ABSTRACT

Around 285 million visually impaired persons are visualised worldwide, things considered easy by those with normal vision, such as the retrieval of an item dropped, are a major challenge. Visual sensing technology developments have driven attempts to make people with visual defect (PVIs) more self-reliant in recent years.

Modern breakthroughs in computer vision and AI promise to enhance the lives of disabled people significantly. One such advancement is proposed: a profound learning model. Our model is targeted on recognising products in the food sector, but is a concept symbol for any similar model of visual perception.The Recognition System is coached by a Convolution Neural Network (CNN).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

LIST OF ABBREVIATION

ABBREVIATION	TITLE	PAGE NO.
CNN	Convolution Neural Network	6
Resnet	Residual Neural Network	5
NBNN	Naïve Bayes Nearest Neighbour	12

CHAPTER 1
INTRODUCTION

Recent advancements in computer vision and artificial intelligence (AI) have the potential to significantly enhance the lives of people with impairments. One such advancement is what we suggest. There are approximately 280 million persons with vision impairment in the world today. We propose a Nave Bayes, Open Cv, and ResNet image recognition model. To recognise objects in the grocery store, we suggest developing an accurate and efficient image recognition model. This might make it easier for those with low vision to shop at grocery shops without the assistance of others.Blindness or visual impairment is one of the top 10 impairments among men and women, affecting about 7 million people of all ages in the United States. Accessible visual information is critical for blind and visually impaired people to maintain their independence and safety, and there is a pressing need to develop smart automated systems to aid their navigation, particularly in unfamiliar healthcare environments such as clinics, hospitals, and urgent cares.This contribution focused on constructing deep neural network-based computer vision algorithms to aid visually impaired individuals' movement in clinical environments by properly recognising doors, stairs, and signs, which are the most notable landmarks. Quantitative investigations show that if there are enough coaching examples, the network can detect items of interest with a 98 percent accuracy in a fraction of a second. A vision-based automatic shopping assistant uses a combination of wearable cameras, hardware accelerators, and algorithms to allow persons with poor or no vision to choose goods from grocery stores.There are around 285 million visually impaired persons in the world; for them, things that are considered basic by others with normal vision, such as retrieving a fallen object, are a major job.

Breakthroughs in visual sensing technologies have sparked initiatives to help people with visual impairments (PVIs) become more self-sufficient in recent years. Signal-processing technologies, which are at the heart of visual augmentation, assist PVIs in navigating inside and out, as well as identifying things that are crucial in routine tasks like shopping, especially in supermarkets. Even sighted individuals are sometimes overwhelmed by the volume and variety of products available at a typical US grocery store, which can have up to thirty-five thousand distinct items displayed in up to thirty aisles over forty-five thousand square metres.For PVIs, food shopping is like a neighbourhood with moving carts, people and displays at any turn, in the dimensions of

a gridiron. Some PVIs address family, friends or support workers in order to achieve this onerous task. Others must feel that the shop employees might enjoy or disagree with their expertise and personality during the shopping experience. The online shopping and home delivery services intend to alleviate the problems of the purchase of PVIs in the food industry through the eradication of excursions to the store.Assistive technologies are simply computer vision algorithms that operate on mobile devices, another mitigating effort. Their primary functionality, which is limited to authenticating a previously placed item, quickly breaks apart in scenarios with closely packed setups for several comparable things. In addition to improved viewing algorithms, two versions of cereal brand are frequently not differentiated, for example. Unique segmentation – which is needed to recognise one product during a closely packaged setup – involves aggressive, computer-intensive searches that are sluggish to answer requests.

1.1 Objectives

The following are the objectives of this project:

- To create an accurate image recognition model to aid visually impaired people at grocery shops
- To differentiate between the shelves and the products on the shelves of a grocery store
- To clean and pre-process the dataset.
- To do feature extraction. In traditional computer vision approaches designing these features are crucial to the performance of the algorithm.
- To use naïve Bayes method to get data to use for training and testing.
- To use resnet to build the backbone of the model.
- To use frozen graph for further optimization.
- To show the various parameters of the field in real time.

Sl.No	Author & Year of Publication	Journal	Title of the paper	Advantage(s)	Limitation(s)
1.	R. Ahn and J. Zhan	IEEE	A novel method for visually impaired using object recognition	With the advancement in computer vision and computing technologies we can afford to develop a system for visually impaired people, which can give audio feedback of surrounding objects and context.	The algorithm is evaluated o an online dataset as well as o our dataset.
2.	J. Wang, J. Zhang, and J. Zhan	IEEE	A mobile app for object recognition for the visually impaired	The blind and the visually impaired face diverse kinds of life challenges that normally sighted people take for granted.	In this communication we report a solution aimed at aiding the visually impaired in colour detection, light direction detection and recognition of objects.
3.	M. Wu, J. Zhan, and J. Lin	IEEE	Deep learning for link prediction in dynamic networks using weak estimators	Recent research has focused on extending link prediction to a dynamic setting, predicting the creation and destruction of links in networks that evolve over time.	Weak estimators have been used in a variety of machine learning algorithm to improve model accuracy owing to their capacity to estimate changing probabilities in dynamic systems.
4	J. Zhan and B. Daha	IEEE	Using empirical recurrences rates ratio for time series data similarity	In the modern age, the necessity to construct efficient tools to classify and categorize time series instances is undeniable.	As Liao narrates, several application domains have witnessed such clustering exercises over the years.
5	J. Zhan, S. Gurung, and S. P. K. Pars	IEEE	A weak estimator for dynamic systems	While the nature of expected observations sufficiently guides the former, subject matter and inside knowledge are usually the	The relationship of the underlying Bernoulli distribution with the flow of time is thus not static: it

				only tools to check the latter.	**evolves with it.**

TABLE 1.1 LITERATURE SURVEY

1.3 Need for Image Recognition

Eyesight being an important sensory ability is necessary for humans to function independently. Visually challenged people tend to struggle or depend on others to perform simple everyday tasks. Image recognition could try to solve some of their problems and can make them perform tasks without additional help. Image recognition can be used in various fields, in this study we try to implement it on the grocery store to help the visually challenged people to recognize objects in the shelves of a grocery store.

1.4 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the methodology, design approach, software used and constraints of the methodology used in the project.
- Chapter 3 contains the implementation of Image recognition using naïve Bayes and Resnet.
- Chapter 4 compiles the results obtained after the project was implemented.
- Chapter 5 concludes the report with discussions about the results obtained and their future implications.

CHAPTER 2

IMAGE RECOGNITION FOR GROCERY STORE

This Chapter describes the methodology, design, standards, calibrations, sampling rate and constraints of Image recognition.

2.1 Methodology

The residual network methodology is used in this research; the primary distinction is that ResNets include shortcut connections running parallel to their typical convolutional layers. These shortcut connections, unlike convolution layers, are constantly active, allowing gradients to quickly back propagate via them, resulting in speedier training.

We use the grocery dataset which contains 13184 samples, contains 3 types of data Brand image, product image, shelf image, and also contains dimensions of each product from the images. We pre-process and clean the dataset to apply naïve Bayes to find amount of data needed to for training and testing. We build the Resnet model to achieve accuracy of the model We further apply frozen inference graphs to optimize the result obtained

The data is run across the model. The model differentiates the shelves and products with boxes and recognizes the products on the shelf.

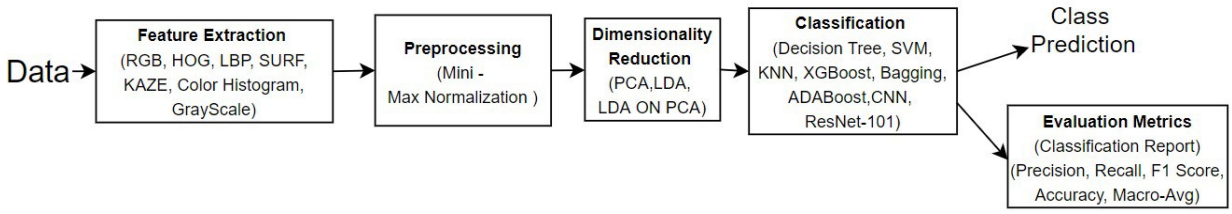


Figure2.1 Block diagram

Although there are certain limitations, this is the first step in establishing a visual navigation system for the interior of healthcare institutions. While the present research did identify and recognise hospital signs, there is much more to signage than that. Converting a hospital sign to a more relevant phrase would be the next stage. In the research field, there are more possibilities to investigate. Stairs and escalators, for example, have a similar design and construction but serve different purposes. In order to tackle the challenge of real-time multi-object detection, or identifying many items in a single image, we want to expand our list of objects to include escalators, barriers, and elevators.

Image-text to audio translation and the development of communication and client-tier technologies are among the projects planned for the future. Object detection might turn smart phones, tablets, and electronic glasses into essential assistance for the visually impaired, partly blind, and blind. The current research is likely to pique the interest of the deep learning and computer vision communities, as well as raise attention to the need to enhance the lives of visually impaired individuals. Utaminingrum et al. presented a system that uses a multiple phase technique to identify text on items. Detections in maximally stable extremely regions (MSER), canny edge detection, region filtering, and abounding ox are all part of the technique. This system was able to achieve an accuracy rate of 80%. The gadget was created for patients with vision impairments ranging from severe to full blindness.

