ENHANCING HUMAN ACTIVITY RECOGNITION PERFORMANCE USING DEEP LEARNING TECHNIQUES

MAJOR PROJECT

Submitted by:

Yash Daga

2K21/SWE/25

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

Under the supervision of

Ms. Shweta Meena

SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi - 110042

MAY, 2023

MTech (Software Engineering) Yash Daga 2023

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

Bawana Road, Delhi - 110042

Candidate's Declaration

I, YASH DAGA, Roll No. 2K21/SWE/25 student of M.Tech (Software Engineering), hereby declare that the project Report titled "Enhancing Human Activity Recognition Performance Using Deep Learning Techniques" which is submitted by me to the Department of Software Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

[illegible] 31/05/2023

YASH DAGA

Place: Delhi

Date: 31/05/2023

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

Bawana Road, Delhi - 110042

CERTIFICATE

I hereby certify that the Project titled "Enhancing Human Activity Recognition performance using Deep Learning techniques" which is submitted by Yash Daga (2K21/SWE/25) for fulfillment of the requirements for awarding of the degree of Master of Technology (M.Tech) is a record of the project work carried out by students under my guidance & supervision. To the best of my knowledge, this work has not been submitted in any part of the fulfillment for any Degree or Diploma to this University or elsewhere.

[illegible] 31/5/2023

Shweta Meena

(SUPERVISOR)

Assistant Professor

Department of Software Engineering

Delhi Technological University

Place: Delhi

Date: 31/05/2023

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my project guide Ms Shweta Meena as well as our Head of Department Prof. Ruchika Malhotra who gave me the golden opportunity to do this wonderful project on the topic “Enhancing Human Activity Recognition Performance Using Deep Learning Techniques”, which also helped me in doing a lot of research and I came to know about so many new things, I am really thankful to them.

Secondly, I would also like to thank my parents, my sister and my friends who helped me a lot in finalizing this project within the limited time frame.

ABSTRACT

Numerous astounding miracles that will improve our lives have been inspired by the advancement of technology. Computer vision, image detection, and facial recognition are innovative techniques that also make it possible to recognise human behavior. Activity tracking eliminates the need for manual entry and enables forecasting and analysis of human behavior, exposing hitherto hidden benefits.

Here, we've explored many approaches to carrying out human activity recognition and contrasted them in terms of benefits, effectiveness, accuracy, methodology, datasets, and constraints. Along with the difficulties and the approach employed, we also covered several areas where this could be useful, including augmented reality, security, and the healthcare industry.

CONTENTS

List of Figures

List of Tables

CHAPTER-1
INTRODUCTION

The modernization of technology has sparked plenty of incredible accomplishments which will make our life hassle-free. The pioneering methods that also made it possible to recognise human behaviour include computer vision, image detection, & facial recognition.

1.1 General Overview

Using computer & vision technology, HAR (human activity recognition) determines human motions. Activities, gestures, or behaviours which are classified as human motions are recorded by sensors. In order for computers to understand and carry out the action commands, the movement data is subsequently translated into human activity recognition code. Activity tracking makes it possible to forecast and analyze human behaviour, revealing previously untapped advantages and removing the need for manual entry.

1.2 Problem Statement

In order to build automated systems or enhance images several studies have been conducted over a few years to extract significant information from images and videos. But in recent years, the demand for such automated systems has increased rapidly and also with that the complexity has increased as well. This has also increased the complexity by which the processing of data takes place for example we now need to work with certain other challenges such as noises in data, abrupt motion or illumination variation and background variation. Human Activity Recognition seeks to identify the actions carried out by a person or group of people based on data extracted by video or sensors. In an ideal scenario, detection of activities performed by an individual can be done irrespective of the environment it is being performed in [14].

One of the most significant and difficult tasks in computer vision problems is human activity recognition as it has vast usage in various domains which include security, healthcare and many more.

1.3 Applications

The purpose of human activity recognition is automation. Majorly, if there is a need to automate daily human activities then the first stage would be to detect the actions carried out by a person or group of people. Human Activity Recognition is also very much important with respect to scientific advancements in areas such as human-computer interaction, healthcare, sports and surveillance [15].

Using the surveillance point of view, Human activity recognition systems can be installed in public places such as airports or railways, bus stations, banks, parks, and many more. It is possible to determine if any malicious tasks which are executed by a person or group of people in such public places can be instantaneously notified to the concerned authorities for safety reasons. This can also be extended to be used in traffic cameras for traffic rules violations as well.

