE and IPython. While generally following the visual style of the Python shell, they implement features like auto-completion, session state retention, and syntax highlighting. In addition to standard desktop integrated development environments (Python IDEs), there are also web browser-based IDEs, SageMath (intended for developing science and math-related Python programs), and a browser-based IDE and hosting environment, PythonAnywhere. Additionally, the Canopy IDE is also an option for writing Python programs. Implementations Reference implementation The main Python implementation, named CPython, is written in C meeting the C89 standard. It compiles Python programs into intermediate bytecode, which is executed by the virtual machine. CPython is distributed with a large standard library written in a mixture of C and Python. It is available in versions for many platforms, including Windows and most modern Unix-like systems. CPython was intended from almost its very conception to be cross- platform. Other implementations PyPy is a fast, compliant interpreter of Python 2.7 and 3.5. Its just-in-time compiler brings a significant speed improvement over CPython. A version taking advantage of multi-core processors using software transactional memory is being created. Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version. MicroPython is a Python 3 variant that is optimised to run on microcontrollers. No longer supported implementations Other just-in-time compilers have been developed in the past, but are now unsupported: Google began a project named Unladen Swallow in 2009 with the aim of speeding up the Python interpreter fivefold by using the LLVM, and of improving its multithreading ability to scale to thousands of cores. syco is a just-in-time specialising compiler that integrates with CPython and transforms bytecode to machine code at runtime. The emitted code is specialised for certain data types and is faster than standard Python code. In 2005, Nokia released a Python interpreter for the Series 60 mobile phones named PyS60. It includes many of the modules from the CPython implementations and some additional modules to integrate with the Symbian operating system. This project has been kept up to date to run on all variants of the S60 platform and there are 6/13/2017 Python (programming language) – Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 10/19 several third party modules available.

The Nokia N900 also supports Python with GTK widget libraries, with the feature that programs can be both written and run on the target device. [96] Cross- compilers to other languages There are several compilers to high-level object languages, with either unrestricted Python, a restricted subset of Python, or a language similar to Python as the source language: Jython compiles into Java byte code, which can then be executed by every Java virtual machine implementation.

This also enables the use of Java class library functions from the Python program.IronPython follows a similar approach in order to run Python programs on the .NET Common Language Runtime. The RPython language can be compiled to C, Java bytecode, or Common Intermediate Language, and is used to build the PyPy interpreter of Python. Pyjamas compiles Python to JavaScript. Cython compiles Python to C and C++. Pythran compiles Python to C++. Somewhat dated Pyrex (latest release in 2010) and Shed Skin (latest release in 2013) compile to C and C++ respectively. Google's Grumpy compiles Python to Go. Performance A performance comparison of various Python implementations on a non-numerical (combinatorial) workload was presented at EuroSciPy '13. Development Python's development is conducted largely through the Python Enhancement Proposal (PEP) process. The PEP process is the primary mechanism for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python.

Outstanding PEPs are reviewed and commented upon by the Python community and by Van Rossum, the Python project's benevolent dictator for life. [98]

Enhancement of the language goes along with development of the CPython reference implementation. The mailing list python-dev is the primary forum for discussion about the language's development; specific issues are discussed in the Roundup bug tracker maintained at python.org. Development took place on a selfhosted source code repository running Mercurial, until Python moved to GitHub in January 2017. CPython's public releases come in three types, distinguished by which part of the version number is incremented: Backwards- incompatible versions, where code is expected to break and must be manually ported. The first part of the version number is incremented. These releases happen infrequently—for example, version 3.0 was released 8 years after 2.0. Major or "feature" releases, which are largely compatible but introduce new features. The second part of the version number is incremented. These releases are scheduled to occur roughly every 18 months, and each major version is supported by bugfixes for several years after its release. Bugfix releases, which introduce no new features but fix bugs. The third and final part of the version number is incremented. These releases are made whenever a sufficient number of bugs have been fixed upstream since the last release, or roughly every 3 months. Security vulnerabilities are also patched in bugfixreleases.6/13/2017 Python (programming language)-Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 11/19 Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, this is often pushed back if the code is not ready.

