HUMAN ACTIVITY RECOGNITION USING CNN

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

KEERTHANA J R (Reg.No - 39110483)
KHARISHMA V (Reg.No – 39110497)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119

APRIL - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Keerthana J R (Reg.No -39110483) and Kharishma V (Reg.No - 39110497)** who carried out the Project Phase-2 entitled **"HUMAN ACTIVITY RECOGNITION USING CNN"** under my supervision from January 2023 to April 2023.

Bharathi

Internal Guide

Dr Bharathi B, M.E., Ph.D

[illegible]

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CSE

Submitted for Viva voce Examination held on 24.04.2023

[illegible]

Internal Examiner

J. Refonaa

External Examiner

DECLARATION

I, **Keerthana J R (Reg.No - 39110483),** hereby declare that the Project Phase-2 Report entitled **“HUMAN ACTIVITY RECOGNITION USING CNN”** done by me under the guidance of **Dr Bharathi B, M.E., Ph.D** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 24.04.2023 **Keerthana.J**

PLACE: Chennai **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.B.Ankayarkanni M.E.,Ph.D,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

In computer vision, activity recognition refers the manner of classifying an action that is present in the given video and action detection helps with locating actions of interest in space and time. To put it simply, the task of classifying or predicting the activity/action being performed by someone is called Action recognition. Human activities recognition has become a groundwork area of great interest because it has many significant and futuristic applications; including automated surveillance, Automated Vehicles, language interpretation and human computer interfaces (HCI). Traditional surveillance camera systems are mostly inefficient and expensive. Therefore, this project will provide the mandatory motivation for recognizing human action effectively in real-time (future work). In recent years, deep learning has been given particular attention by the computer vision community. This project presents an overview of the current state-of-the-art in action recognition using video analysis with deep learning techniques, along with the most important deep learning models for recognizing human actions, analyze them to provide the current progress of deep learning algorithms applied to solve human action recognition problems in realistic videos highlighting their advantages and disadvantages. Based on the quantitative analysis using recognition accuracies reported in the literature, this project identifies state-of-the-art deep architectures in action recognition and then provides current trends and open problems for future works in this filed. This project provides motivation for efficient human activity recognition by analyzing CCTV and camera images to detect and classify human activities using HAAR-based and Convolutional Neural Network (CNN) classifiers. It also identifies the best-performing deep architectures in action recognition and outlines future trends and open problems in this field. The current model aims to achieve high detection and recognition accuracy at approximately 22 frames per second after twenty epochs using a dataset of 5648 images.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, human activity recognition continues to be active research in the computer vision community due to the interest in the development of many intelligent systems involving surveillance, control, and analysis. The main goal of this area is to determine, and then predict what humans do in a video or a sequence of images. There are many potential applications such as intelligent surveillance systems, human-computer interfaces, health care, virtual reality, or security and military applications.

Human action recognition targets recognizing different actions from a sequence of observations and different environmental conditions. A wide range of applications is applicable to vision - based action recognition research Video analysis tasks have seen great variations and it has been moving from inferring the present state to predicting the future state.

Various human actions are inferred based upon the complete movements of that action. It also helps in prediction of future state of the human by inferring the current action being performed by that human. It has become a hot topic in recent years since it applies to real world problems directly. Many research works have been done in this field to develop a good human action recognizer. Also, it is expected that more and more work is going to be done in this regard since Human Action Recognition finds its use in numerous applications.

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand. Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image.
Computer vision trains machines to perform these functions, but it has to do it in

much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.
Computer vision is used in industries ranging from energy and utilities to manufacturing and automotive – and the market is continuing to grow. It is expected to reach USD 48.6 billion by 2022.

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN).Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will “look” at the data and teach itself to tell one image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.
A CNN helps a machine learning or deep learning model “look” by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is “seeing.” The neural network runs convolutions and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

Much like a human making out an image at a distance, a CNN first discerns hard edges and simple shapes, then fills in information as it runs iterations of its predictions. A CNN is used to understand single images. A recurrent neural network (RNN) is used in a similar way for video applications to help computers understand how pictures in a series of frames are related to one another. Here are a few examples of established computer vision tasks:

Image classification sees an image and can classify it (a dog, an apple, a person's face). More precisely, it is able to accurately predict that a given image belongs to a certain class. For example, a social media company might want to use it to automatically identify and segregate objectionable images uploaded by users.

Object detection can use image classification to identify a certain class of image and then detect and tabulate their appearance in an image or video. Examples include detecting damages on an assembly line or identifying machinery that requires maintenance.

Object tracking follows or tracks an object once it is detected. This task is often executed with images captured in sequence or real-time video feeds. Autonomous vehicles, for example, need to not only classify and detect objects such as pedestrians, other cars and road infrastructure, they need to track them in motion to avoid collisions and obey traffic laws.

Content-based image retrieval uses computer vision to browse, search and retrieve images from large data stores, based on the content of the images rather than metadata tags associated with them. This task can incorporate automatic image annotation that replaces manual image tagging. These tasks can be used for digital asset management systems and can increase the accuracy of search and retrieval.

With the emergence of wearable cameras (e.g., GoPro and Google Glass), first person activity recognition has also been of interest to many in the computer vision community. First and third-person action recognition algorithms are two very closely related tasks. However, there is a significant difference between the two. First-person action recognition involves determining the action executed by the person wearing the camera from an egocentric viewpoint. Third-person action recognition, on the other hand, involves determining the action executed by a person as captured by someone other than the actor. This difference results in contrasting datasets, actions of interest, and viewpoints.

CHAPTER 2
LITERATURE SURVEY

This section considers publicly available datasets and related surveys in human activity recognition. Referring to the major conferences and journals and even several earlier surveys have been published.

Aggarwal and Cai [6] in this paper on Human Motion Analysis (1997) gives an overview of the various tasks involved in motion analysis of the human body. They focus on three major areas related to interpreting human motion: (1) motion analysis involving human body parts, (2) tracking a moving human from a single view or multiple camera perspectives, and (3) recognizing human activities from image sequences. Motion analysis of human body parts involves the low-level segmentation of the human body into segments connected by joints and recovers the 3D structure of the human body using its 2D projections over a sequence of images. Tracking human motion from a single view or multiple perspectives focuses on higher-level processing, in which moving humans are tracked without identifying their body parts. After successfully matching the moving human image from one frame to another in an image sequence, understanding the human movements or activities comes naturally, which leads to their discussion of recognizing human activities.

Moeslund and granum [9] in this paper on Human Body Model Acquisition and tracking (2001) presents an integrated system for automatic acquisition of the human body model and motion tracking using input from multiple synchronized video streams. The video frames are segmented and the 3D voxel reconstructions of the human body shape in each frame are computed from the foreground silhouettes. These reconstructions were then used as input to the model acquisition and tracking algorithms. The human body model consists of ellipsoids and cylinders and was described using the twists framework resulting in a non-redundant set of model parameters. Model acquisition starts with a simple body part localization procedure based on template fitting and growing, which uses prior knowledge of average body part shapes and dimensions. The initial model was then refined using a Bayesian network that imposes human body proportions onto the body part size estimates. The tracker was an extended Kalman filter that estimates model parameters based on the measurements made on the labelled voxel data. A voxel labelling procedure that handles large frame-to-frame displacements was designed resulting in very robust tracking performance. Extensive evaluation showed that the system performs very reliably on sequences that include different types of motion such as walking, sitting, dancing, running and jumping and people of very different body sizes, from a small girl (nine years old) to a tall adult male.

Turaga et al [7] in this paper on Machine Recognition of Human Activities: A Survey (2008) has witnessed a rapid proliferation of video cameras in all walks of life and has resulted in a tremendous explosion of video content. Several applications such as content-based video annotation and retrieval, highlight extraction and video summarization require recognition of the activities occurring in the video. The analysis of human activities in videos was an area with increasingly important consequences from security and surveillance to entertainment and personal archiving. Several challenges at various levels of processing-robustness against errors in low-level processing, view and rate-invariant representations at midlevel processing and semantic representation of human activities at higher level processing-make this problem hard to solve. In this review paper, they presented a comprehensive survey of efforts in the past couple of decades to address the problems of representation, recognition, and learning of human activities from video and related applications. They discussed the problem at two major levels of complexity: 1) "actions" and 2) "activities." "Actions" are characterized by simple motion patterns typically executed by a single human. "Activities" were more complex and involve coordinated actions among a small number of humans. They have discussed several approaches and classify them according to their ability to handle varying degrees of complexity as interpreted above. They begin with a discussion of approaches to model the simplest of action classes known as atomic or primitive actions that do not require sophisticated dynamical modeling. Then, methods to model actions with more complex dynamics are discussed. The discussion then leads naturally to methods for higher level representation of complex activities.

R Poppe [8] in this paper on A Survey on vision-based human action recognition (2010) reviewed vision-based human action recognition is the process of labeling image sequences with action labels. Robust solutions to this problem have applications in domains such as visual surveillance, video retrieval and human–computer interaction. The task was challenging due to variations in motion performance, recording settings and inter-personal differences. In this survey, they explicitly addressed these challenges. They provided a detailed overview of current advances in the field. Image representations and the subsequent classification process are discussed separately to focus on the novelties of recent research. Moreover, they discussed limitations of the state of the art and outline promising directions of research.

Weinland et al [3] in this paper on A Survey of vision-based for action representation, segmentation and recognition (2011) reviewed that Action recognition has become a very important topic in computer vision, with many fundamental applications, in robotics, video surveillance, human-computer interaction, and multimedia retrieval among others and a large variety of approaches have been described. The purpose of this survey is to give an overview and categorization of the approaches used. They concentrated on approaches that aim on classification of full-body motions, such as kicking, punching, and waving, and they categorized them according to how they represent the spatial and temporal structure of actions; how they segment actions from an input stream of visual data; and how they learn a view-invariant representation of actions.

Andrej Karpathy [1] in this paper on Large-Scale Video Classification with Convolutional Neural Networks (2014) reviewed that Convolutional Neural Networks (CNNs) have been established as a powerful class of models for image recognition problems. Encouraged by these results, they provided an extensive empirical evaluation of CNNs on large-scale video classification using a new dataset of 1 million YouTube videos belonging to 487 classes. They studied multiple approaches for extending the connectivity of a CNN in time domain to take advantage of local spatio-temporal information and suggest a multiresolution, foveated architecture as a promising way of speeding up the training. Their best spatio-temporal networks display significant performance improvements compared to strong feature-based baselines (55.3% to 63.9%), but only a surprisingly modest improvement compared to single-frame models (59.3% to 60.9%). They further studied the generalization performance of this best model by retraining the top layers on the UCF-101 Action Recognition dataset and observe significant performance improvements compared to the UCF-101 baseline model (63.3% up from 43.9%).

Cheng et al [4] in this paper on Advances in Human Action Recognition: A Survey (2015) Human action recognition has been an important topic in computer vision due to its many applications such as video surveillance, human machine interaction and video retrieval. One core problem behind these applications is automatically recognizing low-level actions and high-level activities of interest. The former was usually the basis for the latter. This survey gives an overview of the most recent advances in human action recognition during the past several years, following a well-formed taxonomy proposed by a previous survey. From this state-of-the-art survey, researchers can view a panorama of progress in this area for future research.

Yu-Wei chao [10] in this paper on Rethinking the faster R-CNN Architecture for temporal action Localization (2018) propose TAL-Net, an improved approach to temporal action localization in video that was inspired by the Faster R-CNN object detection framework. TAL-Net addresses three key shortcomings of existing approaches: (1) they improve receptive field alignment using a multi-scale architecture that can accommodate extreme variation in action durations; (2) they better exploit the temporal context of actions for both proposal generation and action classification by appropriately extending receptive fields; and (3) they explicitly consider multi-stream feature fusion and demonstrate that fusing motion late was important. They achieved state-of-the-art performance for both action proposal and localization on THUMOS'14 detection benchmark and competitive performance on ActivityNet challenge.

Feichtenhofer [2] in their paper on SlowFast Networks for Video Rcognition (2019) present SlowFast networks for video recognition. This model involves (i) a Slow pathway, operating at low frame rate, to capture spatial semantics, and (ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution. The Fast pathway can be made very lightweight by reducing its channel capacity, yet can learn useful temporal information for video recognition. This model achieve strong performance for both action classification and detection in video, and large improvements are pin-pointed as contributions by the SlowFast concept. They reported state-of-the-art accuracy on major video recognition benchmarks, Kinetics, Charades and AVA.

2.1 INFERENCES FROM LITERATURE SURVEY

The literature review reveals that there is a gap in the current research on human activity recognition using single-frame CNNs. While there are existing surveys on the topic, there is no reliable review that specifically focuses on the use of single-frame CNNs for human activity recognition. This highlights the need for further research in this area to evaluate the effectiveness of such approaches.

The field of human motion analysis and recognition has been a topic of interest for researchers for several decades, and many different approaches have been proposed to tackle this complex problem. Researchers have focused on tasks such as motion analysis involving human body parts, tracking a moving human, and recognizing human activities from image sequences. Various challenges exist in this field such as robustness against errors in low-level processing, view and rate-invariant representations at mid-level processing, and semantic representation of human activities at higher level processing. Researchers have explored different techniques such as Bayesian network, extended Kalman filter, and convolutional neural networks to address these challenges.

Additionally, the field of human activity recognition is rapidly evolving, with new algorithms and techniques emerging at a rapid pace. Therefore, it is important to provide a comparative analysis of the current state of the field to better understand the strengths and weaknesses of existing approaches. This analysis can also help to identify new trends and directions in the field, which can guide future research. Overall, conducting a comprehensive review of the literature is important to identify gaps and limitations in the current state of research, and to suggest future research directions that can help to address these gaps and push the field forward.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

The challenge of recognizing complex actions and behaviors in realistic scenarios using computer vision is an ongoing issue in the field of activity recognition. A major limitation is the potential for a model to predict an action incorrectly based on just a random snapshot from a video clip. This issue needs to be addressed, as it has implications for the accuracy and reliability of activity recognition systems.
In addition, there is a need to develop applications with acceptable computational and memory requirements. This means that models must be designed to be computationally efficient and scalable. The state-of-the-art datasets for human activity recognition include UCF, HMDB-51, Sports-1M, ActivityNet, and NTU RGB+D. These datasets provide a benchmark for evaluating the performance of activity recognition models. RGB-D and skeleton data can also provide a better understanding of the 3D structure of human body movements, which can be used to improve the accuracy of activity recognition systems.