Human activity recognition can be very much essential in healthcare as well as it can be used to monitor numerous things among patients where nurses need not be present. For example, in case of fractures, patients are not allowed to scratch the wounded region and sometimes nurses have to monitor them for some time, or sometimes nurses do not monitor as well but this system can help them to prevent such minor issues and nurses may be present at areas where they are much more needed.

Also, in the case of ICUs, one nurse is allocated to one patient to monitor but some of this can be automated for the nurses to look after other patients who need much more help. In the case of Covid-19, healthcare staff had to make sure no one took their masks or gloves with constant monitoring but this could have been avoided if the systems were automated.

Another application of HAR resides in sports regions. In most sports, if someone is at the beginning of the learning procedure, the coach needs to monitor the posture during playing and also during exercises as well. But now that the automation procedure is in the works, coaches could take monitoring lightly and put their major focus on learning the game and getting better at weak areas. Below is a picture of human activity recognition applications in sports for postures detection.

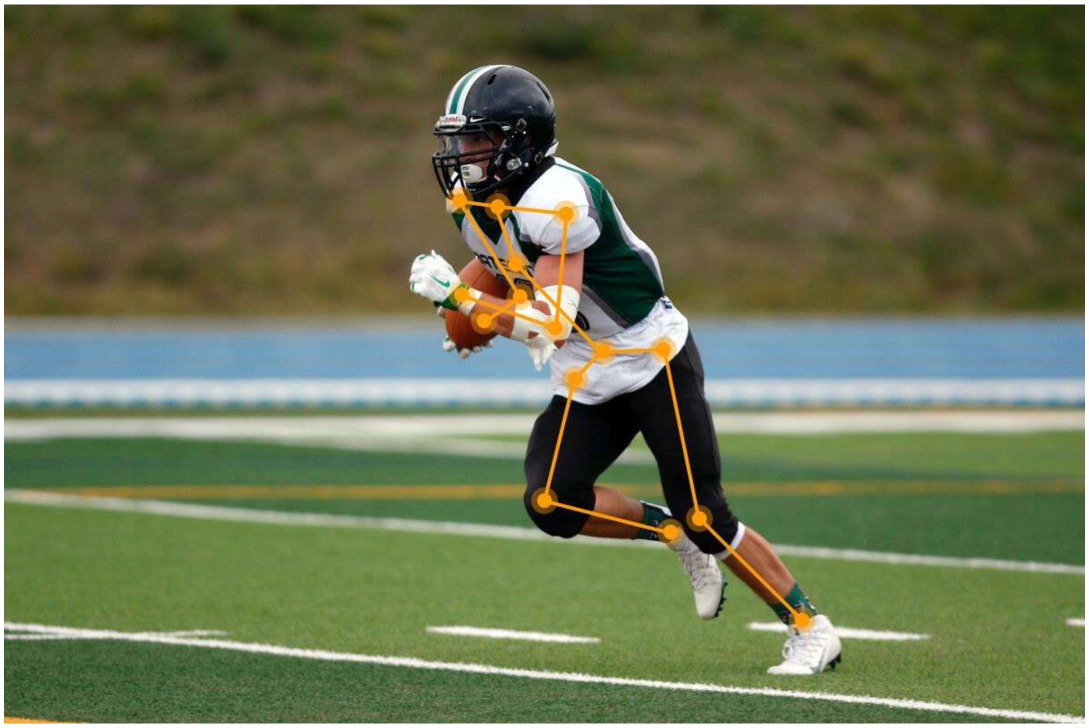


Fig. 1.1-Detection of postures in Sports [16]

Human activity recognition has another vast usage in human-computer interaction applications such as in gaming fields mostly in virtual reality games as they provide a real-life-like environment to the gamer which is far better than the conventional gaming which the user is used to. This sort of gaming is extremely addictive and keeps the users engaged for a very long time. Many multi-million corporations are making huge profits with such technologies.

1.4 Basic Available Algorithms

As the name suggests, machine learning provides machines with the ability to learn independently based on previous encounters, current observations, and trends identified in a set of information without having to be explicitly programmed. When we write an application or piece of the programme for a specific purpose, we generate a set of straightforward guidelines that the machine is going to adhere to. Conversely, when using machine learning, we offer a collection of data so that the computer can identify and evaluate the set's statistical characteristics and learn to arrive at judgements on its own grounds based on its findings while gaining knowledge from the data set itself. In general, there are three different categories of ML algorithms:

- ➢ **Supervised Learning**: This algorithm includes making predictions about a goal or outcome variable (also known as a dependent variable) based on a set of predictors (also known as independent variables). A function is constructed that corresponds inputs to the anticipated results using this group of variables. The training method is performed out regularly until the model's reliability on its initial training data attains the intended level. Regression, logistic regression, decision trees, random forests, and KNN are among the examples of supervised learning.