The development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system. The community of Python developers has also contributed over 86,000 software modules (as of 20 August 2016) to the Python Package Index (PyPI), the official repository of third-party libraries for Python. The major academic conference on Python is named PyCon. There are special mentoring programmes like the Pyladies. Naming Python's name is

derived from the television series Monty Python's Flying Circus, and it is common to use Monty Python references in example code. For example, the metasyntactic variables often used in Python literature are spam and eggs, instead of the traditional foo and bar. Also, the official Python documentation and many code examples often contain various obscure Monty Python references. The prefix Py- is used to show that something is related to Python.

Examples of the use of this prefix in names of Python applications or libraries include Pygame, a binding of SDL to Python (commonly used to create games); PyS60, an implementation for the Symbian S60 operating system; PyQt and PyGTK, which bind Qt and GTK to Python respectively; and PyPy, a Python implementation originally written in Python. Uses Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of March 2017, it is the fifth most popular language. It was ranked as Programming Language of the Year for the year 2007 and 2010. It is the third most popular language whose grammatical syntax is not predominantly based on C, e.g. C++, Objective-C (note, C# and Java only have partial syntactic similarity to C, such as the use of curly braces, and are closer in similarity to each other than C). An empirical study found scripting languages (such as Python) more productive than conventional languages (such as C and Java) for a programming problem involving string manipulation and search in a dictionary. Memory consumption was often "better than Java and not much worse than C or C++". Large organizations that make use of Python include Wikipedia, Google, Yahoo!, CERN, NASA, and some smaller entities like ILM, and ITA.

The social news networking site Reddit is written entirely in Python. Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjamas and IronPython can be used to develop the client-side of Ajax-based applications.

SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox. Libraries like NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, [120][121] with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a "notebook" programmable in Python: its library covers many aspects of 6/13/2017 Python (programming language) - Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 12/19 mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus. The Python language re-implemented in Java platform is used for numeric and statistical calculations with 2D/3D visualization by the DMelt project. Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro, and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS.It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.

Python is also used in algorithmic trading and quantitative finance. Python can also be implemented in APIs of online brokerages that run on other languages by using wrappers. Python has been used in artificial intelligence tasks. As a scripting language with module architecture, simple syntax and rich text processing tools, Python is often used for natural language processing tasks. Many operating systems include Python as a standard component; the language ships with most Linux distributions, AmigaOS 4, FreeBSD, NetBSD, OpenBSD and macOS, and can be used from the terminal. Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage. Python has also seen extensive use in the information security industry, including in exploit development. Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python. The Raspberry Pi single-board computer project has adopted Python as its main user- programming language. LibreOffice includes Python and intends to replace Java with Python. Python Scripting Provider is a core feature since Version 4.0 from 7 February 2013. Languages influenced by Python Python's design and philosophy have influenced several programming languages, including: Boo uses indentation, a similar syntax, and a similar object model. However, Boo uses static typing (and optional duck typing) and is closely integrated with the .NET Framework. Cobra uses indentation and a similar syntax. Cobra's "Acknowledgements" document lists Python first among languages that influenced it. However, Cobra directly supports design-by- contract, unit tests, and optional static typing. ECMAScript borrowed iterators, generators, and list comprehensions from Python. Go is described as incorporating the "development speed of working in a dynamic language like Python". Groovy was motivated by the desire to bring the Python design philosophy to Java. Julia was designed "with true macros [.. and to be] as usable for general programming as Python [and] should be as fast as C". Calling to or from Julia is possible; to with PyCall.jl (https://github.com/steve ngj/PyCall.jl) and a Python package pyjulia (https://github.com/JuliaLang/pyjulia) allows calling, in the other direction, from Python. 6/13/2017 Python (programming language) – Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language) 13/19 OCaml has an optional syntax, named twt (The Whitespace Thing), inspired by Python and Haskell. Ruby's creator, Yukihiro Matsumoto, has said: "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python.
That's why I decided to design my own language." Coffee Script is a programming language that cross-compiles to JavaScript; it has Python-inspired syntax. Swift is a programming language invented by Apple; it has some Python-inspired syntax. Python's development practices have also been emulated by other languages. The practice of requiring a document describing the rationale for, and issues surrounding, a change to the language (in Python's case, a PEP) is also used in Tcl and Erlang because of Python's influence. Python has been awarded a TIOBE Programming Language of the Year award twice (in 2007

and 2010), which is given to the language with the greatest growth in popularity over the course of a year, as measured by the TIOBE index.