To date, various deep learning architectures have been proposed and have produced state-of-the-art results on many tasks, including human activity recognition. The development of new deep learning models and techniques is crucial for improving the accuracy and robustness of activity recognition systems.

CHAPTER 3
REQUIREMENT ANALYSIS

The software and hardware requirements for a human activity recognition system using a CNN model will depend on the specific application and the complexity of the CNN model. Here are some general requirements that may be needed:

3.1 FEASIBILITY STUDY

Feasibility study is an important aspect of any machine learning model, including a CNN-based model for human activity recognition. It involves identifying potential risks associated with the model's deployment, assessing the likelihood and impact of these risks, and developing strategies to mitigate them.

3.1.1 Risk Analysis

Here are some potential risks that could be associated with a CNN-based model for human activity recognition:

Privacy: The use of sensor data to recognize human activities raises privacy concerns, as this data can reveal sensitive information about an individual's daily routine and habits. To mitigate this risk, appropriate measures must be taken to protect the privacy of individuals, such as anonymizing data and limiting access to sensitive information.

Safety: If the model is deployed in safety-critical applications such as fall detection, a failure to accurately recognize an activity or detect a fall could result in serious harm to the individual. To mitigate this risk, the model must be rigorously tested and validated on a diverse range of activities and scenarios to ensure that it can reliably detect falls and other safety-critical activities.

Bias: CNN-based models can be susceptible to bias if the training data is not diverse or representative of the population. This can result in unfair or discriminatory outcomes, particularly if the model is used in decision-making processes. To mitigate this risk, it is important to ensure that the training data is diverse and representative of the population, and that the model is regularly monitored for bias.

Robustness: CNN-based models can be vulnerable to adversarial attacks, where malicious actors attempt to manipulate the input data to deceive the model. To mitigate this risk, the model must be designed to be robust to such attacks, for example, by incorporating techniques such as input preprocessing and defensive distillation.

Overall, a study for a CNN-based model for human activity recognition should consider a wide range of potential risks and develop appropriate mitigation strategies to ensure the safe and effective deployment of the model.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION

3.2.1 Software requirements:

Programming languages: Python is a popular language for building deep learning models and may be used for building the human activity recognition system.

Deep learning libraries: A deep learning library such as TensorFlow or PyTorch may be used for building and training the CNN model.

Data processing libraries: Libraries such as NumPy, Pandas, and OpenCV may be used for preprocessing and analyzing the sensor data.

***Annotation tools*:** Software tools such as Labelbox or VGG Image Annotator may be used for labeling the collected sensor data.

Visualization tools: Visualization tools such as Matplotlib or TensorBoard may be used for visualizing the performance of the CNN model.

3.2.2 Hardware requirements:

Processing power: Building and training a CNN model can be computationally intensive, requiring powerful CPUs and GPUs. A high-end GPU such as NVIDIA's GeForce RTX or Tesla may be required for efficient training of the CNN model.

Memory: Deep learning models require large amounts of memory to store the model parameters and intermediate computations. Sufficient RAM and disk space are required to handle the data and model parameters.

Network connectivity: If the human activity recognition system is deployed in a distributed or cloud-based environment, network connectivity with sufficient bandwidth and low latency may be required.

The software and hardware requirements for a human activity recognition system using a CNN model may include programming languages, deep learning and data processing libraries, annotation and visualization tools, processing power, memory, sensors, wearable devices or smartphones, and network connectivity. The specific requirements may vary based on the complexity of the CNN model and the application's specific needs.

3.3 System use case

Here are some use cases for human activity recognition for video classification using CNN:

Security and surveillance: Human activity recognition using CNN can be used for video surveillance systems to detect and classify suspicious activities such as trespassing, theft, or vandalism. A camera can capture video footage, and a CNN model can classify the activity and alert security personnel.

Traffic management: Human activity recognition using CNN can be used to monitor traffic and classify different types of vehicles such as cars, buses, or trucks. This can help traffic authorities to manage congestion and plan infrastructure improvements.

Sports analysis: Human activity recognition using CNN can be used in sports to analyze game footage and classify different types of movements such as passing, shooting, or dribbling. This can help coaches and analysts to understand player performance and identify areas for improvement.

Retail analytics: Human activity recognition using CNN can be used in retail to analyze customer behavior and classify different types of activities such as browsing, selecting, or purchasing. This can help retailers to optimize store layouts and improve customer experience.

Medical diagnosis: Human activity recognition using CNN can be used in medical diagnosis to analyze patient movements and classify different types of activities such as walking, running, or climbing stairs. This can help healthcare providers to diagnose and monitor conditions such as Parkinson's disease, multiple sclerosis, or stroke.

CHAPTER 4

DESCRIPTION OF CURRENT SYSTEM

4.1 SELECTED METHODOLOGY

The first step is to gather and preprocess the data. For image analysis, this includes tasks such as resizing the images, converting them to grayscale or RGB format, and normalizing the pixel values. The data is then split into training, validation, and testing sets.

The next step is to design the CNN architecture. This involves deciding on the number and type of convolutional layers, activation functions, pooling layers, and fully connected layers. The architecture can be customized based on the complexity of the problem and the amount of available training data.

After designing the architecture, the next step is to compile the model by selecting a loss function, an optimizer, and performance metrics. The loss function measures the difference between the predicted and actual outputs, while the optimizer updates the weights of the network to minimize the loss. The metrics are used to evaluate the performance of the model during training.

Once the model is compiled, it is trained on the training data. During training, the weights of the network are updated based on the backpropagation algorithm and the chosen optimizer. The model is evaluated on the validation set to monitor its performance and prevent overfitting.

Finally, the trained model is evaluated on the testing set to measure its accuracy and generalization performance. If the model does not perform well, the architecture can be modified or the hyperparameters can be tuned and the training process can be repeated until satisfactory results are obtained.

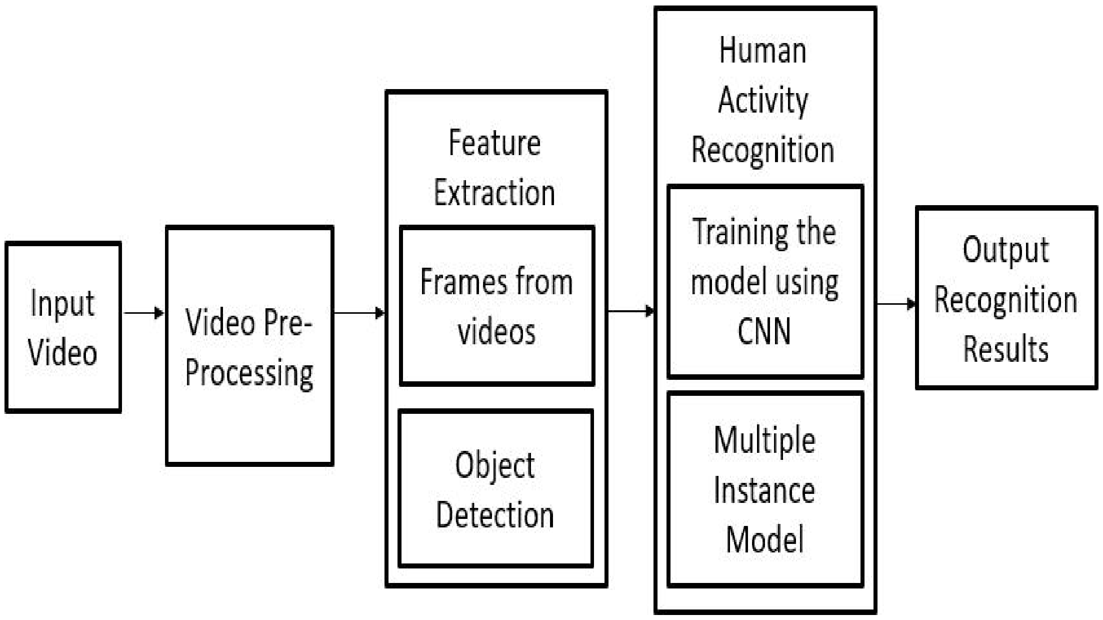


Fig 4.1– Overall design of Current system

4.2 OVERALL DESIGN OF CNN

The overall design of a Convolutional Neural Network (CNN) typically involves several key components:

Convolutional Layers: These layers perform convolutional operations on the input data, applying a set of filters or kernels to extract local features and create a set of feature maps. These layers are the core of the CNN architecture, responsible for learning spatial patterns in the input data.

Pooling Layers: These layers downsample the feature maps by aggregating neighboring values, reducing the dimensionality of the data while retaining important features. Max-pooling and average-pooling are common types of pooling operations used in CNNs.

Activation Functions: These functions are applied to the output of convolutional and pooling layers to introduce non-linearity and enable the network to learn complex, non-linear relationships between the input and output.

Fully Connected Layers: These layers connect all neurons in one layer to all neurons in the next layer, allowing the network to learn higher-level representations of the input data. These layers are typically used at the end of the network, and may be preceded by one or more convolutional and pooling layers.
Dropout Layers: These layers randomly drop out a subset of the neurons during training, helping to prevent overfitting and improve the generalization performance of the network.

Loss Functions: These functions measure the error between the predicted output of the network and the true output, and are used to guide the optimization process during training. Common loss functions include cross-entropy and mean squared error.

Overall, the design of a CNN involves a sequence of these components, each of which is designed to learn increasingly complex representations of the input data, leading to more accurate predictions. The specific architecture and hyperparameters of a CNN will depend on the particular task and dataset being used.

Here is an overall design image of a Convolutional Neural Network (CNN) (Fig 4.2):

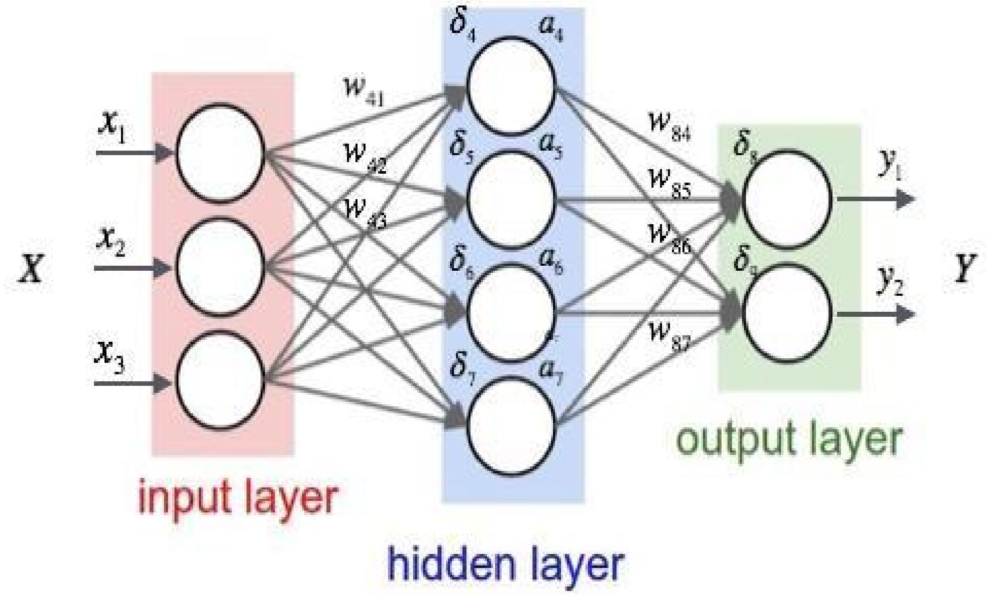


Fig 4.2 – CNN Design

The image shows a typical CNN architecture consisting of multiple convolutional and pooling layers, followed by one or more fully connected layers. The input is fed into the first convolutional layer, where it is convolved with a set of learnable filters. The output of the convolutional layer is then passed through a non-linear activation function, such as ReLU, before being fed into a pooling layer. The pooling layer reduces the spatial size of the representation while retaining the important features. This process of convolution and pooling is repeated several times, with each subsequent layer learning more complex features. The output of the final convolutional layer is then flattened into a vector and passed through one or more fully connected layers, which are responsible for making the final classification decision. The final layer usually employs a softmax activation function to produce a probability distribution over the different classes.

Forward propagation

Forward propagation is where input data is fed through a network, in a forward direction, to generate an output. The data is accepted by hidden layers and processed, as per the activation function, and moves to the successive layer.

Convolutional Layers

Suppose they have some N×N square neuron layer which is followed by the convolutional layer. If used an m×m filter ω, convolutional layer output will be of size (N−m+1)×(N−m+1). In order to compute the pre-nonlinearity input to some unit x^{ℓ}_{ij} in layer, need to sum up the contributions (weighted by the filter components) from the previous layer cells:

$x^{\ell}_{ij}=\sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\omega_{ab}y^{\ell-1}_{(i+a)(j+b)}$.

This is just a convolution, which can be express in Matlab via

conv2(x, w, 'valid')

Then, the convolutional layer applies its nonlinearity:

$$y^{\ell}_{ij}=\sigma(x^{\ell}_{ij}).$$

Max-Pooling Layers

The max-pooling layers are quite simple, and do no learning themselves. They simply take some k×k region and output a single value, which is the maximum in that region. For instance, if their input layer is a N×N layer, they will then output a Nk×Nk layer, as each k×k block is reduced to just a single value via the max function.

Backward propagation

The process of moving from the right to left i.e backward from the Output to the Input layer is called the Backward Propagation. Backward Propagation is the preferable method of adjusting or correcting the weights to reach the minimized loss function.
Convolutional Layers

Assume some error function, E , and know the error values in convolutional layer. What, then, are the error values at the layer before it, and what is the gradient for each weight in the convolutional layer? Note that the error known and that need to compute for the previous layer is the partial of E with respect to each neuron output $\left(\frac{\partial E}{\partial y_{ij}^{\ell}}\right)$. Let's first figure out what the gradient component is for each weight by applying the chain rule. Note that in the chain rule, must sum the contributions of all expressions in which the variable occurs.