- ➢ **Unsupervised Learning**: No target or outcome parameter for foreseeing or assessment is present in this approach. It is frequently utilised to split customers into different groups for targeted interventions by clustering the population into those categories. Examples involve K-means and the apriori algorithm.

- ➢ **Reinforcement Learning**: This algorithm instructs the computer to perform specific tasks. It functions as follows: a configuration is presented to the machine where it continuously learns by making mistakes. This system learns from the past and tries to acquire the best data it can in order to make accurate business decisions. Markov Decision Process is an example of reinforcement learning.

1.5 Deep Learning (DL)

Deep neural networks are a common name for contemporary neural networks. Despite the fact that multi-layer neural networks have been around since the 1980s, successful training of these networks has been hampered by a number of factors. The plague of dimensionality is just one of the major issues. The variety of possible configurations of the variables expands exponentially as the number of variables rises. The quantity of training samples should rise in proportion to the number of configurations. It takes too much time, money, and energy to gather a training dataset big enough, or it is just not doable.

Computer Vision or DL is a study that helps computers determine differences between images. Some major examples include the classification of PET images or recognition of activities performed by a human and so on. A computer sees an image as an array of two-dimensional matrices divided in RGB or grayscale format. Each pixel is assigned a value related to the color it represents in the image's format. That value is stored as per the pixel color.

There are various challenges faced during a computer vision algorithm as computer vision algorithm needs to be robust upon development in scenarios provided below

- ➢ **Viewpoint Problem**: If we are determining a cat and the camera is moved to a different angle, the output still needs to be a cat only, basically, if the viewpoint changes the output need not change.

- ➢ **Illumination Problem**: Not only just viewpoint, but illumination is another problem where light conditions matter. If we take the same example of a cat, with different light conditions, the cat could have different representations and our algorithm needs to be robust enough to understand the difference among the same.

- ➢ **Deformation**: The position at which the object is present should not be an issue. Considering the same example of the cat, the cat can be seated at any position and our algorithm needs to be smart enough to understand, locate and determine the cat at its position.

- ➢ **Occlusion**: This is the issue where only part of the object is visible maybe due to difficult lighting conditions or the main object being overlapped by another object and many more. In such cases, we need to make sure that our algorithm determines the objects irrespective of the features hidden and the features shown. For example, while identifying the cat hiding behind a person, we need to extract features of only the cat and then start with processing. We also need to take care of things where our important feature information is not corrupted by the person behind whom the cat is hiding.

- **Background Clutter**: This is yet another challenge in computer vision problems where the object we are trying to determine is located at a place where the background is similar to the object, in such scenarios, we need our algorithm to carefully identify the difference between both them and only extract features from areas where the object is present not from the one background as well. For example, in a similar case of the identification of cats, if a cat is placed at a location exactly with the same color as the background, it gets hard for the algorithm to determine features from just the cat and remove the background, we need to be sure that our algorithm is built in such a way that it can handle similar scenarios in utmost all cases.

- **Interclass Variation**: This issue is raised in cases where one notation of objects spans with lots of different appearances. For example, if a cat looks similar to a cub, we need to make sure that classification is handled properly. Possible solutions to this approach would be to classify based on other parameters as well such as shape, size, age, color and many more.

CHAPTER-2
RELATED WORK

This chapter discusses the numerous developments that have been made in this particular field. The below section covers basic ideas and recent trends in Neural Networks and their different types. It also consists of one of the most promising models for Image classification, that is CNN model.

2.1 Neural Networks

The architecture of neural networks is inspired by the working of neural networks in the human brain. The working idea of neural networks revolves around working back and forth around a particular dataset again and again. For example, a human baby learns the difference between oranges and apples by constantly learning from many images of both fruits until he or she can differentiate between the two. The model is trained by constantly learning from different images or points on a dataset. Neural Networks make patterns to identify objects. This is handled by multiple Neurons. A neural network consists of many neurons which provide output for a part of the image which needs to be classified. A Neuron is a function that gives output values ranging from 0 to 1. For example, if we have a classification problem to determine whether the provided image belongs to an apple or an orange, then the image that needs to be classified is analyzed pixel by pixel, each pixel can be determined as a neuron and return the output of specified part. This section follows binary classification, if the output value is between 0 to 0.5, it's an orange or 0.51 to 1 then it's an apple.