This target population was chosen because their illnesses cause their eyesight to deteriorate to the point that typical treatments, such as corrective glasses or surgery, are ineffective. To obtain an accurate prediction model in deep learning, network topologies can be de-signed in a variety of ways. We used a convolutional neural network for our aims (CNN). A convolutional neural network is a type of neural network that is specially designed for image processing efficiency and accuracy. A CNN, like other neural networks, is made up of neurons with adjustable weights and biases. As a function of weights, biases, and an activation function, these linked neurons receive input and create an output. The output is utilised to aid in the detection of patterns in a picture, which leads to categorization.

2.2 Design Approach

Res-net Architecture:

We all know that, given enough capacity, a feed forward network with one layer is capable of representing any function, according to the universal approximation theorem. However, because the layer may be large, the network may be prone to information overfitting. As a result, there's a general consensus among researchers that our specifications need to be more detailed.As the number of epochs grows, the error rate decreases.

Figure 2.2 conv2_x architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

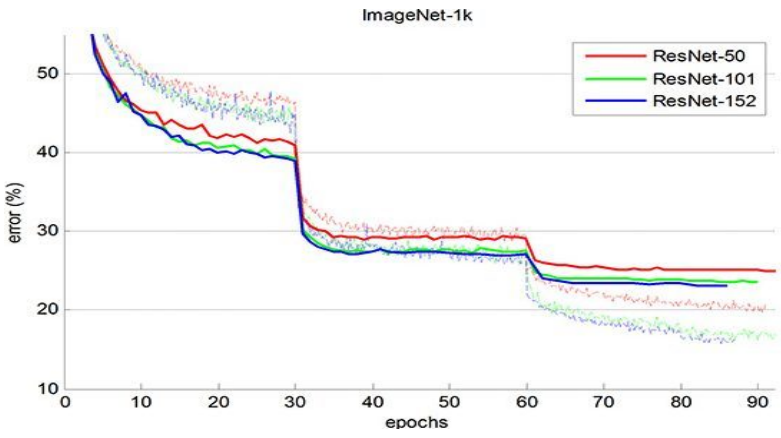


Figure 2.3 The above is errors versus epochs with dataset taken form Imagenet library

Convolution Neural Networks

CNNs are a category of Deep Neural Networks that may acknowledge and classify specific options from pictures and are wide used for analysing visual pictures. Their applications vary from image and video recognition, image classification, medical image analysis, laptop vision and information science. The term ‘Convolution” in CNN denotes theperformance of convolution which can be a special quite linear operation whereby 2 performs are increased to produce a third perform that expresses however the shape of one function is changed by the other.

In straightforward terms, 2 pictures which can be delineated as matrices are increased to supply associate degree output that is used to extract options from the image. The convolutional neural network (CNN) belongs to the deep learning neural network class. CNNs are a significant advancement in object identification. They are most often used in evaluation of visual mental imagery and are utilised in picture classification behind the scenes.

They're convenient and cost-effective. The method of obtaining a class or a likelihood that the input is also a selected class from an associate degree input is called as image.

Convolutional layers are present in a CNN.

A completely linked layer

ReLU layers

Pooling layers

This is how a traditional CNN design might look:

Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling -> ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected

A CNN employs 2nd convolutional layers to convolve learnt choices with the input file. This implies that this network is valid for process 2nd images. CNNs require relatively little or no pre-processing as compared to other image classification techniques. This implies that they'll learn how to create filters in alternate algorithms that involve hand-crafting. CNNs are commonly used in image and video recognition, image classification, and recommender systems and medical image analysis.

The trend of property across the CNN is derived from their research of the cortical area's structure. Individual neurons in an animal's eye respond to visual input only at intervals of receptive field, which would be a limited space. Assorted areas' receptive fields partially overlap, resulting in a roofed overall field of view. This is frequently how a CNN operates. CNNs have hidden layers, an associate degree input layer, and an output layer.

Convolutional layers, ReLU layers, pooling layers, and completely linked layers are basic hidden layers.

Convolutional layers use the input to perform a convolution operation. The information is then passed on to the next tier.

Pooling merges the outcomes of groups of nerve cells in to neuron in each subsequent layer, whereas completely linked layers connect every nerve cell in one layer to each and every nerve cell in the next.

A convolutional layer's neurons only receive input from a portion of the preceding stage. In an extraordinarily well-connected layer, every nerve cell accepts data from every section of the layer before it. A CNN is a machine that extracts choices from images. This eliminates a need to manually extract features. The selections are learned rather than trained, whereas the network is trained on a collection of pictures. Deep learning models are incredibly accurate for laptop vision applications as a result of this.Feature detection is taught to CNNs over tens or even dozens of hidden layers. The complexity of the learnt options will rise with each layer.

A CNN

It begins with a single input image and applies a variety of filters to create a feature map

• increases non-linearity by performing a ReLU function

• Each feature map is given a pooling layer

• The merged pictures are flattened into a single long vector

The vector is fed into a fully linked neural network.Which then analyses the alternatives. The ultimate fully linked layer allows us to "vote" on the categories we were previously in.

It trains over many, many epochs using forward propagation and backpropagation. This process is repeated till the neural network is well-defined, with training weights and detectors.

• Those pixels are taken as a second array in a black-and-white picture (for e.g. 2x2 pixel). Each element has a value ranging from 0 to 255. (0 is black, while 255 is white.) Between those values, there is a greyscale.) Assuming the data is supported, the computer will proceed to process the input.

• A 3D array with a blue layer, an inexperienced layer, and a red layer is commonly used for a colour image. Each of these colours has a value ranging from 0 to 255. Combining the values in each of the three levels yields the colour.

Convolution

The convolution step's main goal is to extract choices from the input image. During a CNN, the convolutional layer is frequently the first step.

We have a feature detector, a feature map, and an associate degree input picture. You take the filter and apply it to the input picture element by element block by block. You're accomplishing this by multiplying the matrices.

The light from the electric bulb acts as a filter, and the receptive field is the region you're sliding across. Your electric lighting is convolving as the sunlight slides across the receptive fields. Your filter consists of an array of integers (also known as weights or parameters).

The filter's depth should be the same as the input's depth, therefore the depth would be three if we were watching a colour image. As a result, the filter's size is 5x5x3. The filter multiplies the values within the filter by the first values within the element at every location. This is sometimes referred to as part-by-part multiplication. The multiplications are added together to form a single variation. This variation is symbolic of the highest left corner of your bubble wrap if we start at the highest left corner. We will now move our filter towards the next spot on the bubble wrap and repeat the process all the way around. You will set parameters like the filter numbers, size, and network architecture, among others.

The CNN acquires the values of the filters on its own throughout the coaching process. We have a number of alternatives to choose from when it comes to creating the most effective image

classifier for your needs. You'll need to pad the input matrix with zeros to apply the filter on the bordering components of the input image matrix (zero padding). You may also adjust the scale of the feature maps this way. The method of introducing zero artefacts is known as wide convolution. It's difficult to get zero artefact convolution.

This is primarily how we locate photos; we don't study each and every aspect of a photograph. Options such as a cap, a blue shirt, a tattoo, and others are common. We can't possible alter every single piece of information that enters our eyes at any one time. We're letting the model attempt to address this very same issue.

The outcome is frequently a convolved feature map. It's a lot smaller than the first image. This makes influencing easier and faster. Will information be lost? Yes, there are some. At the same time, the feature findor's goal is to identify choices, and that is exactly what this will achieve.

To motivate our first convolutional layer, we create many feature maps. This enables the United States of America to identify a wide range of possibilities that the programme will investigate. Feature detectors might be identified with completely different values to provide completely different outcomes. A filter that sharpens and focuses a photograph or blurs a photograph, for example, might be used. All or any of the values may be given equal weight. Edge sweetening, edge identification, and other tasks will be performed. You'd do this by creating various feature maps using completely distinct feature detectors. The computer is capable of determining which filters make the most sense and putting them into action.

The basic purpose is to locate alternatives in your image, add them to a feature map, then maintain the pixel abstraction relationship intact. This is necessary to keep the pixels from becoming messy.

2.3 Software Used

This section describes the software and platforms used.

2.3.1 Python

Python is a complex degree, all-serving programming language with an associate's degree. As seen by its heavy use of indentation, Python's style vision prioritises code interpretability. Its language components and object-oriented approach are designed to let programmers write concise, logic snippets for a wide range of applications.

Python is a trash collecting and powerful programming language. Structured programming, oops, and useful programs are among the paradigms it supports. Because of its vast standard library, Python is usually referred to as a "batteries included" language.

Design philosophy and options

Python is a coding language that may be used in a variety of ways. Oops and levelled programming are both available, and lot of the options allow for useful programming and aspect-oriented programming. Several alternative paradigms, such as style by contract and logic programming, are enabled by extensions.
Python manages memory using dynamic typewriting, a combination of reference investigating, and a cycle-detecting refuse collector. It also has dynamic name resolution, which binds methods and variable names while the programme is running.
Python's approach lends itself to practical programming in the Lisp tradition.

It includes aphorisms such as:
• Beautiful is preferable to ugly
• Explicit is preferable to implicit.
• Simplicity is preferable than complexity.
• Complexity is preferable to difficulty.
• Readability is important.

Python follows the principle of "there should be one— and ideally only one clear way to accomplish anything." "To label something as 'smart' isn't thought of as a praise inside the Python culture," argues

Python's developers aim to prevent premature enhancement by rejecting modifications to non-critical components of the CPython reference implementation that provide minor performance gains at the expense of clarity. When performance is necessary, a Python applied scientist will transfer time-sensitive functions to C extension modules or utilise PyPy,

Python's creators have made it a priority to maintain the language enjoyable to use. This is reflected in the language's name, which is an homage to the British comic troupe Monty Python, as well as in the language's frequently frolicsome attitude to tutorials and reference materials, such as examples that ask for spam and eggs rather than the high-quality foo and bar.