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} \frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} y_{(i+a)(j+b)}^{\ell-1}$$

In this case, sum over all xℓij expressions in which ω ab occurs. (This corresponds to weight-sharing in the neural network!) Note that $\frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{\ell-1}$, just by looking at the forward propagation equations. In order to compute the gradient, know the values $\left(\frac{\partial E}{\partial y_{ij}^{\ell}}\right)$ (which are often called "deltas"). The deltas are fairly straightforward to compute, once more using the chain rule:

$$\frac{\partial E}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial y_{ij}^{\ell}}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial}{\partial x_{ij}^{\ell}} \left(\sigma(x_{ij}^{\ell})\right) = \frac{\partial E}{\partial y_{ij}^{\ell}} \sigma'(x_{ij}^{\ell})$$

As a known fact, the error at the current layer $\left(\frac{\partial E}{\partial y_{ij}^{\ell}}\right)$, can very easily compute the deltas $\left(\frac{\partial E}{\partial y_{ij}^{\ell}}\right)$at the current layer by just using the derivative of the activation function, σ′(x). Since the errors at the current layer, now have everything has to compute the gradient with respect to the weights used by this convolutional layer. In addition to compute the weights for this convolutional layer, need to propagate errors back to the previous layer. Once more use the chain rule:

$$\frac{\partial E}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \omega_{ab}$$

Looking back at the forward propagation equations, can tell that $\frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \omega_{ab}$. This gives us the above value for the error at the previous layer. It looks slightly like a convolution! It has specific filter ω being applied somehow to the layer; however, instead of having x(i+a)(j+b) it has x(i−a)(j−b) . In addition, note that the expression above only makes sense for points that are at least m away from the top and left edges. In order to fix this, it must pad the top and left edges with zeros. If that is done, then this is simply a convolution using ω which has been flipped along both axes!

Max-Pooling Layers

As noted earlier, the max-pooling layers do not actually do any learning themselves. Instead, then reduce the size of the problem by introducing sparseness. In forward propagation, k×k blocks are reduced to a single value. Then, this single value acquires an error computed from backwards propagation from the previous layer.

This error is then just forwarded to the place where it came from. Since it only came from one place in the k×k block, the backpropagated errors from max-pooling layers are rather sparse.

4.3 OVERALL DESIGN OF CURRENT SYSTEM

Computer vision is a branch within computer science that enables computers to recognize the visual world. Several machine learning- and deep learning-based algorithms are available that help with building models to make predictions on images or videos. This article explores convolutional neural networks (CNN), a type of supervised deep learning algorithm.

CNN is a type of deep learning algorithm that is widely used for image and video analysis tasks, such as image classification, object detection, and image segmentation. The key characteristic of CNNs is their ability to learn spatial hierarchies of features from raw input data. A typical CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

A convolutional neural network is an extension of artificial neural networks (ANN) and is predominantly used for image recognition-based tasks. A previous article covered different types of architectures that are built on artificial neural networks . This article explains the different steps that go into creating a convolutional neural network. These steps are:

1. Image channels
2. Convolution
3. Pooling
4. Flattening
5. Full connection

Image Channels

The first step in the process of making an image compatible with the CNN algorithm is to find a way to represent the image in a numerical format. The image is represented (Fig 4.3) using its pixel. Each pixel within the image is mapped to a number between 0 and 255. Each number represents a color ranging between 0 for white and 255 for black.

For a black and white image, an image with length *m* and width *n* is represented as a 2-dimensional array of size *mXn*. Each cell within this array contains its corresponding pixel value. In the case of a colored image of the same size, a 3-dimensional array is used. Each pixel from the image is represented by its corresponding pixel values in three different channels, each pertaining to a red, blue, and green channel.

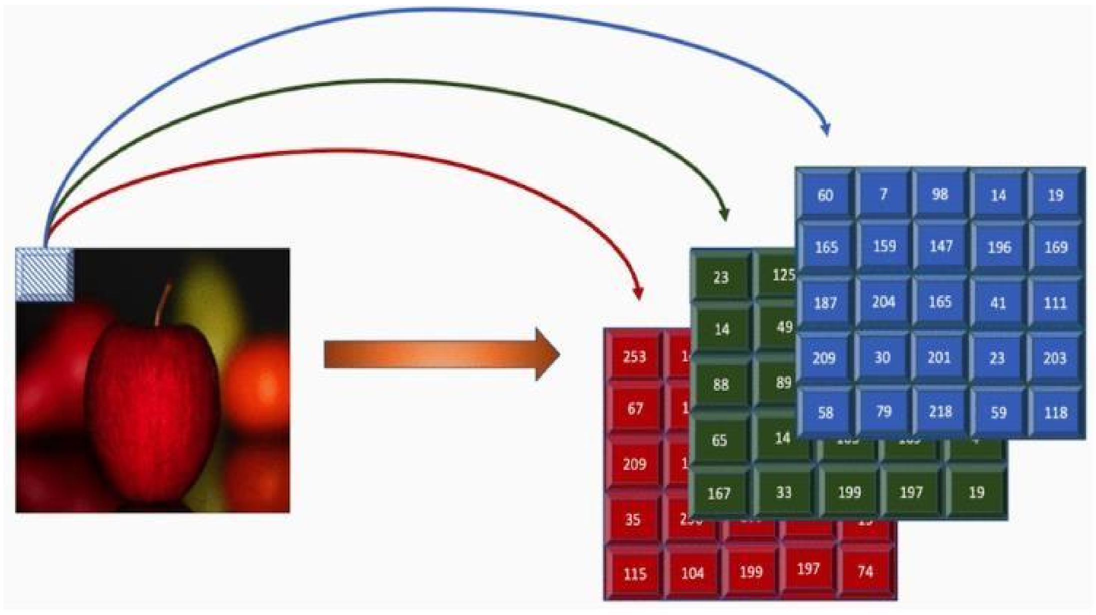
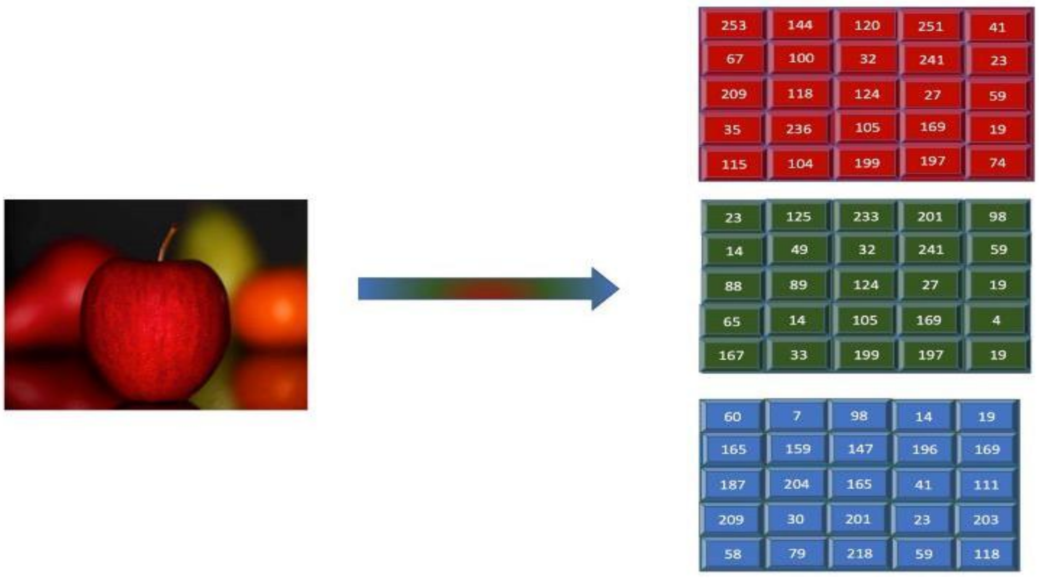


Fig 4.3 – Representation of pixels in three different channels[11]

Fig 4.3 is represented as a 3-dimensional array, with each channel representing red, green, and blue values, respectively, as shown in the following image.



253	144	120	251	41
67	100	32	241	23
209	118	124	27	59
35	236	105	169	19
115	104	199	197	74

23	125	233	201	98
14	49	32	241	59
88	89	124	27	19
65	14	105	169	4
167	33	199	197	19

60	7	98	14	19
165	159	147	196	169
187	204	165	41	111
209	30	201	23	203
58	79	218	59	118

Fig 4.4 – Values in each channel [11]

Convolution

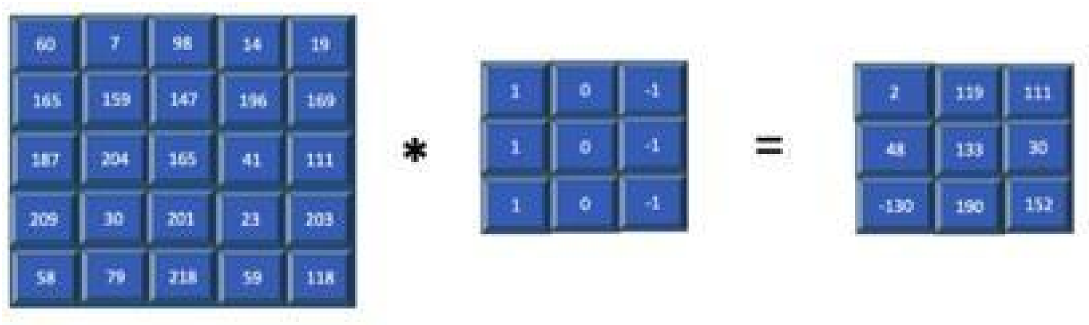
Now that the image (Fig 4.4) has been represented as a combination of numbers. the next step in the process is to identify the key features within the image. This is extracted using a method known as convolution.

Convolution is an operation where one function modifies (or convolves) the shape of another. Convolutions in images are generally applied for various reasons such as to sharpen, smooth, and intensify. In CNN, convolutions are applied to extract the prominent features within the images.

How are features detected

To extract key features within an image, a filter or a kernel is used. A filter is an array that represents the feature to be extracted. This filter is strided over the input array, and the resulting convolution is a 2-dimensional array that contains the correlation of the image with respect to the filter that was applied. The output array is referred to as the *feature map*.

For simplicity, the following Fig 4.5, shows how an edge detector filter is applied to just the blue channel output from the previous step. Later, filters are applied to all three channels.



$$\begin{bmatrix} 60 & 7 & 98 & 14 & 19 \\ 165 & 159 & 147 & 196 & 169 \\ 187 & 204 & 165 & 41 & 111 \\ 209 & 30 & 201 & 23 & 203 \\ 58 & 79 & 218 & 59 & 118 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 119 & 111 \\ 48 & 133 & 30 \\ -130 & 190 & 152 \end{bmatrix}$$

Fig 4.5 – Application of filters [11]

The resulting image contains just the edges present in the original input. The filter used in the previous example is of size *3x3* and is applied to the input image of size *5x5*. The resulting feature map is of size *3x3*. In summary, for an input image of size *nXn* and a filter of size *mXm*, the resulting output is of size *(n-m+1)X(n-m+1)*.

Strided convolutions

During the process of convolution, the input array is transformed into a smaller array while still maintaining the spatial correlation between the pixels by applying filters. Here, it discuss an option on how to help compress the size of the input array much more.

What is striding

In the previous section, the filter is applied to each 3x3 section of the input image. The window is slid by one column to the right each time, and the end of each row is slid down by one row. In this case, sliding of the filter over the input was done one step at a time. This is referred to as striding. The following example (Fig 4.6) shows the same convolution, but strided with 2 steps.

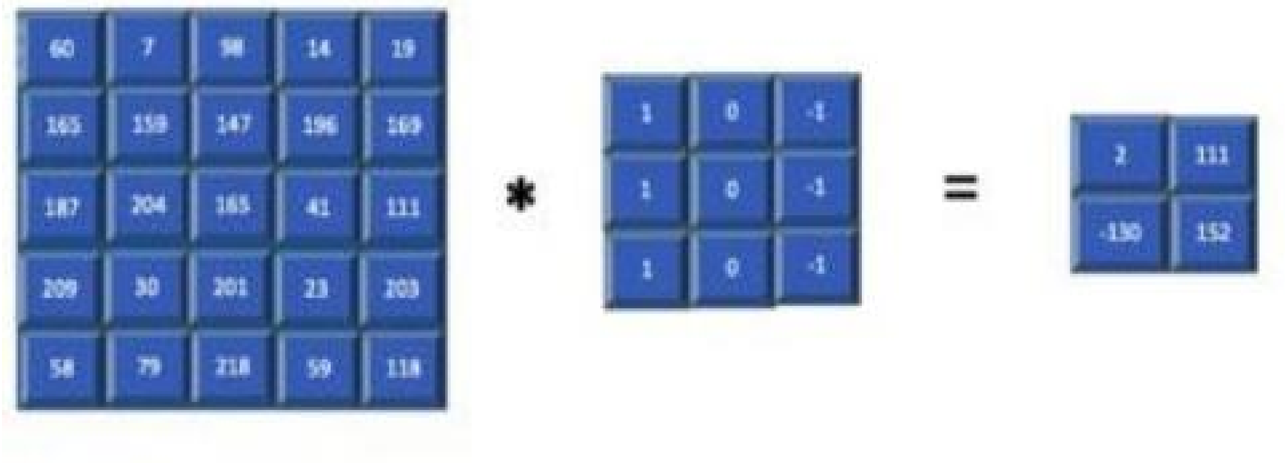


Fig 4.6 – Strided with two steps [11]

The filter used in the previous example is of size *3x3* and is applied to the input image of size *5x5* with a stride=2. The resulting feature map is of size *2x2*. In summary, for an input image of size *nXn* and a filter of size *mXm* with stride=k, the resulting output will be of size *((n-m)/k+1)X((n-m)/k+1)*.

Padding

During convolution, notice that the size of the feature map is reduced drastically when compared to the input. Also, notice that the filter stride touches the cells in the corners just once, but the cells to the center are filtered quite a few times.

To ensure that the size of the feature map retains its original input size and enables equal assessment of all pixels, apply one or more layers of padding to the original input array. Padding refers to the process of adding extra layers of zeros to the outer rows and columns of the input array.