Neurons are also known as Activation Functions, and there are three stages in each neural network:

➢ Input Layer

➢ Hidden Layer

➢ Output Layer

The input and output layers are used to provide input and output data to the function. The hidden layers consist of a set of layers for shape, size, color, etc. Determining the number of hidden layers required for this is dependent on the data, the more layers, the more significant would be the calculation time and effort. Each connection between every pair of neurons is called a weight and the determination of how high our weighted sum should be is given by a bias. If the bias is negative, then the weighted sum is less, then the activation function receives the weighted sum after that and returns a suitable output value.

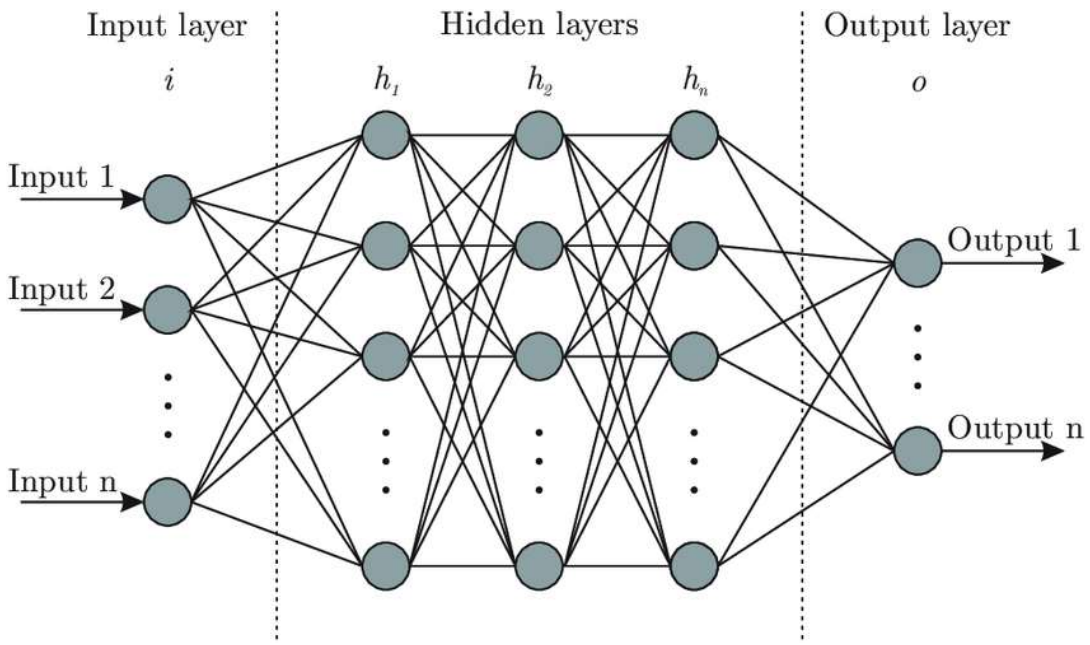


Fig. 2.1 Artificial Neural Network [19]

Types of neural networks:

➢ Perceptron
➢ Feed-Forward Neural Network or Multi-Layer Perceptron (MLP)
➢ Convolutional Neural Networks (CNNs)
➢ Recurrent Neural Networks (RNNs)

2.2 Convolutional Neural Network for Human Activity Recognition

The history of CNN lies in the biological view of the visual cortex. It suggests that the response of a particular cell depends on which part of the visual area is being shown to the person. Hubel and Wiesel, in 1962, took it upon themselves to explain this phenomenon. They pointed out that some neuron cells of the brain get activated when edges are shown in a particular direction. For example, a cat was taken to demonstrate how this pans out, the cat was shown multiple images and how its brain responds to various images, and the data was recorded. It was seen that some brain cells had more sensitive responses to horizontal lines, some had more responses to vertical lines and some to diagonal lines. This core idea formed the basis for Convolution Neural Network (CNN). Initially, CNN was used for handwriting recognition but over the years its applications have drastically increased such as pedestrian detection, behavior and posture recognition. And now, these are being used for Natural Language Processing (NLP) and Speech Recognition as well [7].

Convolution layers are a set of hidden layers that help us determine the edges from the images. Convnets are general matrix multiplication in one layer at least. A simple kernel or a filter is a two-dimensional matrix with values depending on the operation to be performed. We also have a set of predefined matrices that correspond to a special type of edge detection such as the one given in the example. Suppose our entire image is generated as in a matrix-like structure then we multiply the image with a predefined edge detection matrix, in this case, a kernel or a filter, and in the result, we get a matrix with edges. We have different predefined matrices for different types of edges we need to detect, such as horizontal, vertical or diagonal. In general, suppose the size of the image is (m*m) sized matrix and the filter's dimensions are (n*n) sized matrix, then, following convolution the picture's dimensions gets reduced to (m-n+1)*(m-n+1) size, as shown in the following image.