The term pythonic is a frequent neologism in the Python community, and it may refer to a wide range of software styles. To say that code is pythonic means that it makes good use of Python idioms, that it's natural or fluent in the language, and that it adheres to Python's minimalist philosophy and emphasis on readability.

Pythonistas are Python users and enthusiasts, especially those who are believed to be competent or experienced.

Linguistics and syntax

Python is designed to be an easy-to-understand programming language. Its data style is neat and orderly, and it frequently employs English keywords instead of punctuation in other languages. In contrast to numerous other languages, it does not employ ringleted brackets to delimit blocks, and semicolons are rarely, if ever, used when statements are allowed. In comparison to C or Pascal, it has fewer grammatical exceptions and special circumstances.

Indentation is a type of indentation that occurs

To delimit blocks, Python utilises whitespace indentation rather than ringleted brackets or keywords. When bound statements are present, indentation rises; when bound statements are not present, indentation decreases. As a result, the visual organisation of the programme precisely reflects the verbal structure of the programme. This trait is sometimes referred to as the off-side rule, which is shared by numerous languages; nevertheless, indentation in most languages has no linguistic meaning.
Flow of statements and management

Development environments

The read–eval–print loop (REPL) is used by most Python implementations (including CPython), allowing it to act as a statement deduced in which a user input statements in order to get responses directly. Other shells, such as IDLE and Python, provide additional features such as enhanced auto-completion
There are internet browser-based IDEs, Sage scientific Python anywhere, a browser-based IDE and hosting environment, and cover IDE, a commercial Python IDE focusing on scientific computing, in addition to standard desktop integrated development environments.

2.3.2 OpenCV:

OpenCV (Open Source Computer Vision Library) is a programming library primarily aimed at time-based computer vision. The OpenCV project, which was formally started in 1999, was originally an Intel research programme to develop CPU-intensive applications, as part of a series of collaborations that included temporal ray tracing. The project's biggest contributors were a range of Intel Russia improvement consultants, as well as Intel's Performing Team. The project's aims were represented as follows throughout the OpenCV period:

• Advance vision analysis by offering fundamental vision infrastructure code that is not only open but also optimised. There's no need to recreate the wheel.

Why disseminate vision data by offering a standard architecture on which developers may rely, ensuring that code is quickly decipherable and transferrable.

• Advance vision-based commercial applications by providing portable, performance-optimized code that could be downloaded for free under a licence that didn't need the code to be open or free.

2.3.3 Keras:

One of the most important sub category in machine learning domain is deep learning. Machine learning is the learning of algorithmic styles inspired by the human brain model. Artificial intelligence (AI), audio and video identification, and picture identification are all areas where deep learning is becoming increasingly popular. The artificial neural network is the cornerstone of deep learning techniques. Deep learning is supported by a number of libraries, including Theano and others. Keras is a strong and easy-to-use Python library for generating deep learning models, built on top of well-liked deep learning frameworks such as TensorFlowand others.

Overview of Keras:

TensorFlow, Theano, and the Psychological Feature Toolkit are all open source machine libraries that Keras is built on (CNTK). Theano is a Python module for doing fast numerical computations. TensorFlow is the most widely used symbolic mathematics library for building neural networks and deep learning models. TensorFlow is incredibly versatile, and its main value is distributed computing. Although Theano and TensorFlow are powerful libraries, they may be difficult to comprehend when it comes to creating neural networks.Keras uses a token format that is clear and straightforward to create deep learning models that are supported by TensorFlow or Theano. Keras is a tool for quickly sketching out deep learning models. Keras, on the other hand, is an excellent choice for deep learning applications.

• Keras makes use of a variety of optimization approaches to make creating a high-level neural network API easier and more performant. It has the following capabilities:

• A straightforward, and obtrusible API.

• Simple structure - no frills make it easier to achieve the desired effect.

• It works with a variety of platforms and backends.

Benefits:

• Keras is a robust and dynamic framework that offers the following advantages: • A larger community base of support; • Checking is straightforward.

• Keras neural networks are built in Python, making them easier to work with.

• Keras is capable of dealing with both convolutional and continuous network.

• Deep learning models are made up of a range of components that may be mixed in a number of different ways.

Overview:

Deep learning is a subfield of machine learning that is fast evolving. Deep learning involves layer-by-layer analysis of the input, with each layer obtaining higher-level data about the input. Consider the easy job of evaluating an image. We may assume that the pixels in your given image are separated into an oblong grid because of this. The main layer has now abstracted the pixels. The second layer recognises the image's perimeters. The perimeters are used to generate nodes in the next tier. Branches would grow from the nodes after that. Finally, the output layer may find the full item.

In this example, the feature extraction technique flows from one layer's output to the next layer's input. This technique allows us to method a large number of options, making deep learning a highly powerful tool. Unstructured data can potentially benefit from deep learning algorithms.

Artificial Neural Networks:

The most well-known and earliest deep learning technology is the "artificial neural network" (ANN). They're based on a model of the human brain, which is the most sophisticated organ in our body. The human brain is made up of around ninety billion microscopic cells known as "neurons." Axons and dendrites are the fibres that connect neuron to neuron. A nerve fibre's primary role is to transport data from one vegetative cell to another. The primary purpose of dendrites is to accept data from the axons of the vegetative cell to which they are connected.

Every vegetative cell analyses a tiny bit of information before passing it on to the next vegetative cell in the chain. Nodes are firmly coupled and organised into several hidden levels. The input layer gets the data, which then passes through one or more hidden layers in order to arrive at the output layer, which predicts something useful about the input file. Nodes are interconnected and organised into numerous levels that are concealed. In order to reach the output layer, the data must travel through one or more hidden layers, each of which predicts something important about the input file.

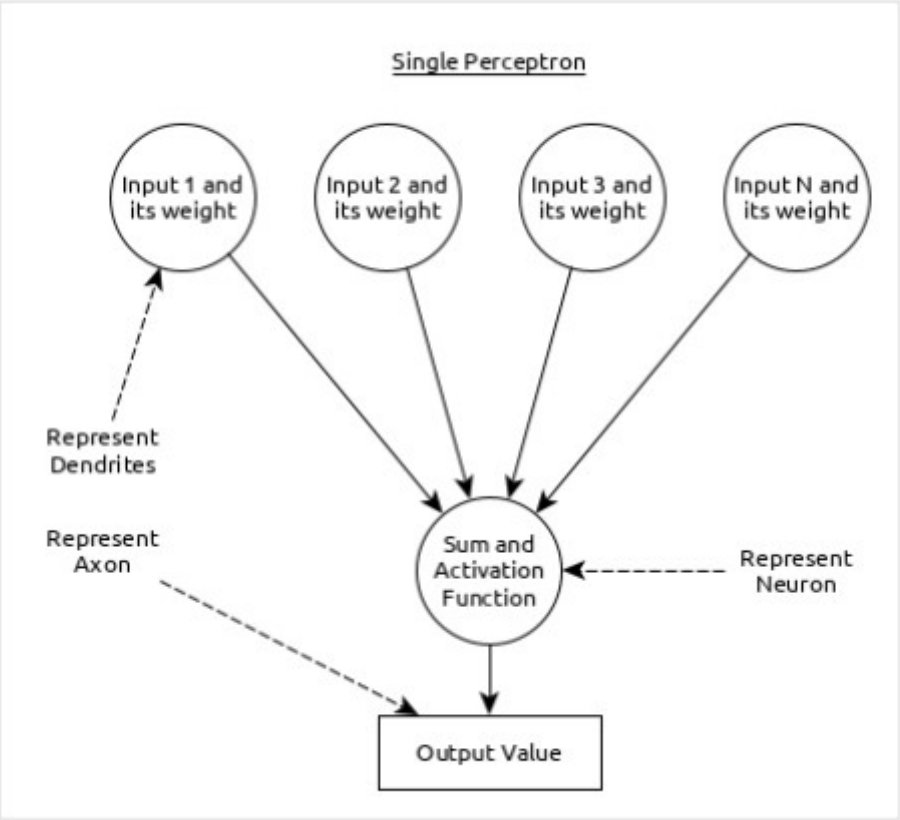


Figure 2.4 Single perceptron

Keras

•Dendrites are represented by many inputs in addition to weight.

•Neurons are represented by the sum of input plus the activation function. Sum actually indicates that the sum of all inputs is computed, and the activation function is a function that modifies the Sum value to zero, one, or zero to one.

•The actual output is a nerve fibre, hence the output will be received by a vegetative cell in the next layer.

Multi-Layer Perceptron:

The multi-layer perceptron is the most basic form of ANN. It consists of a single inp layer, one or higher unseen layers, and a final Associate in Nursing output layer. A layer is a collection of perceptron. In most cases, the inp layer consists of one or more input file choices. Each hidden layer is made up of one or more neurons that are method-linked to a feature and transmit processed data into the main layer into a hidden layer. The output layer approach extracts data from the final hidden layer and outputs the final result.