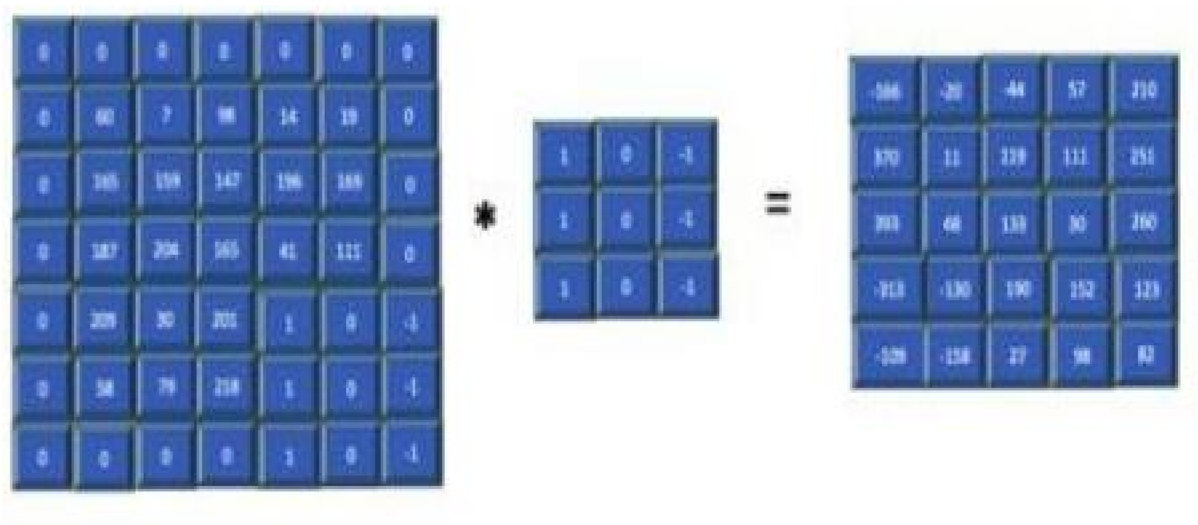


Fig 4.7 – Padding [11]

The previous image (Fig 4.7) shows how 1 layer of padding is added to the input array before a filter is applied. An input array of size *5x5*, padding is set to one, filter is size *3x3*, and the output is a *5x5* array. In general, for an input image of size *nXn* and a filter of size *mXm* with padding = p, the resulting output is of size *(n + 2p - m +1)X(n + 2p - m +1)*.

How are convolutions applied over the RGB channels

So far, the convolution operation looks if images were represented with just one channel. The blue channel from the first step used to walk through an example. Now, let's look at Fig 4.8, how convolution looks when a color image over 3 channels is used.

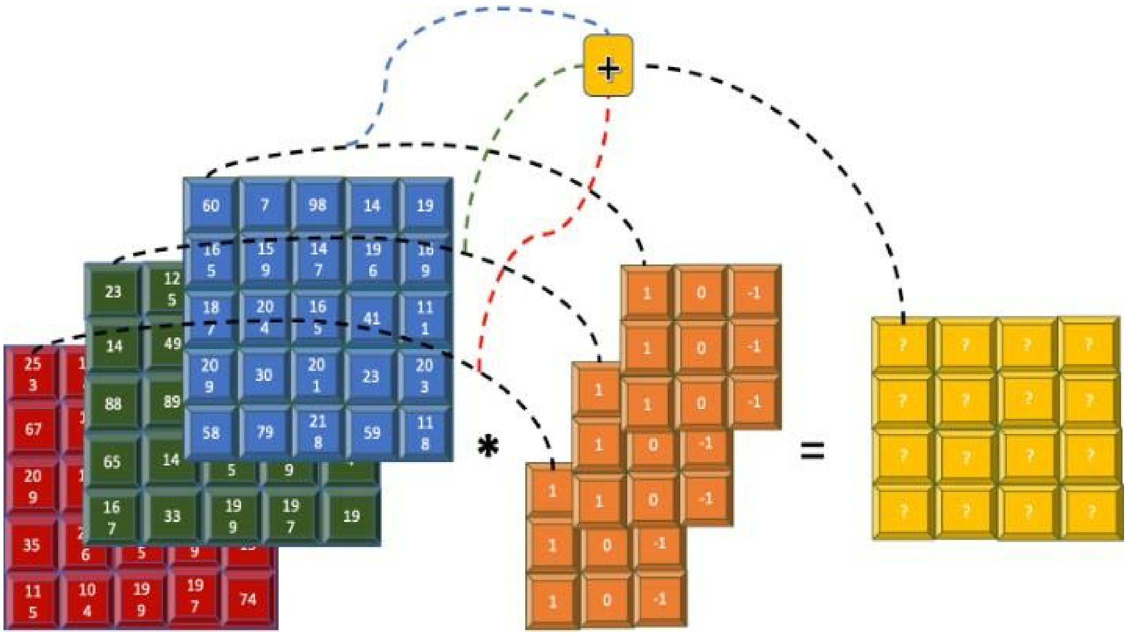


Fig 4.8 – Convolution applied over RGB channel [11]

For a *5x5* image represented over 3 channels, the *3x3* filter is now replicated three times, once for each channel. The input image is a *5x5x3* array, and the filter is a *3x3x3* array. However, the output map is still a 2D *4x4* array. The convolutions on the same pixel through the different channel are added and are collectively represented within each cell.

In general, for an input image of size *nXn* and filter of size *mXm* over N channel, the image and filters are converted into arrays of sizes *nXnXN* and *mXmXN*, respectively, and the feature map produced is of size *(n-m+1) X (n-m+1)* assuming stride=1.

How are convolutions applied to more than one filter

In the previous section, the output of applying convolution over multiple channels looks like. In doing that, take an edge filter and calculated the convolutions using it. However, this was using just one filter. In reality, the CNN model needs to use multiple filters at the same time to observe and extract key features. Visualize what the output to using multiple filters looks like.

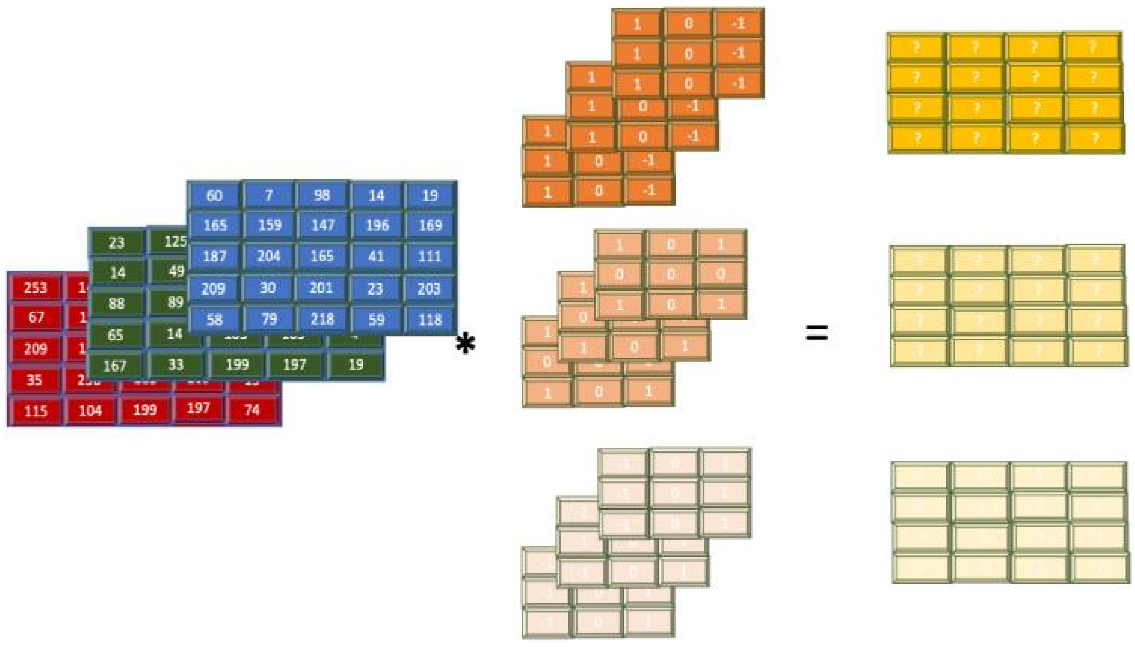


Fig 4.9 – Application of more than one filter [11]

In the Fig 4.9, applying convolution using three filters over the RGB channels produces three *4x4* arrays. Thus, for an input image of size *nXn* and filter of size *mXm* over *N* channel and *F* filters, the feature map produced is of size *(n-m+1)X(n-m+1)XF* assuming that stride=1.

Pooling layers

To further reduce the size of the feature map generated from convolution, apply pooling before further processing. This helps to further compress the dimensions of the feature map. For this reason, pooling is also referred to as *subsampling*.

Pooling is the process of summarizing the features within a group of cells in the feature map. This summary of cells can be acquired by taking the maximum, minimum, or average within a group of cells. Each of these methods is referred to as min, max, and average pooling, respectively.

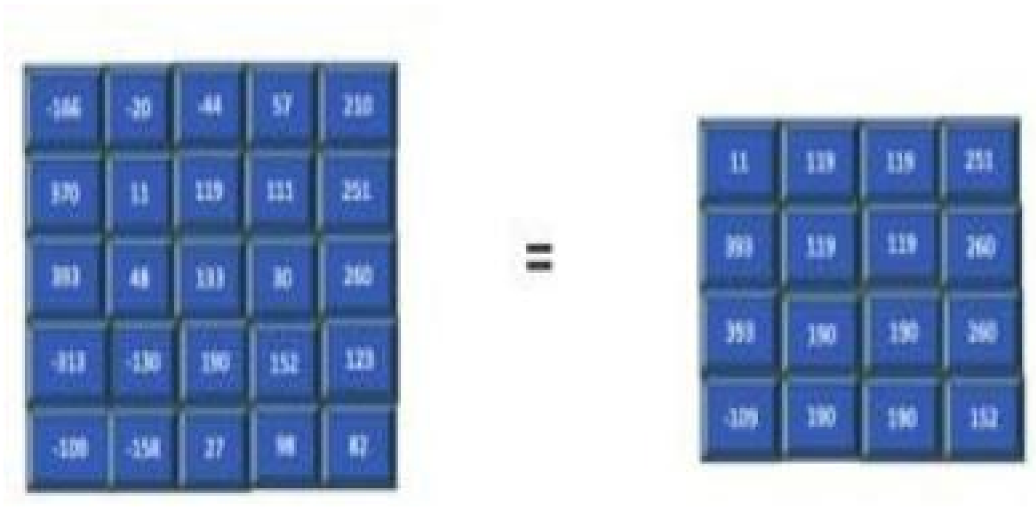


Fig 4.10 – Pooling layers [11]

Fig 4.10 shows how *max pooling* is applied to a filter of size 2 (*2x2*) and stride=1. This means that for every *2x2* cell group within the feature map, the maximum value within this region is extracted into the output cell.

Note that the previous example shows a pooling-applied outcome of just one feature map. However, in CNN, pooling is applied to feature maps that result from each filter.

Flattening

Think of CNN as a sequence of steps that are performed to effectively capture the important aspects of an image before applying ANN on it. In the previous steps, there are different transitions that are applied to the original image.

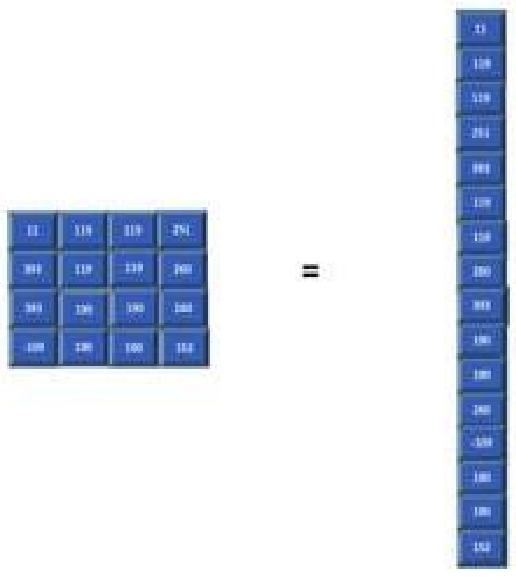


Fig 4.11 – Flattening [11]

The final step in this process is to make the outcomes of CNN be compatible with an artificial neural network. The inputs to ANN should be in the form of a vector. To support that, apply *flattening*, which is the step to convert the multidimensional array into an *nX1* vector, as shown previously.

Note that the Fig 4.11 shows flattening applied to just one feature map. However, in CNN, flattening is applied to feature maps that result from each filter.

Convolutional layers use filters or kernels to extract features from input images, and these filters are learned through backpropagation during the training process. Pooling layers downsample the feature maps obtained from convolutional layers, reducing the size of the input while preserving the most important features.The output of the convolutional and pooling layers is then flattened and fed into fully connected layers, which are similar to the layers in a traditional artificial neural network. The fully connected layers shown in Fig 4.12, learn to classify the input based on the features extracted by the convolutional and pooling layers.