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 and is now maintained by Itseez. The library is cross-platform and free for use under the open-source BSD license 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. In mid-2008, OpenCV obtained corporate support from Willow Garage, and is now

9. SOURCE CODE

```
#Importing OpenCV Library for basic image processing functions
import cv2
# Numpy for array related functions
import numpy as np
# Dlib for deep learning based Modules and face landmark detection
import dlib
#face_utils for basic operations of conversion
from imutils import face_utils


#Initializing the camera and taking the instance
cap = cv2.VideoCapture(0)

#Initializing the face detector and landmark detector
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

#status marking for current state
sleep = 0
drowsy = 0
active = 0
status=""
color=(0,0,0)

def compute(ptA,ptB):
        dist = np.linalg.norm(ptA - ptB)
        return dist

def blinked(a,b,c,d,e,f):
        up = compute(b,d) + compute(c,e)
        down = compute(a,f)
        ratio = up/(2.0*down)

        #Checking if it is blinked
        if(ratio>0.25):
                return 2
        elif(ratio>0.21 and ratio<=0.25):
                return 1
        else:
                return 0


while True:
   _, frame = cap.read()
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    faces = detector(gray)
    #detected face in faces array
    for face in faces:
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()

        face_frame = frame.copy()
        cv2.rectangle(face_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        landmarks = predictor(gray, face)
        landmarks = face_utils.shape_to_np(landmarks)

        #The numbers are actually the landmarks which will show eye
        left_blink = blinked(landmarks[36],landmarks[37],
            landmarks[38], landmarks[41], landmarks[40], landmarks[39])
        right_blink = blinked(landmarks[42],landmarks[43],
            landmarks[44], landmarks[47], landmarks[46], landmarks[45])

        #Now judge what to do for the eye blinks
        if(left_blink==0 or right_blink==0):
            sleep+=1
            drowsy=0
            active=0
            if(sleep>6):
                    status="SLEEPING !!!"
                    color = (255,0,0)

        elif(left_blink==1 or right_blink==1):
            sleep=0
            active=0
            drowsy+=1
            if(drowsy>6):
                    status="Drowsy !"
                    color = (0,0,255)

        else:
            drowsy=0
            sleep=0
            active+=1
            if(active>6):
                    status="Active :)"
                    color = (0,255,0)

        cv2.putText(frame, status, (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1.2, color,3)

        for n in range(0, 68):
            (x,y) = landmarks[n]
            cv2.circle(face_frame, (x, y), 1, (255, 255, 255), -1)

    cv2.imshow("Frame", frame)
    cv2.imshow("Result of detector", face_frame)
    key = cv2.waitKey(1)
    if key == 27:
            break
```