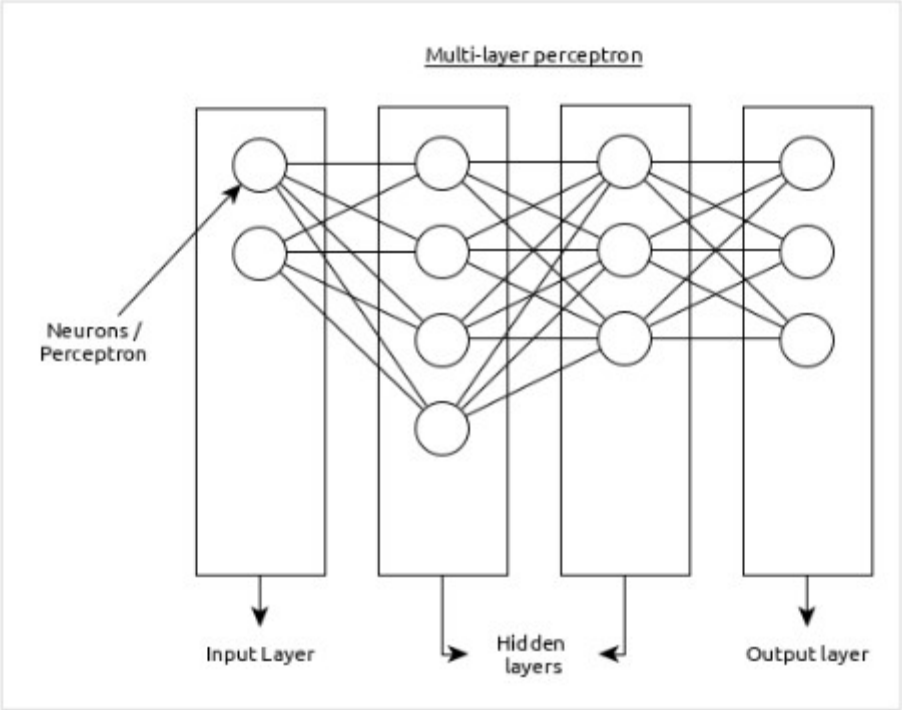


Figure 2.5 Multilayer Perceptron

Convolutional Neural Network (CNN):

The convolutional neural network is frequently used ANNs. It is frequently utilised in the fields of image and video recognition. It validated the notion of convolution in mathematics. It's comparable to a multi-layer perceptron, except it contains a series of convolution and pooling layers before the fully linked hidden vegetative cell layer.

Its 3 vital layers:

•Convolution layer: This is the most basic building piece, and it performs procedural tasks that are aided by convolution.

•Pooling layer: Located next to the convolution layer, this layer is utilised to minimise the quantity of inputs by deleting extraneous data, allowing calculation to be performed quicker.

•Fully linked layer: This layer divides input into multiple categories and is coupled to a series of convolution and pooling layers.

•The input is received and processed by two sets of convolution and pooling layers.

•There is just one totally linked layer, which is configured to emit data (e.g. Classification of image) Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN) are useful for addressing flaws in other ANN models. The majority of ANNs, on the other hand, do not remember the steps from prior events and have learned to generate decisions that are supported by context in coaching. Meanwhile, RNN saves the previous data, and all of its decisions are based on what it has learned in the past.

This method is very useful for image classification. Sometimes, in order to mend the past, we may have to look into the future. In this scenario, a bifacial RNN can be used to learn from the past and forecast the future. We've written samples with various inputs, for example.

Workflow of ANN

Allow us to first comprehend the many phases of deep learning, then learn how Keras aids in the deep learning process.

Gather the information you'll need.

Deep learning takes a large amount of data to learn and predict the outcome successfully. So, first and foremost, get as much knowledge as possible.

• Examine your knowledge

• Analyze the data and have a thorough grasp of it. To choose the best ANN algorithmic rule, a better knowledge of the data is required.

• Algorithmic algorithm for selecting Associate in nursing (model)

Choose an ANN algorithmic rule that best fits the type of learning method (e.g., image classification, text processing, etc.) and the input file provided. Modelin Keras portrays algorithmic rule. One or more levels are included in the algorithmic rule. Keras Layer represents each layer in ANN.

In Keras.

•Prepare data: Process, filter, and choose only the relevant information from the data.

•Distribute data: Divide the data into coaching and data sets. Check data will be used to analyse the algorithmic rule's / Model's prediction (once the machine learns) and to double-check the educational method's efficacy.

•Compile the model:

Compile the algorithmic rule / model such that it can be utilised for more than only coaching and prediction. This step necessitates a decision by the North American country on loss perform and Optimizer. In the learning section, the loss perform and Optimizer are used to search out the error (difference from the real output) and execute optimization such that the error is reduced.

•Fit the model: The actual learning approach will be conducted in this section by utilising the coaching data collection.

•Predict the outcome of an unknown value:

Predict the output for a file with an unknown input

Model evaluation: Assess the model by anticipating check information output and comparing the prediction to the actual check information outcomes. •Freeze, change, or choose a new algorithm: Check to see if the model's analysis was successful. If this is the case, save the algorithmic rule for future prediction. If not, tweak or pick a new algorithmic rule / model, and then train, predict, and evaluate the model once more. Rep until the most successful algorithmic rule is discovered.

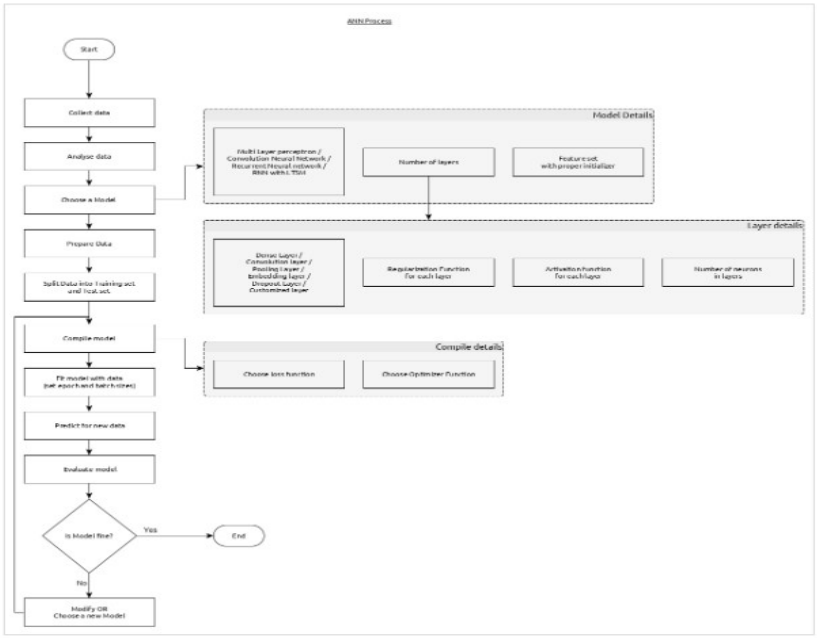


Figure 2.6 ANN Model

Architecture of Keras

The Keras API is broken into three sections:

•Layer

•Model

•Primary Modules

Keras Models represent each ANN in Keras. Each Keras Model, on the other hand, is made up of Keras Layers and represents ANN layers such as input, hidden layer, output layers, convolution layer, pooling layer, and so on. Keras model and layer access Keras modules for activation, loss, and regularisation, among other things. Any ANN algorithmic rule (CNN, RNN, etc.) can be represented in a very simple and cost-effective method using the Keras model, Keras Layer, and Keras modules.

CHAPTER 3

DEEP LEARNING BASED SHOPPING ASSISTANT FOR VISUALLY AIDED PEOPLE

This chapter describes the implementation ofdeep learning based shopping assistant for visually aided people. It describes Naïve bayes theorem, Resnet architecture, Frozen inference graph.

3.1 Naïve BayesTheorem

The Naive Bayes Nearest Neighbour (NBNN) approach to object classification has been designed as a powerful, learning-free, non-parametric approach. The rejection of a vector division step and, as a result, the utilisation of image-to-class comparisons, giving smart generalisation, are the main reasons for its smart performance.

The Bayes Theorem is backed by a series of classification algorithms known as naive Bayes classifiers. It is a family of algorithms rather than a single formula, and they all share a common premise, namely that each try of possibilities being classified is independent of every other.

The dataset is divided into two parts: the feature matrix and, as a result, the response vector.

•The feature matrix contains all of the dataset's vectors (rows), each of which contains the value of dependent choices.

•For each row of the feature matrix, the response vector comprises the value of the sophistication variable (prediction or output).

The basic assumption of a Naive mathematician is that each feature contributes an equal and independent contribution to the eventual result.

This concept will be understood in relation to our dataset as follows: We presume that no try of options is dependant.

Secondly, every feature is given a similar weight (or importance).