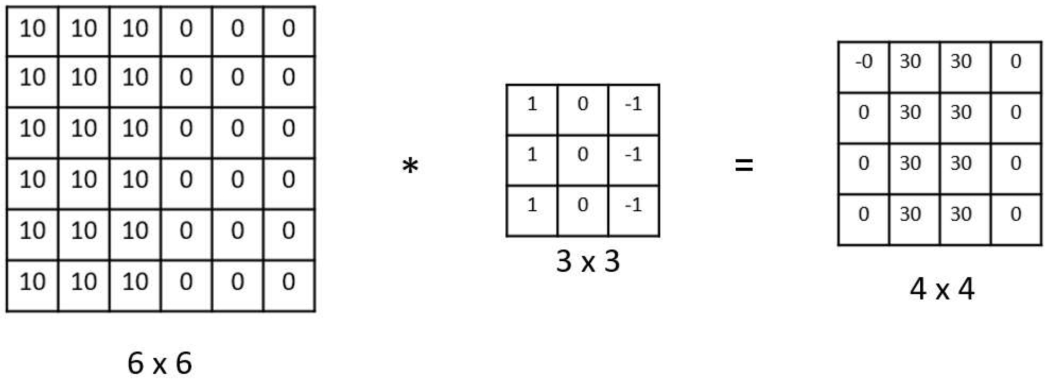


Fig. 2.2 Convolution operation for edge detection [9]

The above matrix is a 6*6 image with a 3*3 vertical edge detector matrix. The result we are getting is a 4*4 image with vertical edges. Also, the formula for m*n-1 is verified here [8]. The matrix here is multiplied one by one, for example, the first 3*3 values in the image are selected and multiplied by the filter, once that has happened, then we shift one position and again perform multiplication for the next 3*3 values in the matrix. This is done until the entire matrix is covered and then we get a result. One of the major issues that convolution operation faces are that it reduces the size of the image after filtration of the process and loss of information based on the image is an issue we need to avoid.

After every convolution operation, the original size of the image matrix gets shrunk with every iteration. In the above example, it went from six to four in the first iteration, in the case of the second iteration, it would have gone from four to two. Another issue is that when the filter iterates over the original image it touches and overlaps the middle section of the image multiple times. Hence, the corner features of the image get lost.

To handle this issue, the concept of padding was brought into action. Padding is required to maintain the original image dimensions. Extra rows and columns of zeros are added in order to preserve the original pixels of the image. Now the convolution formula can be adjusted, assuming, a matrix of size (n*n) is being convolved with (f*f) sized filter (or kernel), then the output image size can be determined by the formula (n+2p-f+1)*(n+2p-f+1) with p as the number of padding we have used. For example, in the given picture, we have a 6*6 matrix, with a filter of 3*3, if we add padding of one layer, it gets reconstructed into an 8*8 matrix and the resultant matrix after convolution operation gets into the original shape of 6*6 [10].

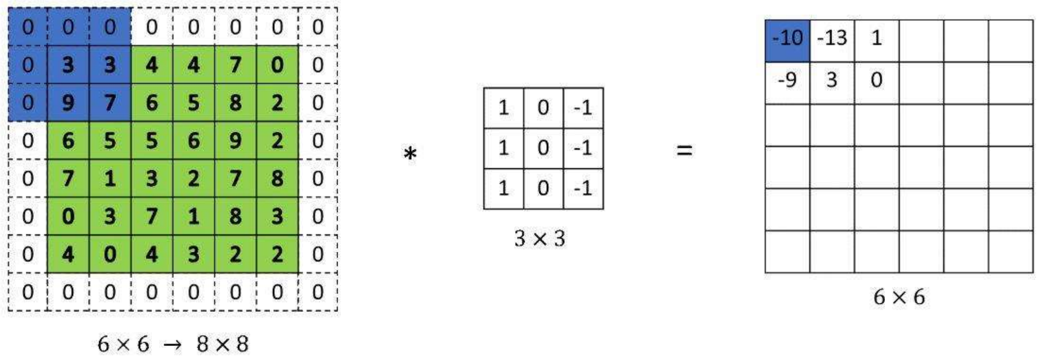


Fig. 2.3 Padding example in CNN [11]

The next step in CNNs consists of Strides. Strides refer to shifts per pixel in our input image matrix. If the strides number is one, then the expected shift per pixel would be one, if it's two, then the pixel shift would be two, and so on. In the shown example image, the strides number is one in the first shift and two in the second shift. Stride is a component useful for compressing images or videos in CNNs. By default, the value of stride is one only. Another advantage of having a stride is that the layer can learn certain extra properties that were unexpected while at the pooling layer. The below image properly explains strides which are shown in red color.