9.SNAP SHOTS

OUTPUT

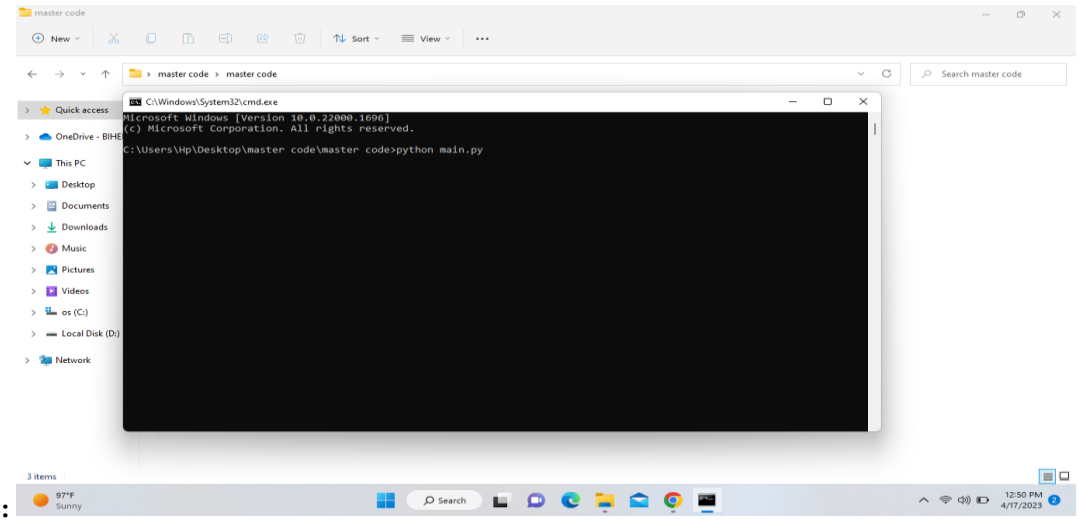


Fig 9.1 output1

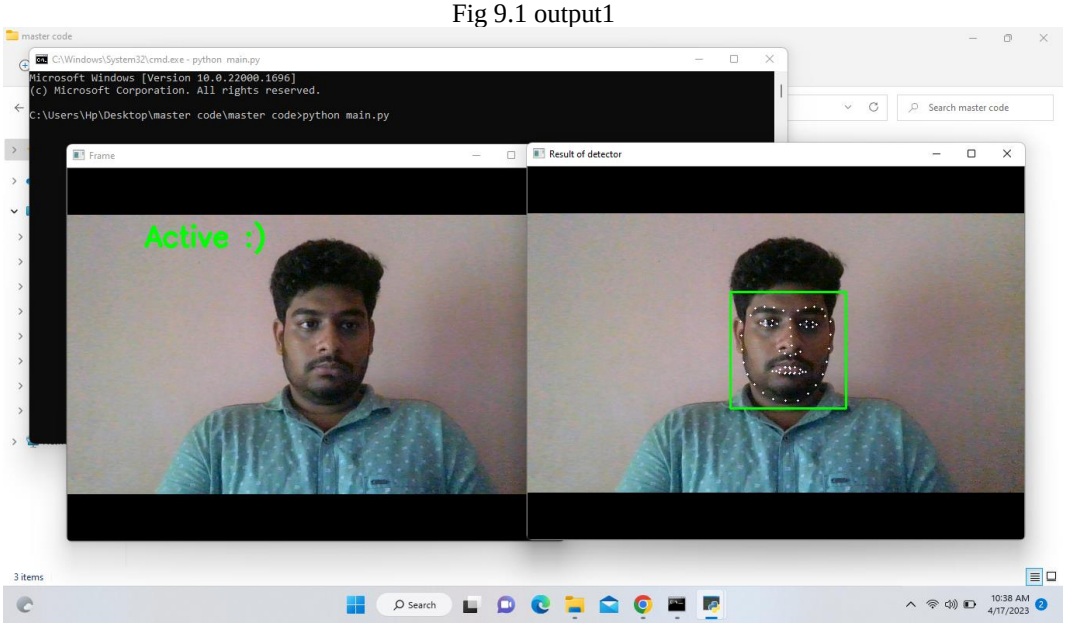


Fig 9.2 Output 2

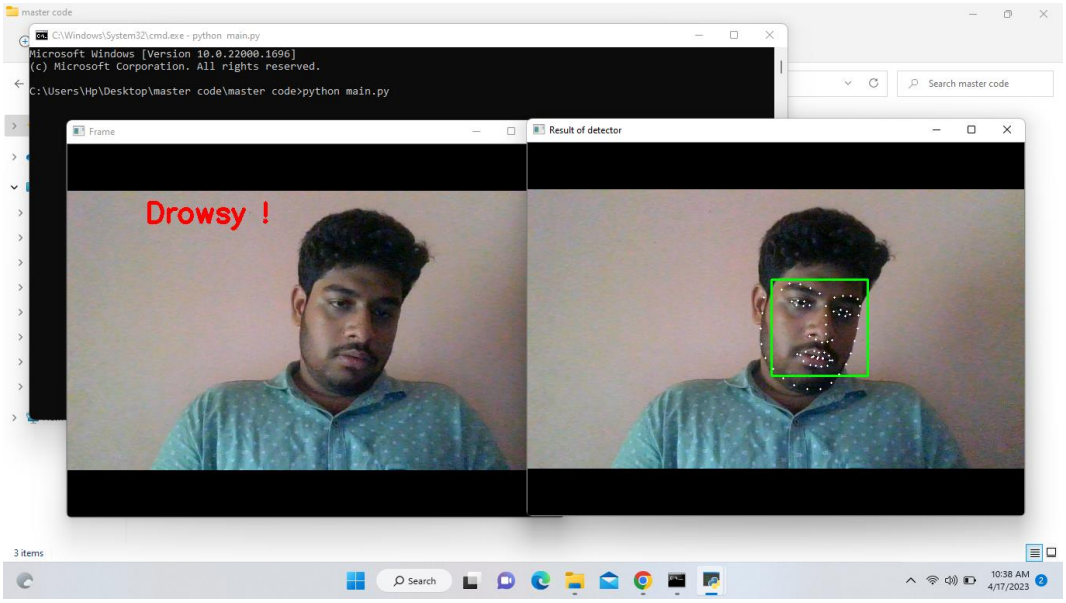


Fig 9.3 Output 3

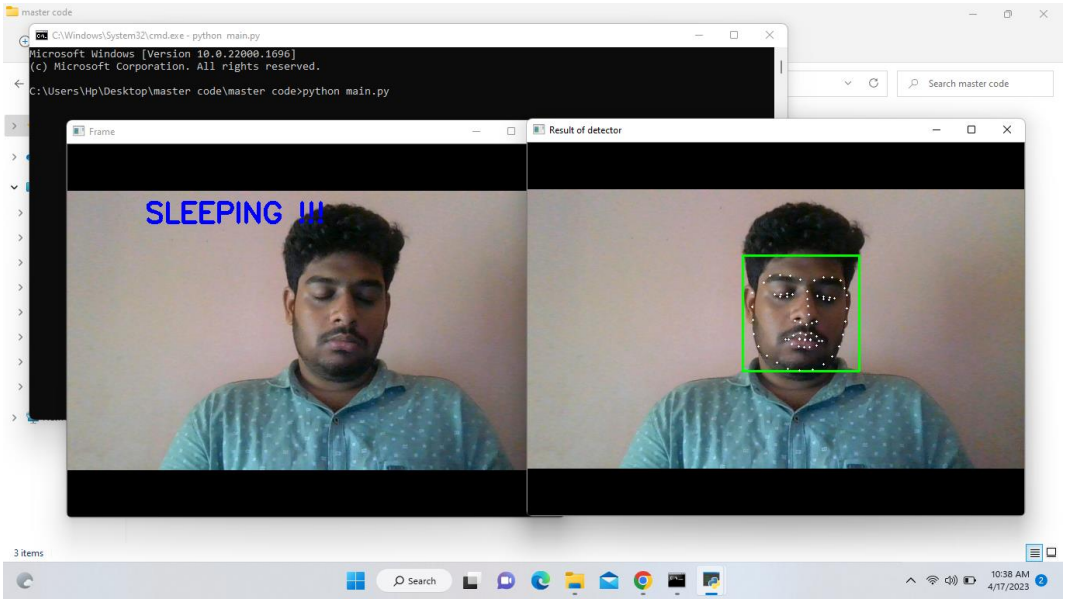


Fig 9.4 Output 4

10. CONCLUSION:

We found that there are many interactive processes between teachers and students in classroom teaching, students' learning attention is also high, and interaction has a higher degree of participation, which is the main reason for students' high attention. In the practice state of classroom teaching, students' attention is also relatively high because the goal of practice is highly oriented. To improve students' attention, we need to improve it from two aspects: students' internal factors and teachers' external teaching methods. Students should be given clear learning objectives; for example, adding some exercises in classroom teaching can also improve students' attention. As another example, giving students multiple small rewards in the learning process is an effective way to improve students' attention. In addition, increasing interactive learning content and carrying out project-based teaching can improve students' attention. In addition, we also found that in the state of independent seat selection, the students in the front row have high attention, and the students in the back row have low attention. In this project, we found that occlusion is a big problem that prevents the system from detecting every face in class when the classroom is very big and there are many students in the classroom. The next step is to add multiple