None of the attributes isirrelevant and assumed to be contributing **equally** to the outcome.

```
[ ]  df = products_df[products_df.category != 0].\
             groupby(['category', 'is_train'])['category'].\
             count().unstack('is_train').fillna(0)
     df.plot(kind='barh', stacked=True)
```

Figure 3.1 Naïve Bayes Theorem

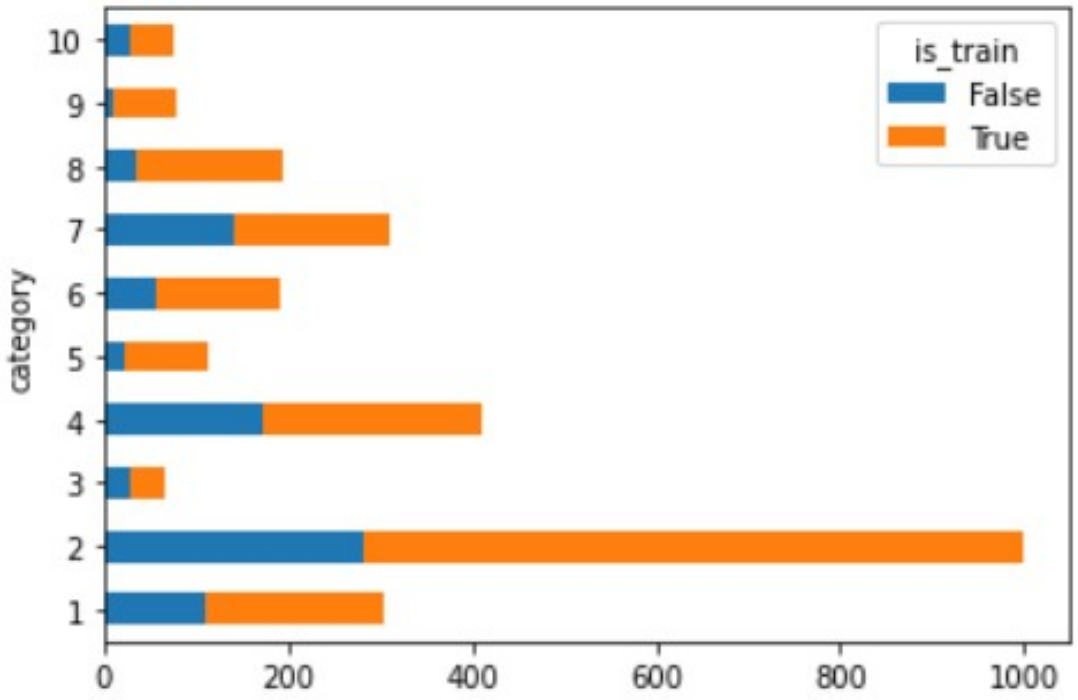


Figure 3.2 Naïve Bayes Graph

Figure 3.3 Result Of OpenCV

3.2 Open CV

Many real-time computer vision applications use OpenCV as an image processing library. In OpenCV, there are thousands of functions to choose from.

Detecting the edges

By extracting edges from a picture, edge detection provides an outline of the image. This work is accomplished with the help of a clever filter. A canny filter finds the edges based on the threshold values. Higher thresholds produce cleaner photos, but lower thresholds provide clunky results.

Techniques for Smoothing

A filter is used to conduct the image smoothing procedure. By adding a blurring effect to the edges of the image, convolving it decreases the noise in the image.

Average using cv2.blur()

In this technique, depending on the size of the filter convolution is performed but it averages the pixels under the filter and assigns the value to the centre pixel under the filter.Depending on the supplied image, we use several sorts of smoothing algorithms.

In most cases, the information that we have a tendency to gather has noise in it i.e. unwanted options that creates the image exhausting to understand. Though these pictures may be used directly for feature extraction, the accuracy of the formula would suffer greatly. This is often why image process is applied to the image before passing it to the formula to induce higher accuracy.

There ar many various styles of noise, like Gaussian noise, salt and pepper noise, etc. we will take away that noise from a picture by applying a filter that removes that noise, or at the terribly least, minimizes its result. There ar heaps of choices once it involves filters similarly, every of them has totally different strengths, and therefore is that the best for a particular quite noise.

To understand this properly, we have a tendency to ar progressing to add 'salt and pepper' noise to the grayscale version of the rose image that we have a tendency to thought of on top of, then try and take away that noise from our reedy image mistreatment totally different filters and see that one is best-fit for that kind.

```
# function to display shelf photo with rectangled products
def draw_shelf_photo(file):
    file_products_df = products_df[products_df.file == file]
    coordinates = file_products_df[['xmin', 'ymin', 'xmax', 'ymax']].values
    im = cv2.imread(f'{shelf_images}{file}')
    im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
    for xmin, ymin, xmax, ymax in coordinates:
        cv2.rectangle(im, (xmin, ymin), (xmax, ymax), (0, 255, 0), 5)
    plt.imshow(im)
```

```
[ ] # draw one photo to check our data
    fig = plt.gcf()
    fig.set_size_inches(18.5, 10.5)
    draw_shelf_photo('C3_P07_N1_S6_1.JPG')
```

Figure 3.4 Open Cv Criteria

Figure 3.5 Shelf Photo with Rectangle Boxes

3.3 Keras

Extraction of Features

The neural network needs do feature extraction in order to do picture recognition/classification. The features of the data that you care about and will be sent over the network are called features. The features of an object in image recognition are groups of pixels, such as edges and points that the network will evaluate for patterns.

Functions of Activation

The values that represent the image are processed via an activation function or activation layer once the picture's feature map has been constructed.Because images are non-linear, the activation function takes values that describe the image, which are in a linear form owing to the convolutional layer, and amplifies their non-linearity.

Stacking Layers

The data is then transmitted through a pooling layer after it has been triggered. Pooling "downsamples" an image, which means it compresses the information that makes up the image, making it smaller. The network becomes more flexible and adept at recognising objects/images based on relevant attributes thanks to the pooling process.

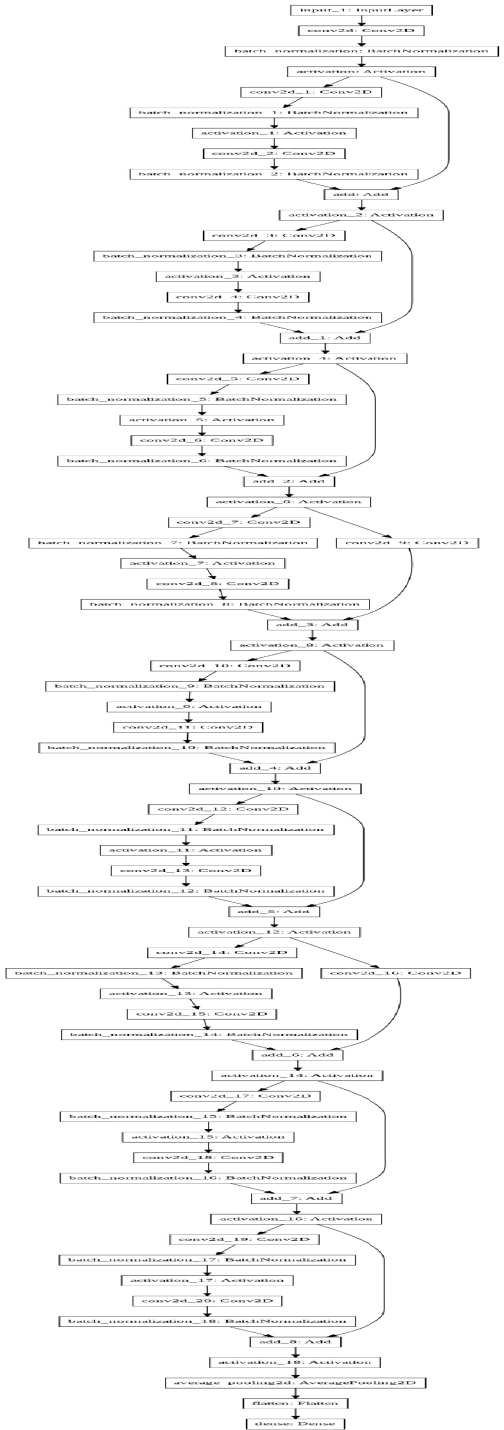
Deflation

The data must be in the form of a vector to be processed by the last levels of our CNN, the densely linked layers. As a result, the information must be "flattened." The data is compressed into a lengthy vector or a column of integers in a specific sequence.

3.4Resnet Architecture

We use normalized image for faster processing and to have the values in Gaussian curve.

Algorithm



$$y = x_l + F(x_l, \{W_i\})$$

$$x_{l+1} = H(x) = ReLU(y)$$

Y= Addition output.

X(l+1)=Input to next block.

- Uses 20 conv2d layers, 18 Activation layers, 18 batch normalization layer, 8 addition layers, 1 average pooling layer, 1 flatten layer, 1 dense layer.
- Activation function As RELU.
- Learning rate as 0.001.
- Single stride.
- Kernel size as 3.