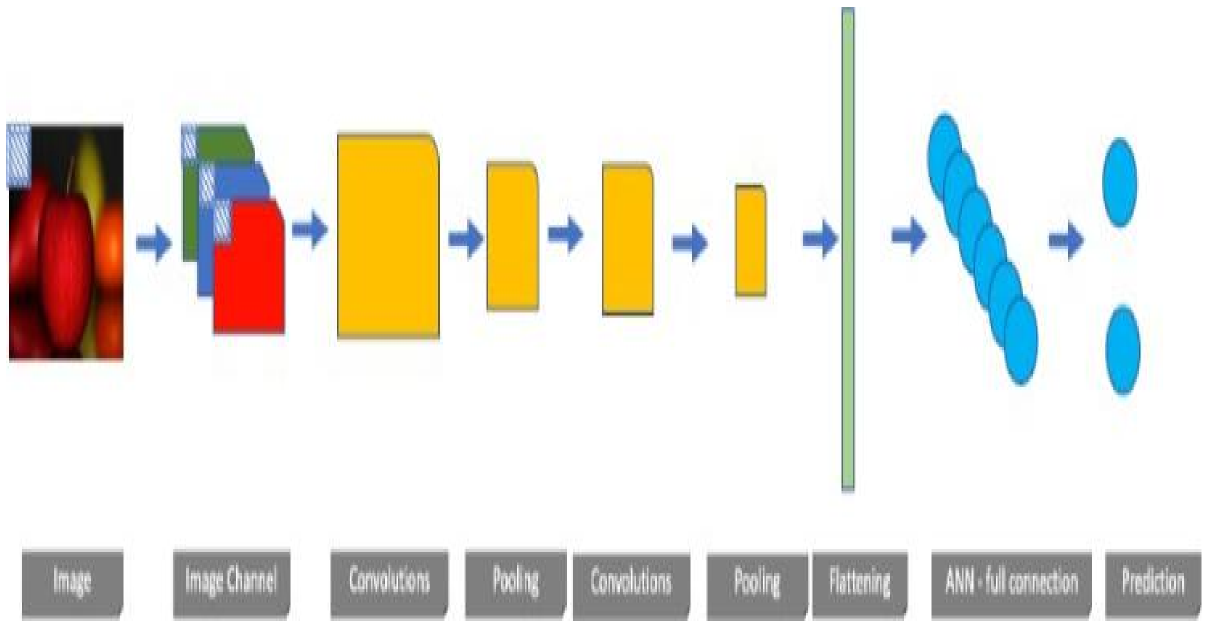


Fig 4.12 – Fully connected layers [11]

CNNs have shown impressive performance on a wide range of image and video analysis tasks and have become the state-of-the-art approach for many applications, including object detection, facial recognition, and self-driving cars.

4.4 FEATURE ANALYSIS

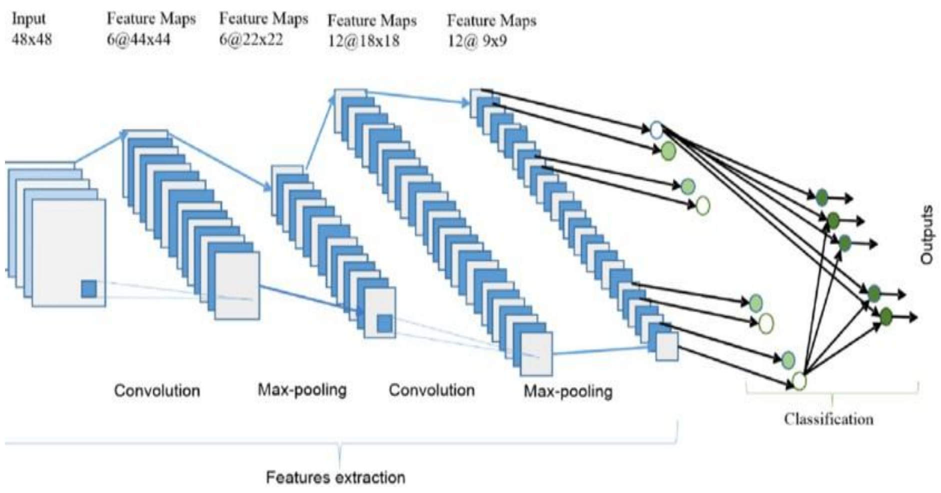


Fig 4.13 - Feature Analysis for the videos

In CNNs, feature analysis refers to the process of learning and identifying important visual features from the input images. These features are learned automatically through the training process, where the weights of the convolutional filters are adjusted to extract the most relevant features from the input data.

The feature analysis in CNNs involves a series of convolutional layers followed by pooling layers (Fig 4.13), which work together to extract hierarchical features from the input images. The convolutional layers consist of filters that slide over the input image, computing dot products between the filter weights and the pixel values in the local receptive field. The result is a feature map that highlights the presence of specific visual patterns in the input image.

The pooling layers are then used to down sample the feature maps by selecting the most representative pixel values. This helps to reduce the dimensionality of the data and makes the model more robust to spatial variations in the input images.
The extracted features from the convolutional and pooling layers are then fed into one or more fully connected layers, which perform the final classification or regression task. During training, the weights of all the layers are adjusted using backpropagation to minimize the loss function, which measures the difference between the predicted and actual outputs.

Overall, the feature analysis in CNNs is a powerful technique for automatically learning and identifying important visual patterns in complex images. This makes CNNs well-suited for a wide range of applications in computer vision, including image classification, object detection, and segmentation.

4.5 DESCRIPTION OF IMPLEMENTATION

An activity recognition system is a type of machine learning system that is designed to automatically recognize and classify human actions or activities based on input video or sequence of frames. The process of building an activity recognition system involves several key steps that are essential to ensure that the system can accurately recognize and classify the activities it is designed to identify.

The first step in building an activity recognition system is to gather and prepare a dataset of videos or sequences of frames that will be used for both training and testing the system. It is essential to ensure that the dataset is diverse and accurately labeled to ensure that the system can recognize and classify a broad range of activities.

The next step is to extract low-level features from the frames or video sequence. Low-level features are the basic building blocks of activity recognition systems, and they can include features such as color histograms, motion vectors, or edge information. These features provide important information about the video that can be used to identify and classify activities.

After extracting low-level features, the system must then derive mid-level pose/gesture or action descriptions from these features. This involves using more complex algorithms to identify and classify specific movements, gestures, or actions based on the low-level features.

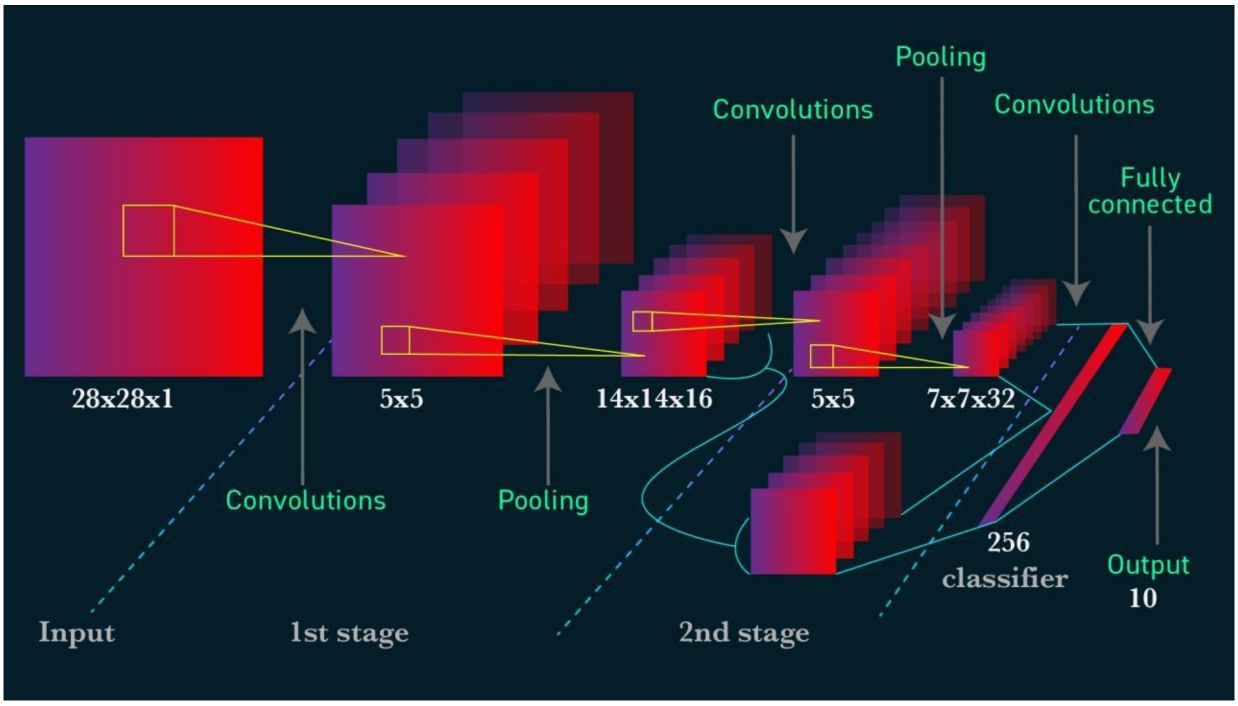


***Fig 4.14 – Descriptive performance** [13]*

To execute the activity recognition system, several steps must be followed. As shown in Fig 4.14 First, the dataset needs to be worked on, which includes cleaning the data, selecting relevant videos or frames, and labeling them accurately. Then, features must be extracted from the frames using convolutional neural networks (CNN) or other techniques. These extracted features are then passed to the training process to develop a model that can accurately classify activities. Finally, the trained model is used to classify new videos or frames and recognize the activities they contain.

In summary, an activity recognition system is built on a series of essential steps, including working with the dataset, extracting low-level features, deriving mid-level pose/gesture or action descriptions, training the model, and using it to classify videos. These steps are critical in ensuring that the system can accurately recognize and classify activities and provide useful insights into human behavior.

CHAPTER 5
IMPLEMENTAION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

Creating a convolutional neural network (CNN) model is a complex process that involves several key steps. The first and foremost step in creating a CNN model is data collection. The data collection process involves gathering and preparing a diverse dataset of images that will be used for both training and validation purposes. It is essential to ensure that the dataset is well-organized, accurately labeled, and contains a variety of images that cover all possible scenarios.

Once the dataset is prepared, the next step is data pre-processing. This step involves resizing the images to a standardized size, normalizing pixel values, and using techniques such as flipping or rotation to augment the dataset. Data pre-processing is essential to ensure that the model can recognize and differentiate between different objects in an image.

The third step in creating a CNN model is designing the architecture of the model. This involves selecting the appropriate number and type of layers, activation functions, optimization algorithms, and regularization techniques. The architecture of the model plays a crucial role in determining its accuracy and performance.

Once the architecture is designed, the fourth step is to train the CNN model using the pre-processed dataset. Training the model involves setting up the training process with hyperparameters such as the learning rate, batch size, and number of epochs. The training process is iterative, and the model learns from its mistakes and adjusts its parameters to improve its accuracy.

After training the model, it needs to be evaluated using the validation dataset to measure its performance. This is done by calculating metrics such as accuracy, precision, recall, F1-score, and a confusion matrix. Based on the evaluation results, the model can be fine-tuned, which involves adjusting the model architecture or hyperparameters to further improve its performance.

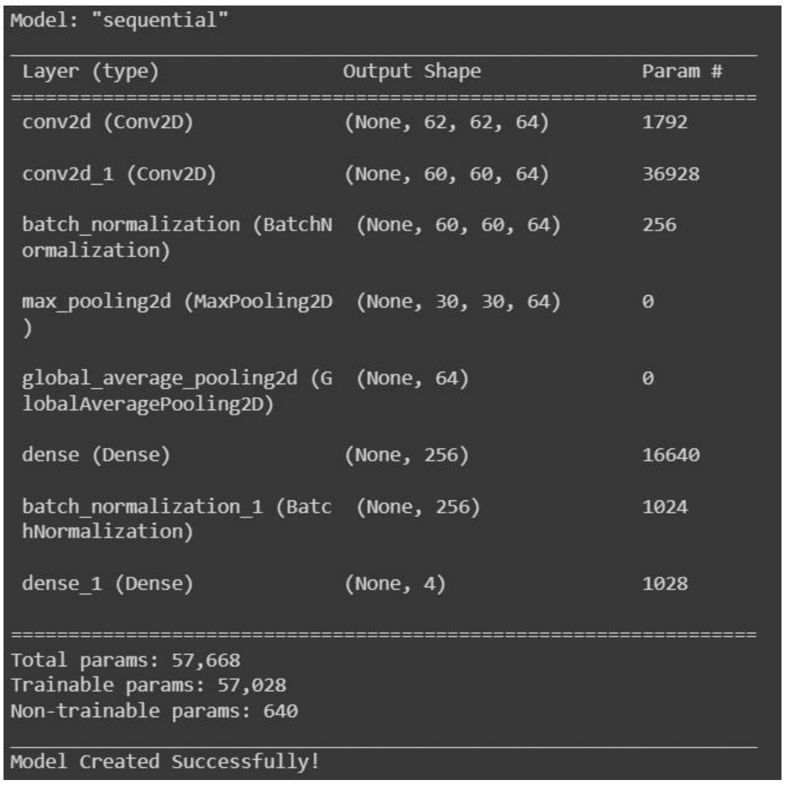
Finally, the trained and fine-tuned model can be used to make predictions on new images. The process of creating a CNN model involves careful data collection, pre-processing, architecture design, training, evaluation, fine-tuning, and prediction. Each of these steps is crucial, and any errors or inaccuracies in the process can significantly impact the accuracy and performance of the model. Therefore, it is essential to follow each step carefully and ensure that the model is thoroughly evaluated before using it for predictions.

5.2 CONSTRUCTION OF PREDICITIVE MODEL

To develop a Convolutional Neural Network (CNN), several essential libraries such as os, numpy, and dataframe need to be imported. The first step is to download and visualize data, including labels, to understand the UCF50 dataset, which comprises 50 categories of actions from realistic YouTube videos. This dataset contains an average of 133 videos per category, with 25 groups of videos per category and an average of 199 frames per video. The dataset is downloaded from the UCF Center for Research in Computer Vision in a rar format, which needs to be extracted using the unrar command to access the videos.

After randomly selecting twenty categories, the videos are preprocessed by scanning and resizing frames to (64, 64) for faster computation time. The number of frames that are fed into the model as a single sequence is specified, and the data is normalized between 0 and 1 by dividing the pixel values by 255. This normalization helps accelerate the convergence while training the network. Specific categories are chosen for model training, and the frames_extraction() function is used to create a list of resized and normalized video frames, reading each video frame by frame. The create_dataset() function then iterates through all specified classes and calls the frame_extraction() function on each video file, returning features (frames), label index, and video file path.

Two numpy arrays are created, one containing all images and another containing all class labels in an encoded format. Before splitting the data into training and testing sets, it is shuffled. A CNN model with two layers is then created, and the successful creation of the model is shown in Fig 5.1.