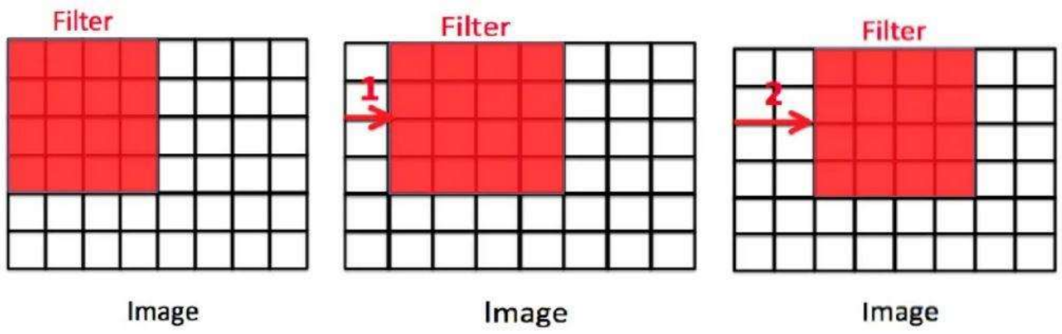


Fig. 2.4 Strides shift from 0, 1, 2 respectively. [10]

If we add padding for p size, in an (n*n) matrix with (f*f) filter or kernel, the expected stride size to be s, then the expected dimensions of the output image matrix would be $[\{(n+2p-f+1)/ s\} + 1]*[\{(n+2p-f+1)/s\}+1]$.

A feature map is the result of the convolution procedure. Two major issues with convolution operations are memory issues and translation variance. Feature map construction and operational size are very high, so we need to perform an operation to optimize memory usage. Another issue that we face is Translation variance and what we need is translation in variance. Feature detection is dependent on the position of the feature. Ideally, if the position changes, the feature need not change.

The next building block in a convolutional neural network is Pooling. Pooling helps the network determine features independently from the location. This also helps in the reduction of size as only required information from the image is stored. Two of the greatest popular methods for pooling are maximum pooling and average pooling. Maximum pooling simply takes the maximum of the region resulting in which we get the most important features of the image. This ensures we get the high-level features of the image and avoid the low-level features. In cases where the background is dark and we need only the lighter part, we can make use of maximum pooling. The below image determines the maximum pooling in a (4*4) matrix resulting in which we get a (2*2) matrix with high-level features from the constructed image.

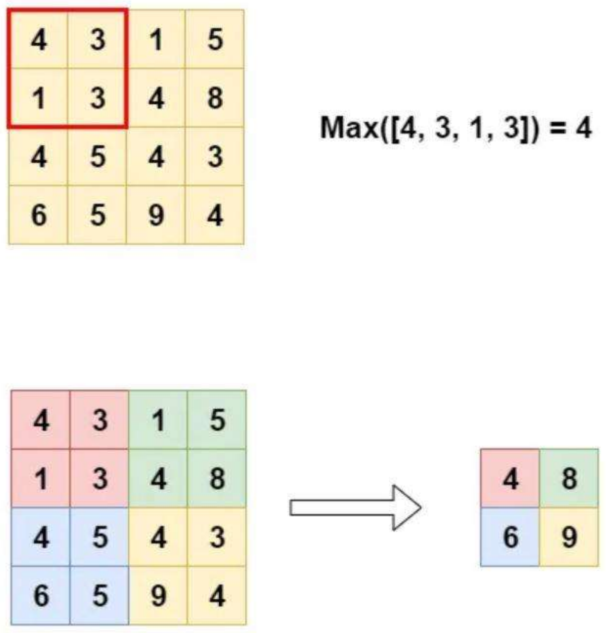


Fig. 2.5 Maximum Pooling [10]

The second one that is mostly followed is average pooling and it stores information from "less important" sections of a matrix. The idea is to find a proper aggregation of all the values in the section and include that in the final matrix. It is different from maximum pooling as maximum pooling rejects all the other information and only accepts the most useful one. The below image shows the proper example of average pooling.