cameras to resolve the problem of occlusion of the back row students' face images, improve the precision of the back row face images, and collect the head video data with high definition to further improve the prediction accuracy of students' attention. The dataset we use to train and test our method in this paper is relatively small; we thus need to collect more snaps of different students and different classes to develop a larger dataset to train and test our method in the next study. The application of this method requires hardware and software systems to acquire and analyze audio and video data. Some schools have been equipped with the required system, but some have not. We can build low-cost portable equipment to meet these needs. In addition, with the development of AI technology, the cost of equipment will keep falling, and AI technology will continuously be applied to education, continuously helping improve education.

REFERENCES:

1. Cebrián, G.; Palau, R.; Mogas, J. The Smart Classroom as a Means to the Development of ESD Methodologies. Sustainability 2020, 12, 3010. [CrossRef]
2. Xiao, N.; Thor, D.; Zheng, M. Student Preferences Impact Outcome of Flipped Classroom in Dental Education: Students Favoring Flipped Classroom Benefited More. Educ. Sci. 2021, 11, 150. [CrossRef]
3. Huang, L.; Su, J.; Pao, T. A Context Aware Smart Classroom Architecture for Smart Campuses. Appl. Sci. 2019, 9, 1837. [CrossRef]
4. Francisti, J.; Balogh, Z.; Reichel, J.; Magdin, M.; Koprda, Š.; Molnár, G. Application Experiences Using IoT Devices in Education. Appl. Sci. 2020, 10, 7286