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 64)        1792

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        36928

 batch_normalization (BatchN  (None, 60, 60, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 30, 30, 64)       0
 )

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               16640

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 4)                 1028

=================================================================
Total params: 57,668
Trainable params: 57,028
Non-trainable params: 640
_________________________________________________________________
Model Created Successfully!
```

Fig 5.1 - Model Creation

Max pooling is critical for extracting low-level features by prioritizing the essential features of each frame. In case computation time is not a concern, a convolutional layer can be used instead of max pooling for the same purpose. Max pooling helps prevent overfitting and is useful in terms of training time. Batch normalization refers to referencing the feature maps, especially the filters. Parameters used are calculated for each map. In regular batch normalization, each feature has a completely different mean and standard deviation. However, in CNN, each feature map has a unique mean and standard deviation, utilizing all possible features it contains. This helps in making the neural network faster and more stable by adding layers to the network.

The structure of the final model is checked using the plot_model function in Fig 5.2, which is useful for creating complex networks. Finally, the model is trained and evaluated using the total loss and accuracy obtained from its performance. The created model can then be used to predict actions performed in provided videos, including those available on YouTube or other video clips. Overall, these steps are crucial to building an effective CNN for activity recognition that can be applied to various real-world scenarios.

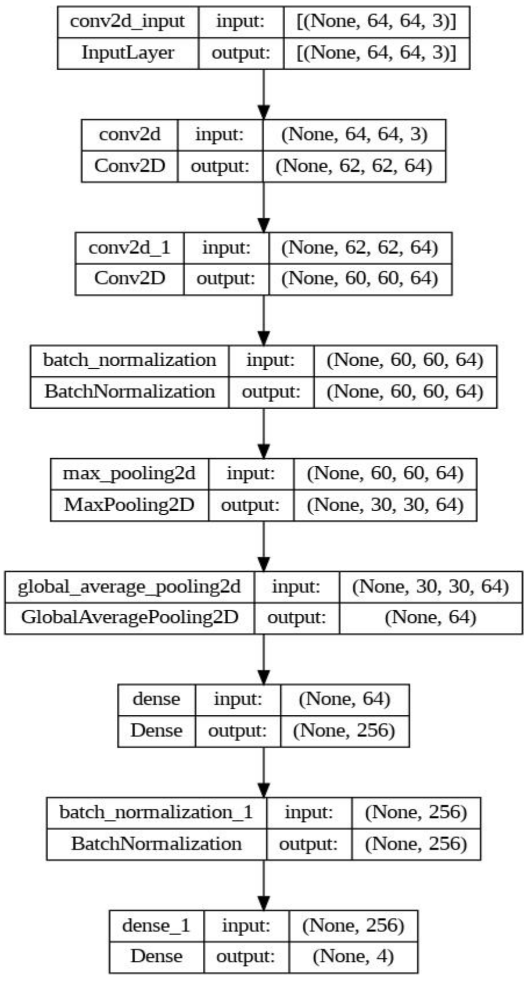

conv2d_input | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 62, 62, 64)
conv2d_1 | input: | (None, 62, 62, 64)
Conv2D | output: | (None, 60, 60, 64)
batch_normalization | input: | (None, 60, 60, 64)
BatchNormalization | output: | (None, 60, 60, 64)
max_pooling2d | input: | (None, 60, 60, 64)
MaxPooling2D | output: | (None, 30, 30, 64)
global_average_pooling2d | input: | (None, 30, 30, 64)
GlobalAveragePooling2D | output: | (None, 64)
dense | input: | (None, 64)
Dense | output: | (None, 256)
batch_normalization_1 | input: | (None, 256)
BatchNormalization | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 4)

Fig 5.2 – Structure of the model

CHAPTER 6
RESULTS AND DISCUSSION

The developed CNN model has the potential to be adapted and trained for various video classification tasks beyond surveillance. The ability to fine-tune and adjust the number of epochs allows for improved performance and the ability to handle any video input.

In addition to surveillance, the model can be applied to a wide range of use cases. For instance, in the field of sports, the CNN model can be used for real-time video classification to identify actions such as goal-scoring or fouls. Similarly, in the field of medicine, the model can be applied to classify medical images such as X-rays or MRIs to detect abnormalities or diagnose diseases.

Furthermore, the CNN model can be utilized for real-time video classification tasks that demand efficient localization and detection. This can include applications such as autonomous driving, where the model can be used to detect and classify different objects such as pedestrians, vehicles, and traffic signs in real-time.

Overall, the developed CNN model has the potential to be adapted and trained for various video classification tasks beyond surveillance and can be utilized in a wide range of applications that demand real-time classification and detection.

In the context of human activity recognition using a CNN, the total loss and total validated loss are both important metrics for evaluating the performance of the model.

The total loss is the sum of the loss values for all the examples in the training set. In the case of human activity recognition, the training set would typically consist of various physical activities such as walking, running, sitting, etc. The loss function is typically defined in such a way that lower values indicate better performance, and the goal of the training process is to minimize this loss.

The total validated loss is the sum of the loss values for all the examples in the validation set. The validation set is used to monitor the generalization performance of the model and to prevent overfitting. During training, the model is iteratively updated to minimize the total loss on the training set, while the validation set is used to evaluate the model's performance on data that it has not seen before.

In human activity recognition, the total validated loss is a crucial metric for evaluating the model's performance, as it provides an estimate of the model's ability to generalize to new, unseen data. If the total validated loss is significantly higher than the total loss, it may indicate that the model is overfitting to the training data and is not generalizing well to new data.

Overall, both the total loss and total validated loss are important metrics for evaluating the performance of a CNN-based model for human activity recognition. The goal is to minimize both of these metrics while ensuring that the model can generalize well to new, unseen data. Fig 6.1 provides total loss vs total validation loss graph for the produced model.

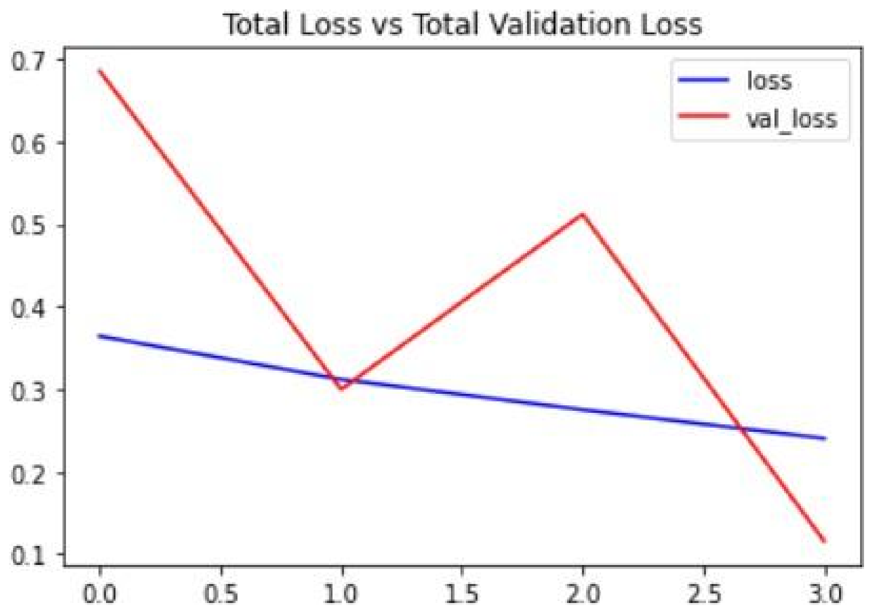


Fig 6.1 – Total loss vs total validation loss

The total validated accuracy is the percentage of correctly classified examples in the validation set. The validation set is used to monitor the generalization performance of the model and to prevent overfitting. During training, the model is iteratively updated to maximize the total accuracy on the training set, while the validation set is used to evaluate the model's performance on data that it has not seen before.

In human activity recognition, the total validated accuracy is a crucial metric for evaluating the model's performance, as it provides an estimate of the model's ability to generalize to new, unseen data. If the total validated accuracy is significantly lower than the total accuracy, it may indicate that the model is overfitting to the training data and is not generalizing well to new data. Fig 6.2 visualizes the total accuracy against the total validation accuracy for the constructed model.

Overall, both the total accuracy and total validated accuracy are important metrics for evaluating the performance of a CNN-based model for human activity recognition. The goal is to maximize both of these metrics while ensuring that the model can generalize well to new, unseen data.

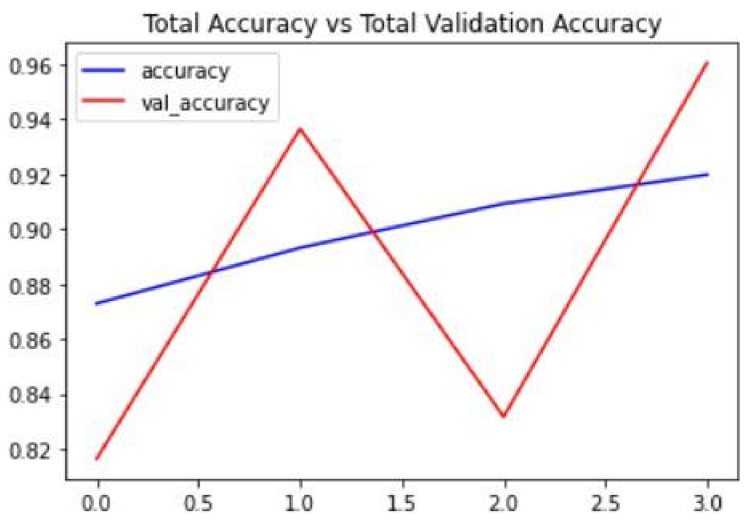


Fig 6.2- Total accuracy vs total validation accuracy

OUTPUT SCREENSHOTS

The model has been evaluated using a loss function and validation set, and based on the obtained results, it correctly identifies the following,

1. A man is playing a keyboard in the image as we can see in Fig 6.3.

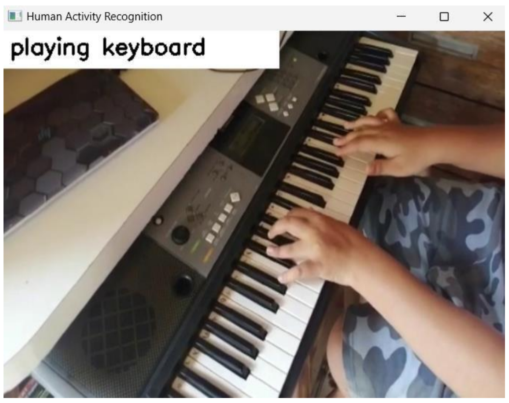


Fig 6.3 Playing Keyboard

2. A man is about to perform a side kick in Fig 6.4.

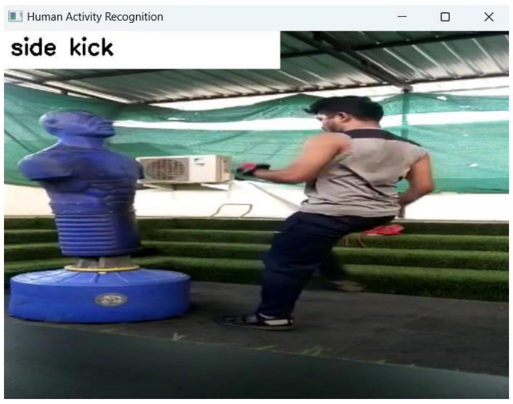


Fig 6.4 Side Kick

3. A man is taking pushups in Fig 6.5

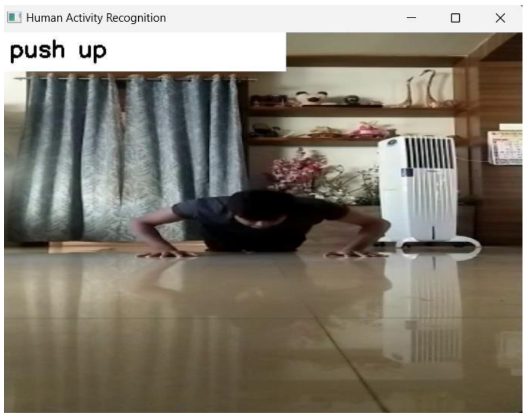


Fig 6.4 Push Up

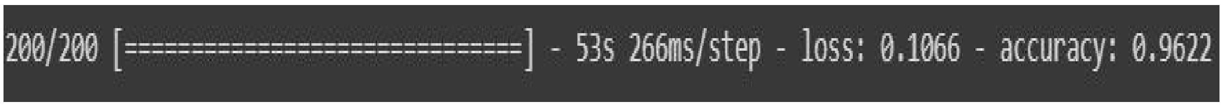
This implies that the model has learned the relevant features that allow it to distinguish between different human activities and is performing well in this specific task**.**

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

In this statement, it is being emphasized that the current method is superior to existing approaches in terms of accuracy while requiring fewer nodes and layers in the CNN model. This indicates that the current method is more efficient and effective in handling classification tasks. Furthermore, it is stated that the model has the ability to generalize across various datasets, suggesting that it is capable of learning from different types of human activity data.

The performance of the model is evaluated based on its detection and recognition accuracy. The model achieved a detection accuracy, indicating that it was able to correctly identify actions being performed in the videos. Moreover, the recognition accuracy of 96.22% at around 22 frames per second is also notable(Fig 7.1), indicating that the model was able to recognize specific actions being performed in the videos.