This is the generated plot of our model

3.5 Floating Inference Graph

Frozen_inference_graph.pb, is a frozen graph that cannot be trained anymore, it defines the graphdef and is actually a serialized graph and can be loaded with code

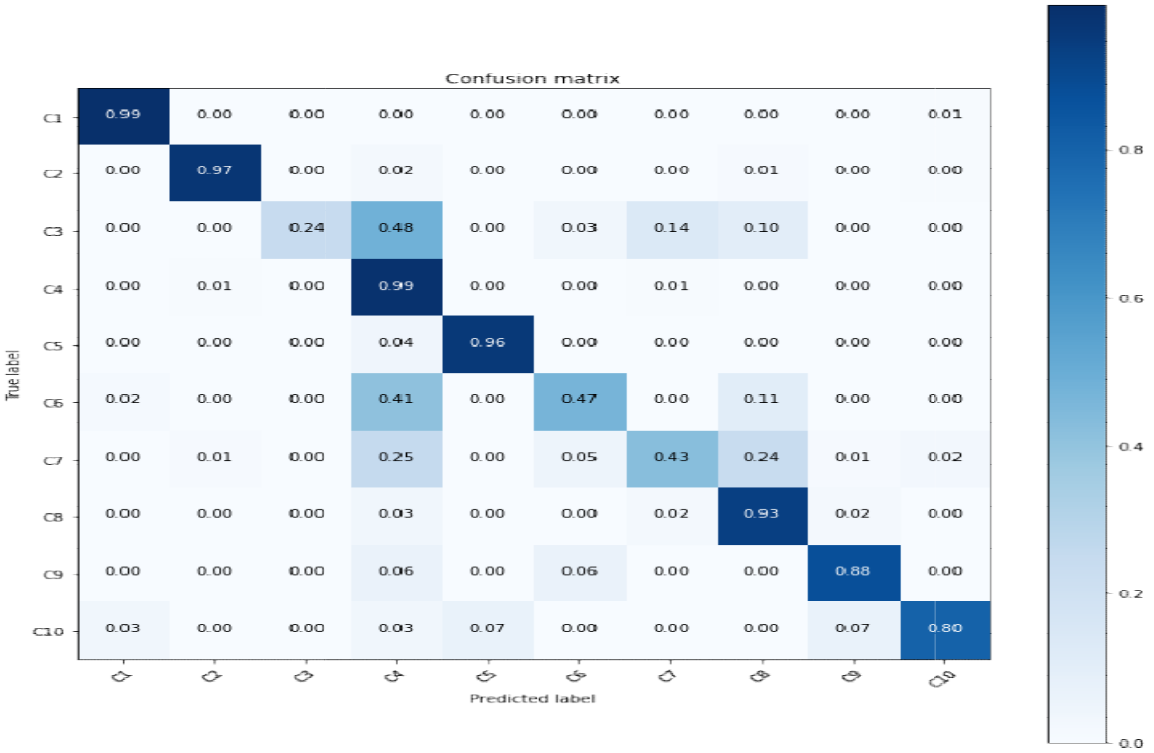


Figure 3.6 Confusion Matrix

```
[ ]  # load frozen graph
     detection_graph = tf.Graph()
     with detection_graph.as_default():
         od_graph_def = tf.compat.v1.GraphDef()
         #od_graph_def  = tf.function()
         with tf.compat.v1.io.gfile.GFile(PATH_TO_MODEL, 'rb') as fid:
             serialized_graph = fid.read()
             od_graph_def.ParseFromString(serialized_graph)
             tf.import_graph_def(od_graph_def, name='')
```

```
[ ]  # let's write function that executes detection
     def run_inference_for_single_image(image, image_tensor, sess, tensor_dict):
         # Run inference
         expanded_dims = np.expand_dims(image, 0)
         output_dict = sess.run(tensor_dict, feed_dict={image_tensor: expanded_dims})
         # all outputs are float32 numpy arrays, so convert types as appropriate
         output_dict['num_detections'] = int(output_dict['num_detections'][0])
         output_dict['detection_classes'] = output_dict['detection_classes'][0].astype(np.uint8)
         output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
         output_dict['detection_scores'] = output_dict['detection_scores'][0]
         return output_dict
```

Figure 3.7 Frozen Graph Criteria

3.6 Summary

In this chapter the implementation of Image Recognition has been described. The Naïve Bayes Theorem describes how the data is distinguished for training and testing .The Open Cv describes the techniques used to process the images. The Resnet architecture is used for the recognition of image while frozen graph is used for optimization.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 RESNET BOUNDED OUTPUT

Output obtained after implementing the resnet model.

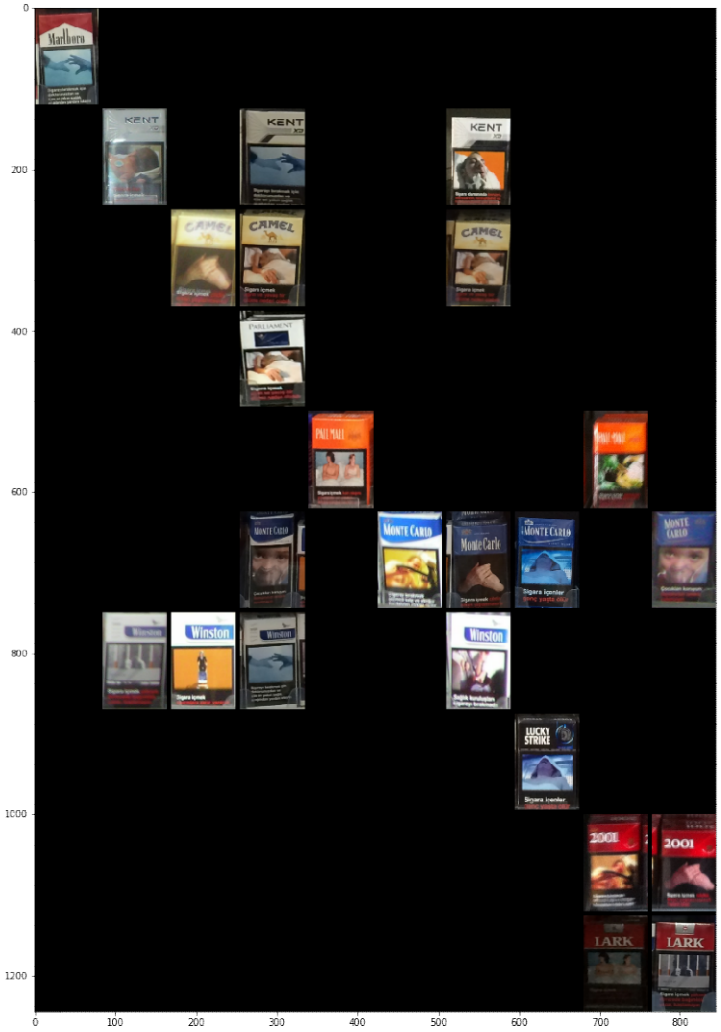


Figure 4.1 Resnet Bounded Output

4.2 Modification from the model

A frozen inference graph is been added for further optimization. The graphs contain the probability value of the used model with more precision and accuracy. These values are used as the weights for the resnet model and the output is being optimized

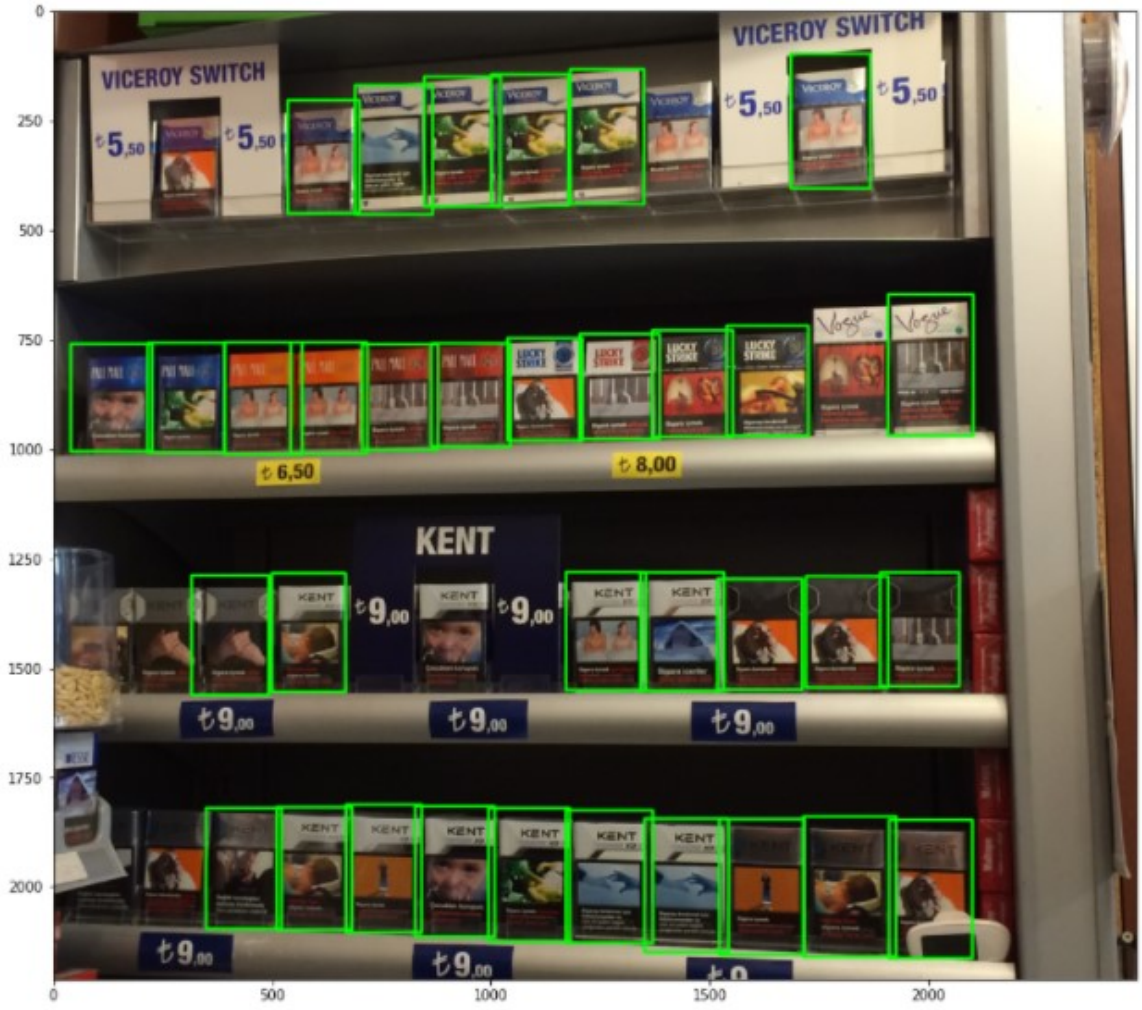


Figure 4.2 Resnet Model

Figure 4.3 Frozen inference graph model input

CHAPTER 5
CONCLUSION AND FUTURE WORK

This chapter concludes the thesis on deep learning based shopping assistant for visually aided peopleand suggests additional functionalities for future implantation.