5. Sun, S.; Liu, R. A Review of Research on Attentiveness Recognition. Sci. Technol. Inf. 2021, 4, 6–8. [CrossRef]
6. Wang, P.; Wang, D. Research on the Application of Focus Recognition Technology in Teaching Monitoring. Comput. Knowl. Technol. 2020, 16, 38–42.
7. Chang, Z. Analysis and Reflection on Concentration of Online Teaching in Vocational Colleges. J. Wuhan Eng. Inst. 2020, 32, 91–94.
8. Deng, Q.; Wu, Z. Students' Attention Assessment in eLearning based on Machine Learning. IOP Conf. Ser. Earth Environ. Sci. 2018, 199, 032042. [CrossRef]
9. Zhong, M.; Zhang, J.; Lan, Y. Study on Online Education Focus Degree Based on Face Detection and Fuzzy Comprehensive Evaluation. Comput. Sci. 2020, 47, 196–203. [CrossRef]
10. Lo, C.K.; Chen, G. Improving Experienced Mathematics Teachers' Classroom Talk: A Visual Learning Analytics Approach to Professional Development. Sustainability 2021, 13, 8610. [CrossRef]
11. Malekigorji, M.; Hatahet, T. Classroom Response System in a Super-Blended Learning and Teaching Model: Individual or Team-Based Learning? Pharmacy 2020, 8, 197. [CrossRef]

12. Farahani, S.; Farahani, I.; Deters, M.A.; Schwender, H.; Burckhardt, B.B.; Laeer, S. Blended, Learning on Blood Pressure Measurement: Investigating Two In-Class Strategies in a Flipped Classroom-Like Setting to Teach Pharmacy Students Blood Pressure Measurement Skills. Healthcare 2021, 9, 822. [CrossRef]
13. Goldberg, P.; Sümer, Ö.; Stürmer, K.; Wagner, W.; Göllner, R.; Gerjets, P.; Kasneci, E.; Trautwein, U. Attentive or Not? Toward a Machine Learning Approach to Assessing Students'Visible Engagement in Classroom Instruction. Educ. Psychol. Rev. 2021, 33, 27–49. [CrossRef]