```
200/200 [==============================] - 53s 266ms/step - loss: 0.1066 - accuracy: 0.9622
```

Fig 7.1 – Model evaluation

7.2 FUTURE WORK

It is important to note that the performance of the model was achieved after twenty epochs, which suggests that further fine-tuning and adjustments to the number of epochs could potentially improve its performance even further. Overall, these results demonstrate the effectiveness of the current method in accurately detecting and recognizing human activity in videos, making it a promising solution for a wide range of applications, including real-time video classification for efficient localization and detection.

7.3 RESEARCH ISSUES

One of the primary challenges in human activity recognition using CNN is the lack of annotated training data. CNNs require large amounts of annotated data to learn and generalize well. Therefore, data augmentation techniques and transfer learning methods are used to overcome this challenge. Recognizing human activities across different users with different body types and motion patterns is a research challenge. Developing a recognition model that can generalize across users is critical for practical applications. Human activity recognition using CNN raises privacy concerns, as the recognition model can reveal sensitive information about the user. Developing privacy-preserving recognition models is a research challenge.

7.4 IMPLEMENTATION ISSUES

Needs a lot of training data is needed for the CNN to be effective and that they fail to encode the position and orientation of objects. They fail to encode the position and orientation of objects. They have a hard time classifying images with different positions. CNNs tend to be much slower because of operations like maxpool. In case the convolutional neural network is made up of multiple layers, the training process could take a particularly long time if the computer does not have a good GPU. Convolutional neural networks will recognize the image as clusters of pixels which are arranged in distinct patterns. They don’t understand them as components present in the image.

REFERENCES:-

[1] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1725-1732.

[2] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, Kaiming He; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 6202-6211

[3] D. Weinland, R. Ronfard, and E. Boyer, “A survey of vision-based methods for action representation, segmentation and recognition,” Computer Vision and Image Understanding, vol. 115, no. 2, pp. 224 – 241, 2011.

[4] G. Cheng, Y. Wan, A. N. Saudagar, K. Namuduri, and B. P. Buckles, “Advances in human action recognition: a survey,” arXiv preprint arXiv:1501.05964, 2015.

[5] Hieu H. Pham, Louahdi Khoudour, Alain Crouzil, Pablo Zegers, Sergio A. Velastin doi.org/10.48550/arXiv.2208.03775,2022.

[6] J. K. Aggarwal and Q. Cai, “Human motion analysis: a review,” in Proceedings IEEE Nonrigid and Articulated Motion Workshop, Jun 1997, pp. 90–102.

[7] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, “Machine recognition of human activities: A survey,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 11, pp. 1473–1488, Nov 2008.

[8] R. Poppe, “A survey on vision-based human action recognition,” Image and Vision Computing, vol. 28, no. 6, pp. 976 – 990, 2010.

[9] T. B. Moeslund and E. Granum, “A survey of computer visionbased human motion capture,” Computer vision and image understanding, vol. 81, no. 3, pp. 231–268, 2001.

[10] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, Rahul Sukthankar; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 1130-1.

[11]https://developer.ibm.com/articles/introduction-to-convolutional-neural-networks/

[12] https://www.ibm.com/in-en/topics/computer-vision#:~:text=Resources-,What%20is%20computer%20vision%3F,recommendations%20based%20on%20that%20information

[13] https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/

APPENDIX

A. Source code

```
import os
import cv2
import math
import pafy
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from moviepy.editor import *
from collections import deque
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model

seed_constant = 23
np.random.seed(seed_constant)
random.seed(seed_constant)
tf.random.set_seed(seed_constant)

!wget -nc --no-check-certificate https://www.crcv.ucf.edu/data/UCF50.rar
!unrar x UCF50.rar -inul -y

plt.figure(figsize = (30, 30))

# Get Names of all classes in UCF50
all_classes_names = os.listdir('UCF50')

# Generate a random sample of images each time the cell runs
random_range = random.sample(range(len(all_classes_names)), 20)

# Iterating through all the random samples
for counter, random_index in enumerate(random_range, 1):
    # Getting Class Name using Random Index
    selected_class_Name = all_classes_names[random_index]

    # Getting a list of all the video files present in a Class Directory
    video_files_names_list = os.listdir(f'UCF50/{selected_class_Name}')

    # Randomly selecting a video file
    selected_video_file_name = random.choice(video_files_names_list)

    # Reading the Video File Using the Video Capture
    video_reader =
cv2.VideoCapture(f'UCF50/{selected_class_Name}/{selected_video_file_name}')

    # Reading The First Frame of the Video File
    _, bgr_frame = video_reader.read()

    # Closing the VideoCapture object and releasing all resources.
    video_reader.release()

    # Converting the BGR Frame to RGB Frame
    rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)

    # Adding The Class Name Text on top of the Video Frame.
    cv2.putText(rgb_frame, selected_class_Name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0 , 255), 2)

    # Assigning the Frame to a specific position of a subplot
    plt.subplot(5, 4, counter)
    plt.imshow(rgb_frame)
    plt.axis('off')

    image_height, image_width = 64, 64
max_images_per_class = 8000

dataset_directory = "UCF50"
classes_list = ["WalkingWithDog", "TaiChi", "Swing", "HorseRace"]

model_output_size = len(classes_list)
```

```
def frames_extraction(video_path):
    # Empty List declared to store video frames
    frames_list = []

    # Reading the Video File Using the VideoCapture
    video_reader = cv2.VideoCapture(video_path)

    # Iterating through Video Frames
    while True:

        # Reading a frame from the video file
        success, frame = video_reader.read()

        # If Video frame was not successfully read then break the loop
        if not success:
            break

        # Resize the Frame to fixed Dimensions
        resized_frame = cv2.resize(frame, (image_height, image_width))

        # Normalize the resized frame by dividing it with 255 so that each pixel value then lies
between 0 and 1
        normalized_frame = resized_frame / 255

        # Appending the normalized frame into the frames list
        frames_list.append(normalized_frame)

    # Closing the VideoCapture object and releasing all resources.
    video_reader.release()

    # returning the frames list
    return frames_list

def create_dataset():

    # Declaring Empty Lists to store the features and labels values.
    temp_features = []
    features = []
    labels = []

    # Iterating through all the classes mentioned in the classes list
    for class_index, class_name in enumerate(classes_list):
        print(f'Extracting Data of Class: {class_name}')

        # Getting the list of video files present in the specific class name directory
        files_list = os.listdir(os.path.join(dataset_directory, class_name))

        # Iterating through all the files present in the files list
        for file_name in files_list:

            # Construct the complete video path
            video_file_path = os.path.join(dataset_directory, class_name, file_name)

            # Calling the frame_extraction method for every video file path
            frames = frames_extraction(video_file_path)

            # Appending the frames to a temporary list.
            temp_features.extend(frames)

        # Adding randomly selected frames to the features list
        features.extend(random.sample(temp_features, max_images_per_class))

        # Adding Fixed number of labels to the labels list
        labels.extend([class_index] * max_images_per_class)

        # Emptying the temp_features list so it can be reused to store all frames of the next class.
        temp_features.clear()

    # Converting the features and labels lists to numpy arrays
    features = np.asarray(features)
    labels = np.array(labels)

    return features, labels

features, labels = create_dataset()

one_hot_encoded_labels = to_categorical(labels)
```

```
def create_model():

    # use a Sequential model for model construction
    model = Sequential()

    # Defining The Model Architecture
    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu', input_shape =
(image_height, image_width, 3)))
    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2, 2)))
    model.add(GlobalAveragePooling2D()) model.add(Dense(256,
    activation = 'relu')) model.add(BatchNormalization())
    model.add(Dense(model_output_size, activation = 'softmax'))

    # Printing the models summary
    model.summary()

    return model


# Calling the create_model method
model = create_model()

print("Model Created Successfully!")

plot_model(model, to_file = 'model_structure_plot.png', show_shapes = True,
show_layer_names = True)

early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 15, mode = 'min',
restore_best_weights = True)

# Adding loss, optimizer and metrics values to the model.
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])

# Start Training
```

```
model_training_history = model.fit(x = features_train, y = labels_train, epochs = 10, batch_size
= 4 , shuffle = True, validation_split = 0.2, callbacks = [early_stopping_callback])

model_evaluation_history = model.evaluate(features_test, labels_test)

date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history
model_name =
f'Model___Date_Time_{current_date_time_string}___Loss_{model_evaluation_loss}___Accura
cy_{model_evaluation_accuracy}.h5'

# Saving the Model
model.save(model_name)

def plot_metric(metric_name_1, metric_name_2, plot_name):
  # Get Metric values using metric names as identifiers
  metric_value_1 = model_training_history.history[metric_name_1]
  metric_value_2 = model_training_history.history[metric_name_2]

  # Constructing a range object which will be used as time
  epochs = range(len(metric_value_1))

  # Plotting the Graph
  plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
  plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

  # Adding title to the plot
  plt.title(str(plot_name))

  # Adding legend to the plot
  plt.legend()

plot_metric('loss', 'val_loss', 'Total Loss vs Total Validation Loss')

plot_metric('accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')

def download_youtube_videos(youtube_video_url, output_directory):
    # Creating a Video object which includes useful information regarding the youtube video.
    video = pafy.new(youtube_video_url)

    # Getting the best available quality object for the youtube video.
    video_best = video.getbest()

    # Constructing the Output File Path
    output_file_path = f'{output_directory}/{video.title}.mp4'

    # Downloading the youtube video at the best available quality.
    video_best.download(filepath = output_file_path, quiet = True)

    # Returning Video Title
    return video.title

def predict_on_live_video(video_file_path, output_file_path, window_size):

    # Initialize a Deque Object with a fixed size which will be used to implement moving/rolling
average functionality.
    predicted_labels_probabilities_deque = deque(maxlen = window_size)

    # Reading the Video File using the VideoCapture Object
    video_reader = cv2.VideoCapture(video_file_path)

    # Getting the width and height of the video
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Writing the Overlayed Video Files Using the VideoWriter Object
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P', '4', 'V'), 24,
(original_video_width, original_video_height))

    while True:

        # Reading The Frame
        status, frame = video_reader.read()

        if not status:
            break
        # Resize the Frame to fixed Dimensions
        resized_frame = cv2.resize(frame, (image_height, image_width))

        # Normalize the resized frame by dividing it with 255 so that each pixel value then lies
between 0 and 1
        normalized_frame = resized_frame / 255

        # Passing the Image Normalized Frame to the model and receiving Predicted
Probabilities.
        predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis =
0))[0]

        # Appending predicted label probabilities to the deque object
        predicted_labels_probabilities_deque.append(predicted_labels_probabilities)

        # Assuring that the Deque is completely filled before starting the averaging process
        if len(predicted_labels_probabilities_deque) == window_size:

            # Converting Predicted Labels Probabilities Deque into Numpy array
            predicted_labels_probabilities_np = np.array(predicted_labels_probabilities_deque)

            # Calculating Average of Predicted Labels Probabilities Column Wise
            predicted_labels_probabilities_averaged =
predicted_labels_probabilities_np.mean(axis = 0)

            # Converting the predicted probabilities into labels by returning the index of the
maximum value.
            predicted_label = np.argmax(predicted_labels_probabilities_averaged)

            # Accessing The Class Name using predicted label.
            predicted_class_name = classes_list[predicted_label]

            # Overlaying Class Name Text Ontop of the Frame
            cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0, 255), 2)

        # Writing The Frame
        video_writer.write(frame)
        # cv2.imshow('Predicted Frames', frame)

        # key_pressed = cv2.waitKey(10)

        # if key_pressed == ord('q'):
        #     break

    # cv2.destroyAllWindows()


    # Closing the VideoCapture and VideoWriter objects and releasing all resources held by
them.
    video_reader.release()
    video_writer.release()

output_directory = 'Youtube_Videos'
os.makedirs(output_directory, exist_ok = True)

# Downloading a YouTube Video
video_title = download_youtube_videos('https://www.youtube.com/watch?v=e4VIw41R-PU',
output_directory)

# Getting the YouTube Video's path
input_video_file_path = f'{output_directory}/{video_title}.mp4'

window_size = 1

# Constructing The Output YouTube Video Path
output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4'

# Calling the predict_on_live_video method to start the Prediction.
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)

def make_average_predictions(video_file_path, predictions_frames_count):
    # Initializing the Numpy array which will store Prediction Probabilities
    predicted_labels_probabilities_np = np.zeros((predictions_frames_count,
model_output_size), dtype = np.float)

    # Reading the Video File using the VideoCapture Object
    video_reader = cv2.VideoCapture(video_file_path)

    # Getting The Total Frames present in the video
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

    # Calculating The Number of Frames to skip Before reading a frame
    skip_frames_window = video_frames_count // predictions_frames_count

    for frame_counter in range(predictions_frames_count):

        # Setting Frame Position
        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter *
skip_frames_window)

        # Reading The Frame
        _ , frame = video_reader.read()

        # Resize the Frame to fixed Dimensions
        resized_frame = cv2.resize(frame, (image_height, image_width))

        # Normalize the resized frame by dividing it with 255 so that each pixel value then lies
between 0 and 1
        normalized_frame = resized_frame / 255

        # Passing the Image Normalized Frame to the model and receiving Predicted
Probabilities.
        predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis =
0))[0]

        # Appending predicted label probabilities to the deque object
        predicted_labels_probabilities_np[frame_counter] = predicted_labels_probabilities

    # Calculating Average of Predicted Labels Probabilities Column Wise
    predicted_labels_probabilities_averaged = predicted_labels_probabilities_np.mean(axis = 0)

    # Sorting the Averaged Predicted Labels Probabilities
    predicted_labels_probabilities_averaged_sorted_indexes =
np.argsort(predicted_labels_probabilities_averaged)[::-1]

    # Iterating Over All Averaged Predicted Label Probabilities
    for predicted_label in predicted_labels_probabilities_averaged_sorted_indexes:

        # Accessing The Class Name using predicted label.
        predicted_class_name = classes_list[predicted_label]

        # Accessing The Averaged Probability using predicted label.
        predicted_probability = predicted_labels_probabilities_averaged[predicted_label]

        print(f"CLASS NAME: {predicted_class_name}   AVERAGED PROBABILITY:
{(predicted_probability*100):.2}")