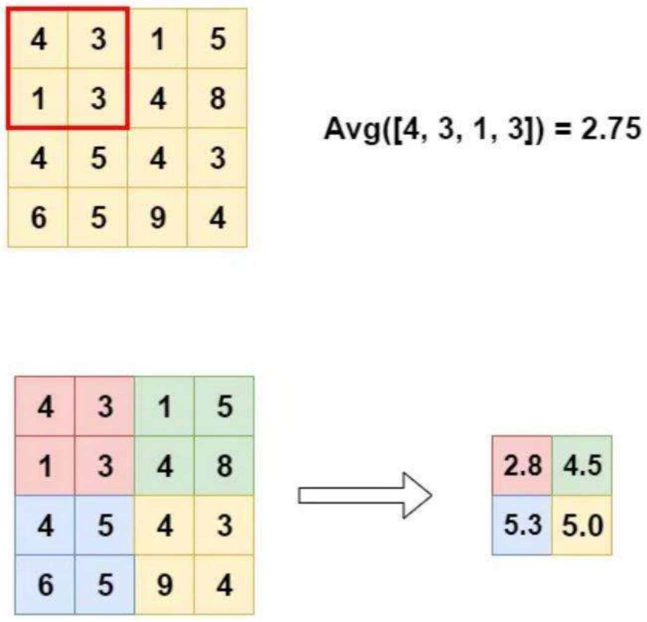


Fig.2.6 Average Pooling [10]

2.3 Recurrent Neural Network for Human Activity Recognition

A neural network is a set of different layers that are connected to each other in a manner such that the input of the second layer is the output of the first layer. A neural network is inspired by the human brain and learns from enormous chunks of data, which in return uses various complex algorithms to train. There are different types of neural networks some of which are already discussed before but namely:

- **Feed-forward neural network**: this is used for basic classical regression and classification problems.

- **Convolution neural network**: this is used to detect objects and classify images.

➢ **Deep belief neural network**: this is used in healthcare departments to detect cancer.

➢ **Recurrent neural network**: used for various applications such as speech and voice recognition, natural language processing and many more.

RNNs operate on the tenet that the output of one layer is stored and used to forecast the output of another layer by sending it back as input. As there were some issues found in the feed-forward neural network, RNNs were created:

➢ It was unable to handle sequential data as a feed-forward neural network needs complete input beforehand, while in the RNN, input is passed one by one and it can also handle sequential data as well.

➢ RNNs can memorize previous inputs unlike feed-forward neural networks to achieve better accuracy. A RNN consists of its own internal memory for the same.

A RNN's initial input layer, let's call it "x," receives input from the network, processes it, and then passes the processed data back to the middle layer. The middle layer, say 'h', consists of multiple hidden layers, depending on the problem we are trying to solve. Each of these layers contains its own weights, biases and activation functions. The RNNs will make each hidden layer equally parameterised by standardizing different weights, biases and activation functions. By doing this, it will not have to create multiple layers, the neural network will just create one layer and loop over it multiple times.

Some major applications of RNN include image captioning problems, time captioning problems such as the prediction of stock prices at a specific month, natural language processing, and machine translation such as translation from English to Hindi language by a model. There are four major types of RNNs namely:

➢ **One to One**: Another name for this network is Vanilla Neural Network and is generally used for problems that require single input and single output. Examples of this could be models that take an image as input and return another image as output.

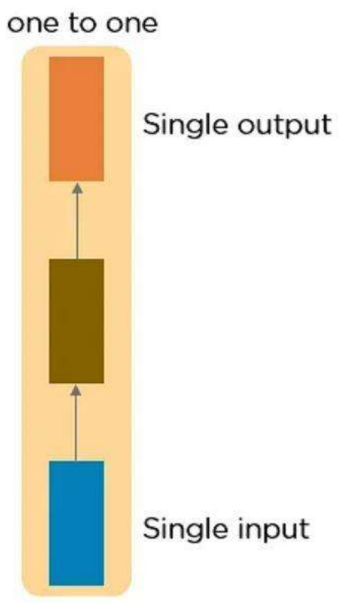


Fig. 2.7 One to One RNN example [13]

➢ **One to Many**: This is another type of neural network that accepts one input as an argument and returns many outputs. One example of this neural network could be an image captioning problem where an image is provided as an input and in output we receive a text field that corresponds to the caption of the image.

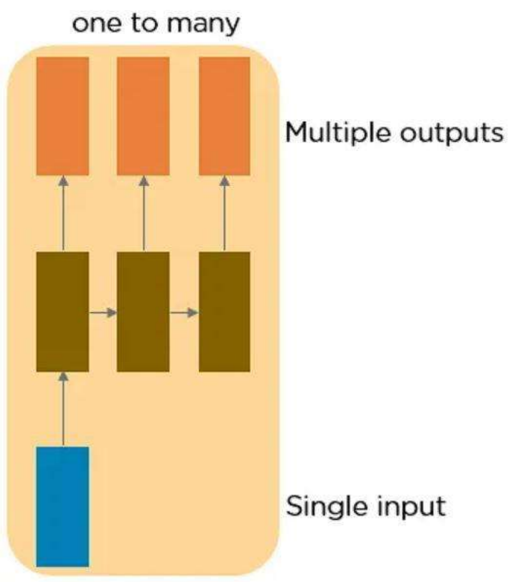


Fig. 2.8 One to Many RNN example [13]

➢ **Many to One**: This is another type of RNN that accepts multiple inputs as an argument and returns only one output. One such example of this category could be the sentiment analysis problem where we provide a complete sentence and we need to return whether the provided sentiment is positive or negative. Another example of many-to-one can be human activity recognition upon using the LSTM approach, we have many frames as input and one class as output which is supposed to be an activity performed by the individual or a set of individuals.