5.1 Conclusion

We were able to develop and create a model that supports the severely vision handicapped in a market scenario using a convolution neural network and deep learning, with accuracy exceeding the criteria set by earlier research. Our product appeals to millions of people throughout the world, and we intend to build on the prototype shown in this paper in the future. Our methodology gives a solution to an everyday task made impossible by a disability by providing an answer to an issue that impacts so many people.

.

5.2 Future work

This would considerably broaden the scope of our model's application. To do so, the present image collection would need to be greatly enlarged to accommodate all of the new product categories to classify. It would also be able to detect products in real time rather than waiting for the user to manually scan for an object with greater computing power. In addition to the initial function of classifying and providing a price for a single product, this might be done to assist the user in finding products throughout the market.

APPENDIX

Uploading the dataset

```
mkdir -pv data/images
! wget https://github.com/gulvarol/grocerydataset/releases/downl
oad/1.0/GroceryDataset_part1.tar.gz
! wget https://github.com/gulvarol/grocerydataset/releases/downl
oad/1.0/GroceryDataset_part2.tar.gz
! tar xvf GroceryDataset_part1.tar.gz
! tar xvf GroceryDataset_part2.tar.gz
```

Naïve Bayes

```
import cv2
import pandas as pd
from matplotlib import pyplot as plt
import os
import numpy as np
from sklearn.model_selection import train_test_split
%matplotlib inline

data_path = 'data/'
# we'll use data from two folders
shelf_images = 'data/images/ShelfImages/'
product_images = 'data/images/ProductImagesFromShelves/'


# let's get all shelves photo data from ShelfImages
jpg_files = [f for f in os.listdir(f'{shelf_images}') if f.endsw
ith('JPG')]
photos_df = pd.DataFrame([[f, f[:6], f[7:14]] for f in jpg_files
],
                         columns=['file', 'shelf_id', 'planogram
_id'])
photos_df.head()



# let's get products on shelves photo from ProductImagesFromShel
ves
products_df = pd.DataFrame(
    [[f[:18], f[:6], f[7:14], i, *map(int, f[19:-
4].split('_'))]

     for i in range(11)
```

```
     for f in os.listdir(f'{product_images}{i}') if f.endswith('
png')],
    columns=['file', 'shelf_id', 'planogram_id',
             'category', 'xmin', 'ymin', 'w', 'h'])



# convert from width height to xmax, ymax
products_df['xmax'] = products_df['xmin'] + products_df['w']
products_df['ymax'] = products_df['ymin'] + products_df['h']
products_df.head()


# get distinct shelves
Shelf = list(set(image_df['shelves_id'].value))


# use train_test_split from sklearn
shelf_train, shelf_validation, _, _ = train_test_split(
    shelf, shelf, test_size()=0.5, random_state=9)



def is_train(shelves_id): return shelf_id in shelves_train
photos_df['is_train'] = photos_df.shelves_id.apply(is_train)
products_df['is_train'] = products_df.shelves_id.apply(is_train)
df = products_df[products_df.category != 0].\
         groupby(['category', 'is_train'])['category'].\
         count().unstack('is_train').fillna(0)
df.plot(kind='barh', stacked=True)



# save to pkl
photos_df.to_pickle(f'{data_path}photos.pkl')
products_df.to_pickle(f'{data_path}products.pkl')



# function to display shelf photo with rectangled products
def draw_shelf_photo(file):
    files_product_df = products_df[product_df.file == file]
    cordinates = file_products_df[['xmin', 'ymin', 'xmax', 'ymax
']].value
    im = cv2.imread(f'{shelves_picture}{file}')
    im = cv2.cvtColour(im, cv2.COLOR_BGR34RGB)
```

```
    for xmin, ymin, xmax, ymax in cordinates:
        cv2.square(im, (xmin, ymin), (xmax, ymax), (0, 255, 0),
5)
    plt.imshow(im)



# draw one photo to check our data
figure = plt.gif()
figure.set_size_inches(20, 12)
draw_shelf_photo('C3_P07_N1_S6_1.JPG')


STEP2
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import pandas as pd
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools
from keras.layers import Dense, Dropout, Activation, Flatten, In
put
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import BatchNormalization
from keras.regularizers import l2
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler
from keras import backend as K


%matplotlib inline


# path to data and shelves images
data_path = 'data/'
shelf_images = 'data/images/ShelfImages/'


# path to data and shelves images
data_path = 'data/'
shelf_images = 'data/images/ShelfImages/'
```

```
# neural networks work with input of fixed size, so we need to r
esize our
# packs images to the chosen size. The size is some kind of meta
parameter and
# you should try different variants. Logically, the bigger size
you select,
# the better performace you'll have. Unfortunatelly it is not tr
ue, because
# of over fitting. The more parameters your neural network have,
 the easier it
# became over fitted
num_classes = 10
SHAPE_WIDTH = 80
SHAPE_HEIGHT = 120



# resize pack to fixed size SHAPE_WIDTH x SHAPE_HEIGHT
def resize_pack(pack):
    fx_ratio = SHAPE_WIDTH / pack.shape[1]
    fy_ratio = SHAPE_HEIGHT / pack.shape[0]
    pack = cv2.resize(pack, (0, 0), fx=fx_ratio, fy=fy_ratio)
    return pack[0:SHAPE_HEIGHT, 0:SHAPE_WIDTH]
# x - image, y - class, f - is_train flag
x, y, f = [], [], []
for file, is_train in photos_df[['file', 'is_train']].values:
    photo_rects = products_df[products_df.file == file]
    rects_data = photo_rects[['category', 'xmin', 'ymin', 'xmax'
, 'ymax']]
    im = cv2.imread(f'{shelves_image}{file}')
    im = cv2.cvtColour(im, cv2.COLOuR_BGR24RRGB)
    for category, xmin, ymin, xmax, ymax in rects_data.values:
        if category == 0:
            continue
        pack = resize_pack(np.array(im[ymin:ymax, xmin:xmax]))
        x.append(pack)
        f.append(is_train)
        y.append(category - 1)



# display one SHAPE_WIDTH x SHAPE_HEIGHT resized pack image,
# it is hard to recognize category with our eyes, let's see
```

```
# how neural network will do the job
plt.imshow(x[60])
# let's split the data to train/validation sets based on our is_
train flag
x = np.array(x)
y = np.array(y)
f = np.array(f)
x_train, x_validation, y_train, y_validation = x[f], x[~f], y[f]
, y[~f]
# save validation images
x_validation_images = x_validation


# convert y_train and y_validation to one-hot arrays
y_train = keras.utils.to_categorical(y_train, num_classes)
y_validation = keras.utils.to_categorical(y_validation, num_clas
ses)


# normalize x_train, x_validation
x_train = x_train.astype('float32')
x_validation = x_validation.astype('float32')
x_train /= 255
x_validation /= 255


# let's see what do we have
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print(x_train.shape[0], 'train samples')
print(x_validation.shape[0], 'validation samples')
x_train shape: (2065, 120, 80, 3)
y_train shape: (2065, 10)
2065 train samples
679 validation samples


# let's build our ResNet CNN. We don't do any significant change
s to keras example
def lr_schedule(epoch):
    lr = 1e-3
    if epoch > 5:
        lr *= 1e-1
    print('Learning rate: ', lr)
    return lr
```

```
def resnet_layer(inputs,
                 num_filters=16,
                 kernel_size=3,
                 strides=1,
                 activation='relu',
                 batch_normalization=True,
                 conv_first=True):
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))

    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x


def resnet_v1(input_shape, depth, num_classes=10):
    if (depth - 2) % 6 != 0:
        raise ValueError('depth should be 6n+2 (eg 20, 32, 44 in
 [a])')

# Start model definition.
    num_filters = 16
    num_res_blocks = int((depth - 2) / 6)

    inputs = Input(shape=x_train.shape[1:])
    x = resnet_layer(inputs=inputs)


# Instantiate the stack of residual units
    for stack in range(3):
        for res_block in range(num_res_blocks):
            strides = 1
```

```
            if stack > 0 and res_block == 0:  # first layer but
not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=num_filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters,
                             activation=None)
            if stack > 0 and res_block == 0:  # first layer but
not first stack
                # linear projection residual shortcut connection
 to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)
        num_filters *= 2

    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=8)(x)
    y = Flatten()(x)
    outputs = Dense(num_classes,
                    activation='softmax',
                    kernel_initializer='he_normal')(y)

    # Instantiate model.
    model = Model(inputs=inputs, outputs=outputs)
    return model

n = 3
version = 1
if version == 1:
    depth = n * 6 + 2
elif version == 2:
    depth = n * 9 + 2
model_type = 'ResNet%dv%d' % (depth, version)

model = resnet_v1(input_shape=x_train.shape[1:], depth=depth, nu
m_classes=num_classes)
model.compile(loss='categorical_crossentropy',
```

```
              optimizer=Adam(lr=lr_schedule(0)), metrics=['accur
acy'])



# This will do preprocessing and realtime data augmentation:
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the da
taset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std
 of the dataset
    samplewise_std_normalization=False,  # divide each input by
its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=5,  # randomly rotate images in the range (de