    # Closing the VideoCapture Object and releasing all resources held by it.
    video_reader.release()
```

B. Research paper

HUMAN ACTIVITY RECOGNITION USING CNN

Kharishma. V, Keerthana. J. R, Dr Bharathi B, M.E., Ph.D

ABSTRACT

Activity recognition in computer vision involves identifying the action present in a given video, while action detection locates actions in space and time. It has become an important area of interest with applications in automated surveillance, automated vehicles, language interpretation, and human-computer interfaces. Deep learning has been widely used in recent years to improve the accuracy of action recognition in videos. The ultimate goal is to improve the effectiveness of real-time human action recognition. This paper provides motivation for efficient human activity recognition by analyzing CCTV and camera images to detect and classify human activities using HAAR-based and Convolutional Neural Network (CNN) classifiers. It also identifies the best-performing deep architectures in action recognition and outlines future trends and open problems in this field. The proposed model aims to achieve high detection and recognition accuracy at approximately 22 frames per second after twenty epochs using a dataset of 5648 images.

Keywords – Human Action Recognition, Deep Learning, Computer vision, Neural Networks, CNN's, Localization, Accuracy

INTRODUCTION

CNN plays an essential role in human activity recognition, which has gained significant attention in the computer vision community. The objective is to identify and distinguish various types of human activities by integrating life cycle modules, including object detection, segmentation, and recognition. This paper implemented the HAAR Feature-based classifier, which utilizes knowledge-based techniques to perform a cascade feature on a set of positive and negative frames. Activity recognition helps to identify the type of activity, such as walking, standing, sitting, and others, which is useful in various applications. The dataset used comprises frames of people from diverse occasions in different locations, which are used for training and testing. This paper propose a unique approach that differs from traditional methods to define and recognize human activities. The proposed approach is divided into two categories: dataset and model creation. The model design includes people detection, localization, segmentation, and video image recognition, which utilize common features and temporal statistics to achieve high accuracy for real-time applications. However, the computational burden increases exponentially as more features are measured. To detect and recognize human motion from videos or images, a model is needed that can produce high localization and accuracy.

LITERATURE SURVEY

This section considers publicly available data and related surveys on human activity recognition.

Turaga et al [6] in their paper on machine recognition activities: A survey (2003) discusses the importance of recognizing human activities in videos for various applications. However, challenges in robustness and semantic representation make this task difficult. The paper surveys past efforts to address representation, recognition, and learning of human activities from video and classifies approaches based on their ability to handle varying degrees of complexity at two levels: "actions" and "activities." The paper explores several approaches, including modeling atomic or primitive actions, modeling actions with more complex dynamics, and higher-level representation of complex activities.

R Poppe [7] in their paper on Survey on vison-based human action recognition (2010) examines the process of labeling image sequences with action labels and its applications in various domains. Poppe discusses the challenges involved and provides a comprehensive overview of current research in the field, including discussions on image representations and classification processes. The author also explores limitations and suggests future research directions.

Weinland et al [3] in their paper on A survey of vision-based action representation, segmentation and recognition (2011) categorizes different approaches used in action recognition based on how they represent spatial and temporal structure, segment actions, and learn view-invariant representations. The paper primarily focuses on full-body motion classification, including actions like kicking, punching, and waving, and discusses their applications in robotics, video surveillance, human-computer interaction, and multimedia retrieval.

Andrej Karpathy [1] conducted a study in 2014 on large-scale video classification using Convolutional Neural Networks (CNNs) and a dataset of 1 million YouTube videos. They explored various methods to incorporate local spatio-temporal information by extending CNN connectivity in the time domain, and found a multiresolution, foveated architecture to be effective in speeding up training. The best spatio-temporal networks slightly outperformed single-frame models and showed improvements compared to feature-based baselines. Additionally, Karpathy observed significant performance improvements in generalization by retraining the top layers on the UCF-101 Action Recognition dataset.

Cheng et al [4] conducted a survey that highlighted recent advances in human action recognition in computer vision (2015). The paper covers low-level actions and high-level activities and categorizes them according to a well-formed taxonomy. The authors aimed to provide a comprehensive overview of the progress in this area to guide future research in fields such as video surveillance, human-machine interaction, and video retrieval.

Yu-Wei Chao [8] introduced TAL-Net, an enhanced method for temporal action localization in videos based on the Faster R-CNN object detection framework in 2018. The approach aimed to overcome three main limitations of previous methods by using a multi-scale architecture to improve receptive field alignment, better utilizing the temporal context of actions, and explicitly considering multi-stream feature fusion. TAL-Net achieved the best results for both action proposal and localization on the THUMOS'14 detection benchmark and showed competitive performance in the ActivityNet challenge.

Feichtenhofer et al [2] in their paper introduced SlowFast networks for video recognition(2019), which contain both Slow and Fast pathways to capture spatial semantics at a low frame rate and fine temporal resolution motion at a high frame rate, respectively. They achieved lightweight models by reducing the channel capacity of the Fast pathway while still capturing useful temporal information. Their models outperformed state-of-the-art results for action classification and detection on Kinetics, Charades, and AVA benchmarks, with the SlowFast concept identified as a significant contributor to their accuracy improvements.

EXISTING SYSTEM

The literature survey reveals a scarcity of reliable reviews and accessible methodologies for human activity recognition based on video classification using single frame CNN. However, activity recognition is a rapidly growing field in industries such as automotive, and innovative algorithms are emerging to help understand and recognize actions from visual data. Previous studies have attempted to locate occupants in a room through radio power (RSS) analysis, but with mixed success due to the strength of the RSS being dependent on gaps and limitations in the line of sight. Consequently, an automated system that relies solely on RSS in one environment may not perform as well in others. Accurately analyzing and predicting the precise location at a very short distance is crucial, particularly for localization of indoor objects in settings such as offices, hotels, and public places. Current models struggle to differentiate between the surrounding environment and actions, leading to poor localization. The detection and recognition accuracies of the current models for training and testing data are 80.41% and 74.14%, respectively. For real-time activity recognition, localization and accuracy are critical factors. This paper aims not only to provide a comparative analysis of the current state of human activity recognition but also to highlight new trends in this field.

PROPOSED SYSTEM

This paper presents the development of a convolutional neural network for the purpose of detecting and identifying human activity in a particular video frame. The proposed model not only detects and identifies the activity but also incorporates localization by utilizing a combination of diverse theories to create an approachable model. The activity recognition model involves three steps: capturing the input video or sequence of frames, extracting low-level features from the frames, and generating mid-level pose, gesture, or action descriptions from the low-level features. The process involves working with a dataset, using CNN to extract features from the frames, training the model with the extracted features, and finally using the model to classify the videos. The primary objective is to achieve high levels of accuracy and localization for the input videos.

Description of implementation

Creating a convolutional neural network (CNN) model involves several key steps. The first step is data collection, which involves gathering and preparing a dataset of images for training and validation. It is important that this dataset is diverse, well-organized, and accurately labeled. The next step is data pre-processing, which involves resizing images to a standardized size, normalizing pixel values, and using techniques like flipping or rotation to augment the dataset.

The third step is designing the architecture of the CNN model. This involves selecting the appropriate number and type of layers, activation functions, optimization algorithms, and regularization techniques. The fourth step is training the CNN model using the pre-processed dataset. This involves setting up the training process with hyperparameters such as the learning rate, batch size, and number of epochs.

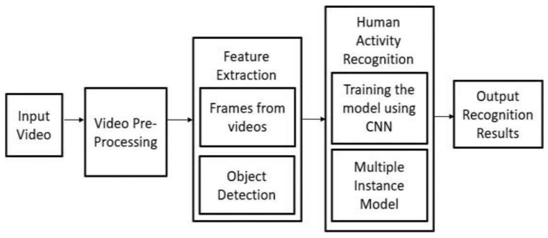


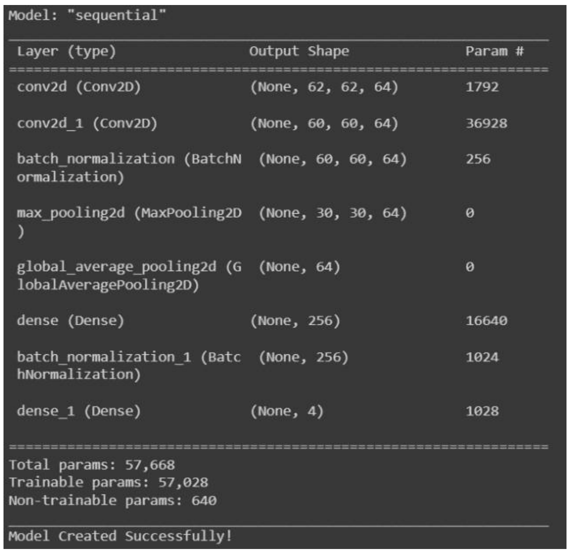
Fig 1 – Overall design of current system

After training the model, it needs to be evaluated using the validation dataset to measure its performance. This is done by calculating metrics such as accuracy, precision, recall, F1-score, and a confusion matrix. Based on the evaluation results, the model can be fine-tuned, which involves adjusting the model architecture or hyperparameters.

Finally, the trained and fine-tuned model can be used to make predictions on new images. Overall, the process of creating a CNN model involves careful data collection, pre-processing, architecture design, training, evaluation, fine-tuning, and prediction.

Construction of a Predictive Model

To create a Convolutional Neural Network (CNN), it is necessary to import essential libraries like os, numpy, and dataframe. The initial steps involve downloading and visualizing data, including labels, to gain insights into the UCF50 dataset, which consists of realistic YouTube videos and contains 50 categories of actions. The dataset has an average of 133 videos per category, with 25 groups of videos per category and an average of 199 frames per video. The dataset is downloaded from the UCF Center for Research in Computer Vision in a rar format, which requires unrar command to extract the videos.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 64)        1792

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        36928

 batch_normalization (BatchN  (None, 60, 60, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 30, 30, 64)       0
 )

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               16640

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 4)                 1028

=================================================================
Total params: 57,668
Trainable params: 57,028
Non-trainable params: 640
_________________________________________________________________
Model Created Successfully!
```

Fig 2 : Model Creation

After randomly selecting twenty categories, the videos are preprocessed by scanning and resizing frames to (64, 64) for faster computation time. The number of frames fed into the model as a single sequence is specified. The data is normalized between 0 and 1 by dividing the pixel values by 255, which accelerates convergence while training the network. Specific categories are chosen for model training. The frames_extraction() function creates a list of resized and normalized video frames, reading each video frame by frame. The create_dataset() function iterates through all specified classes and calls the frame_extraction() function on each video file, returning features (frames), label index, and video file path.

Two numpy arrays are created, one containing all images and another containing all class labels in an encoded format. Before splitting the data into training and testing sets, it is shuffled. A CNN model with two layers is then created, and the successful creation of the model is shown in Fig 2.

Max pooling is critical for extracting low-level features by prioritizing the important features of each frame. In case computation time is not a concern, a convolutional layer can be used instead of max pooling for the same purpose. Max pooling helps in preventing overfitting and is useful in terms of training time. Batch normalization refers to referencing the feature maps, especially the filters. Parameters used are calculated for each map. In regular batch normalization, each feature has a completely different mean and standard deviation. However, in CNN, each feature map has a unique mean and standard deviation, utilizing all possible features it contains. This helps in making the neural network faster and more stable by adding layers to the network.

The structure of the final model is checked using the plot_model function in Fig 3, which is useful for creating complex networks.

Finally, the model is trained and evaluated using the total loss and accuracy obtained from its performance. The created model can then be used to predict actions performed in provided videos, including those available on YouTube or other video clips.

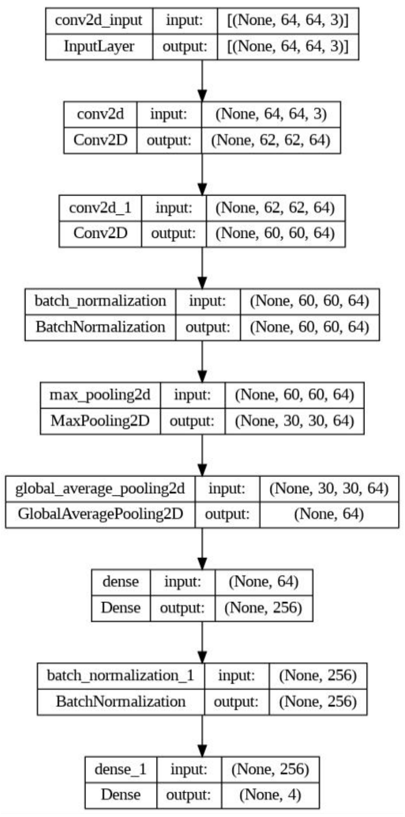

conv2d_input | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 62, 62, 64)
conv2d_1 | input: | (None, 62, 62, 64)
Conv2D | output: | (None, 60, 60, 64)
batch_normalization | input: | (None, 60, 60, 64)
BatchNormalization | output: | (None, 60, 60, 64)
max_pooling2d | input: | (None, 60, 60, 64)
MaxPooling2D | output: | (None, 30, 30, 64)
global_average_pooling2d | input: | (None, 30, 30, 64)
GlobalAveragePooling2D | output: | (None, 64)
dense | input: | (None, 64)
Dense | output: | (None, 256)
batch_normalization_1 | input: | (None, 256)
BatchNormalization | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 4)

Fig 3 – Structure of the model

RESULTS AND DISCUSSION

The generated model can be trained and adapted to handle any video input. Additionally, it can be fine-tuned and the number of epochs adjusted for improved performance. Apart from surveillance, the video classifier can be applied to various other use cases. For real-time video classification, which demands efficient localization and detection, this CNN model can be utilized effectively.

CONCLUSION

The current method demonstrates better and comparable accuracy compared to existing approaches that employ significantly more nodes and layers. Moreover, the model has the ability to generalize across various classification tasks, as demonstrated by its performance on a range of human activity datasets. The model showed acceptable results throughout the training and testing period, achieving detection accuracy and a recognition accuracy of 96.22% at around 22 frames per second after twenty epochs.

REFERENCES

[1] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1725-1732.

[2] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, Kaiming He; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 6202- 6211.

[3] D. Weinland, R. Ronfard, and E. Boyer, "A survey of vision-based methods for action representation, segmentation and recognition," Computer Vision and Image Understanding, vol. 115, no. 2, pp. 224 – 241, 2011.

[4] G. Cheng, Y. Wan, A. N. Saudagar, K. Namuduri, and B. P. Buckles, "Advances in human action recognition: a survey," arXiv preprint arXiv:1501.05964, 2015.

[5] Hieu H. Pham, Louahdi Khoudour, Alain Crouzil, Pablo Zegers, Sergio A. Velastin doi.org/10.48550/arXiv.2208.03775,2022.

[6] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: A survey," IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 11, pp. 1473–1488, Nov 2008.

[7] R. Poppe, "A survey on vision-based human action recognition," Image and Vision Computing, vol. 28, no. 6, pp. 976 – 990, 2010.

[8]Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, Rahul Sukthankar; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 1130-1139