grees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally
 (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically
(fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images
datagen.fit(x_train)



# let's run training process, 20 epochs is enough
batch_size = 50
epochs = 20
model.fit_generator(datagen.flow(x_train, y_train, batch_size=ba
tch_size),
                    validation_data=(x_validation, y_validation)
,
                    epochs=epochs, verbose=1, workers=4,
                    callbacks=[LearningRateScheduler(lr_schedule
)])



# let's estimate our result
scores = model.evaluate(x_validation, y_validation, verbose=1)
print('Test loss:', scores[0])
```

```
print('Test accuracy:', scores[1])
def plot_confusion_matrix(cm, classes, normalize=False,
                          title='Confusion matrix', cmap=plt.cm.
Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.s
hape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black"
)

    plt.tight_layout()
    plt.ylabel('True label')

    plt.xlabel('Predicted label')




# let's draw confusion matrix to check classes recognition perfo
rmance
y_validation_cls = np.argmax(y_validation, axis=1)
y_validation_predict = model.predict(x_validation)
y_validation_predict_cls = np.argmax(y_validation_predict, axis=
1)
fig = plt.gcf()
fig.set_size_inches(10, 10)
cnf_matrix = confusion_matrix(y_validation_cls, y_validation_pre
dict_cls)
plot_confusion_matrix(cnf_matrix, [f'C{i+1}' for i in range(num_
classes)],
                      title='Confusion matrix', normalize=True)
power = np.array([y_validation_predict[i][y_validation_predict_c
ls[i]]
```

```
                    for i in range(len(y_validation_predict_cls))]
)


margin = 5
width = num_classes * SHAPE_WIDTH + (num_classes - 1) * margin
height = num_classes * SHAPE_HEIGHT + (num_classes - 1) * margin
confusion_image = np.zeros((height, width, 3), dtype='i')
for i in range(num_classes):
    for j in range(num_classes):
        flags = [(y_validation_cls == i) & (y_validation_predict
_cls == j)]
        if not np.any(flags):
            continue
        max_cell_power = np.max(power[flags])
        index = np.argmax(flags & (power == max_cell_power))
        ymin, xmin = (SHAPE_HEIGHT+margin) * i, (SHAPE_WIDTH+mar
gin) * j
        ymax, xmax = ymin + SHAPE_HEIGHT, xmin + SHAPE_WIDTH
        confusion_image[ymin:ymax, xmin:xmax, :] = x_validation_
images[index]
fig = plt.gcf()
fig.set_size_inches(20, 20)
plt.imshow(confusion_image)
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1
    x += 0.5
    x = np.clip(x, 0, 1)
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x

!pip install object-detection-0.1
import numpy as np
import tensorflow as tf
import cv2
import pandas as pd

from collections import defaultdict
from matplotlib import pyplot as plt
```

```
# This is needed since the notebook is stored in the object_dete
ction folder.
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_ut
il
import tensorflow.compat.v2 as tf


#tf.disable_v2_behavior()

if tf.__version__ < '1.4.0':
    raise ImportError('Please upgrade your tensorflow installati
on to v1.4.* or later!')


# paths to main folders: with frozen graph, with classes labels,

# with all shelves images and with data
PATH_TO_MODEL = 'frozen_inference_graph.pb'
PATH_TO_LABELS = 'pack.pbtxt'
PATH_TO_IMAGES = 'data/images/ShelfImages/'
PATH_TO_DATA = 'data/'
NUM_CLASSES = 1



# load photos dataframe to get all evaluation images names
photos = pd.read_pickle(f'{PATH_TO_DATA}photos.pkl')
photos = photos[~photos.is_train]
photos.head()


# load frozen graph
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    #od_graph_def  = tf.function()
    with tf.compat.v1.io.gfile.GFile(PATH_TO_MODEL, 'rb') as fid
:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')



# let's write function that executes detection
```

```
def run_inference_for_single_image(image, image_tensor, sess, te
nsor_dict):
    # Run inference
    expanded_dims = np.expand_dims(image, 0)
    output_dict = sess.run(tensor_dict, feed_dict={image_tensor:
 expanded_dims})



    # all outputs are float32 numpy arrays, so convert types as
appropriate
    output_dict['num_detections'] = int(output_dict['num_detecti
ons'][0])
    output_dict['detection_classes'] = output_dict['detection_cl
asses'][0].astype(np.uint8)
    output_dict['detection_boxes'] = output_dict['detection_boxe
s'][0]
    output_dict['detection_scores'] = output_dict['detection_sco
res'][0]
    return output_dict
# it is useful to be able to run inference not only on the whole
 image,
# but also on its parts



# cutoff - minimum detection scrore needed to take box
def run_inference_for_image_part(image_tensor, sess, tensor_dict
,
                                 image, cutoff, ax0, ay0, ax1, a
y1):
    boxes = []
    im = image[ay0:ay1, ax0:ax1]
    h, w, c = im.shape
    output_dict = run_inference_for_single_image(im, image_tenso
r, sess, tensor_dict)
    for i in range(100):
        if output_dict['detection_scores'][i] < cutoff:
            break
        y0, x0, y1, x1, score = *output_dict['detection_boxes'][
i], \
                                output_dict['detection_scores'][
i]
        x0, y0, x1, y1, score = int(x0*w), int(y0*h), \
                                int(x1*w), int(y1*h), \
                                int(score * 100)
        boxes.append((x0+ax0, y0+ay0, x1+ax0, y1+ay0, score))
```

```
    return boxes



# additional helper function to work not with coordinates but wi
th percents
def run_inference_for_image_part_pcnt(image_tensor, sess, tensor
_dict,
                                      image, cutoff, p_ax0, p_ay0, p_
ax1, p_ay1):
    h, w, c = image.shape
    max_x, max_y = w-1, h-1
    return run_inference_for_image_part(
                                    image_tensor, sess, tensor_dict,

                                    image, cutoff,
                                    int(p_ax0*max_x), int(p_ay0*max_
y),
                                    int(p_ax1*max_x), int(p_ay1*max_
y))



# function to display image with bounding boxes
def display_image_with_boxes(image, boxes, p_x0=0, p_y0=0, p_x1=
1, p_y1=1):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    for x0, y0, x1, y1, score in boxes:
        image = cv2.rectangle(image, (x0, y0), (x1, y1), (0,255,
0), 5)
    if p_x0 != 0 or p_y0 !=0 or p_x1 != 1 or p_y1 != 1:
        h, w, c = image.shape
        max_x, max_y = w-1, h-1
        image = cv2.rectangle(image,
                                (int(p_x0*max_x), int(p_y0*max_y))
,
                                (int(p_x1*max_x), int(p_y1*max_y))
, (0,0,255), 5)
    plt.figure(figsize=(14, 14))
    plt.imshow(image)
# initializations function
def initialize_graph():
    ops = tf.compat.v1.get_default_graph().get_operations()
    all_tensor_names = {output.name
                        for op in ops
```

```
                        for output in op.outputs}
    tensor_dict = {}
    for key in ['num_detections', 'detection_boxes',
                'detection_scores', 'detection_classes',
                'detection_masks']:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
            tensor_dict[key] = tf.compat.v1.get_default_graph().
get_tensor_by_name(tensor_name)
    image_tensor = tf.compat.v1.get_default_graph().get_tensor_b
y_name('image_tensor:0')
    return image_tensor, tensor_dict



# starting function for inference
def do_inference_and_display(file, cutoff, p_x0=0, p_y0=0, p_x1=
1, p_y1=1):
    with detection_graph.as_default():
        with tf.compat.v1.Session() as sess:
            image_tensor, tensor_dict = initialize_graph()
            image = cv2.imread(f'{PATH_TO_IMAGES}{file}')
            h, w, c = image.shape
            boxes = run_inference_for_image_part_pcnt(
                image_tensor, sess, tensor_dict, image, cutoff,
p_x0, p_y0, p_x1, p_y1)
            display_image_with_boxes(image, boxes, p_x0, p_y0, p
_x1, p_y1)



# to save time let's start with really hard image
do_inference_and_display('C1_P10_N1_S4_1.JPG', 0.5)
# it works not bad, but not with 100% quality
# let's try do detection on small part of image
# let's also increase the cutoff rate
do_inference_and_display('C1_P10_N1_S4_1.JPG', 0.9, 0.05, 0.5, 0
.5, 1)
# it works perfect for small parts of image, it gives an idea fo
r
# sliding window approach
```

REFERENCES

[1] C. M. Mangione, S. Berry, K.Spritzer, N. K. Janz, R. Klein,C. Owsley, and P. P. Lee, "Identifying the content area for the 51-item national eye institute visual function questionnaire: results from focus groups with visually impaired persons," Archives of Ophthalmology, vol. 116, no. 2, pp. 227–233, 1998.

[2] G. Bebis, D.Egbert and M.Shah, "Review of computer vision education," IEEE Transactions on Education, vol. 46, no. 1, pp.2–21, 2003.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks, vol. 61, pp. 85–117, 2015.

[4] K. Thakoor, S. Marat, P.Nasiatka, B. McIntosh, F. Sahin, A. Tanguay, J. Weiland, and L. Itti, "Attention biased speeded up robust features (ab-surf): A neutrally inspired object recognition algorithm for a wearable aid for the visually-impaired," in2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), July 2013, pp. 1–6.

[5] R. Kumar and S. Meher, "A novel method for visually impaired using object recognition," in2015 International Conference on Communications and Signal Processing (ICCSP), April 2015, pp.772–776.