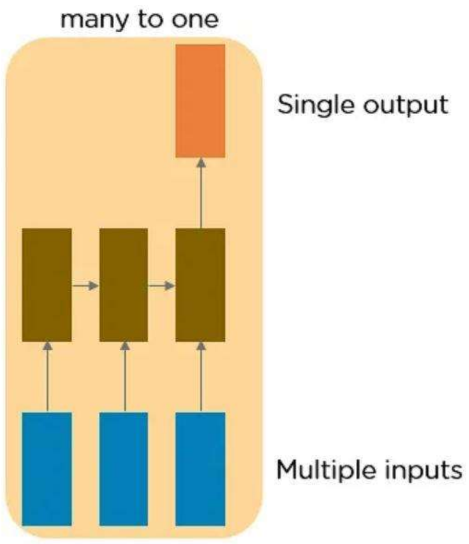


Fig. 2.9 Many to One RNN example [13]

➢ **Many to Many**: This is another type of RNN that accepts multiple inputs as an argument and returns multiple outputs. One such example of this category could be the machine translation problem where we provide a complete sentence and the model needs to return its translation in some other language that is expected by the user who has given the initial sentence.

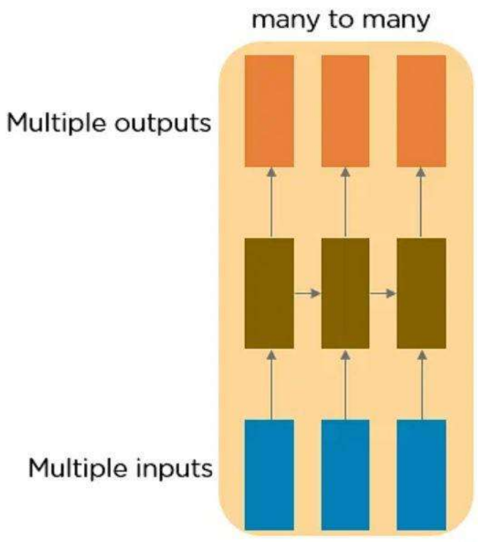


Fig. 2.10 Many to Many RNN example [13]

CHAPTER-3
RESEARCH METHODOLOGY

Following are the basic steps while creating a DL model:

- ➢ Download the dataset

  Download the complete dataset according to the use in proper format.

- ➢ Visualize the dataset with its labels

  Visualize the data along with its label to get a basic idea about the data: how it looks like, what are the different labels, etc.

- ➢ Preprocess the dataset

  Preprocessing of the data is required to make the data consistent. It includes resizing the image, reframing images/videos, normalization of data, image enhancement, etc.

- ➢ Split the dataset into train and test datasets

  The dataset is randomly split between training and testing datasets in some ratio (for eg. 70:30, where 70% data is training data and 30% data is testing data).

- ➢ Implement DL model
  - ○ Construct model

    Create the DL model with an adequate number of neural network layers and cost functions, and loss.
  - ○ Compile and train the model

    Once the model is generated, train the model on training data to increase model learning.
  - ○ Plot the model's loss and accuracy curves

    As the model trains itself on the training data, plot the loss and accuracy curves to test the convergence of the model.

➢ Test performance of the model

Once the model is trained properly, test the model on testing data and find the accuracy and loss of the model.

In the case of more than one DL model, generate the test performance of all the models and do the comparative study.

3.1 Understanding CNN Using Le-Net 5 Architecture

The LeNet-5 architecture of a CNN is a seven-stepped procedure which consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers. The below figure shows a properly structured layout of a LeNet-5 Architecture.

The architecture begins with the first input layer, there is close to nothing to be learnt in this layer and hence, sometimes it's also not considered as the first layer as well, but the input layer is required to take 32*32 images and it is required to pass these dimensions of the image as input to the next layer.

For example, the MNIST dataset consists of images with 28*28 dimensions, but the LeNet-5 architecture can only accept images with dimensions of 32*32, hence, to configure this with the same dimension, the images are padded and then it becomes compatible to work with.

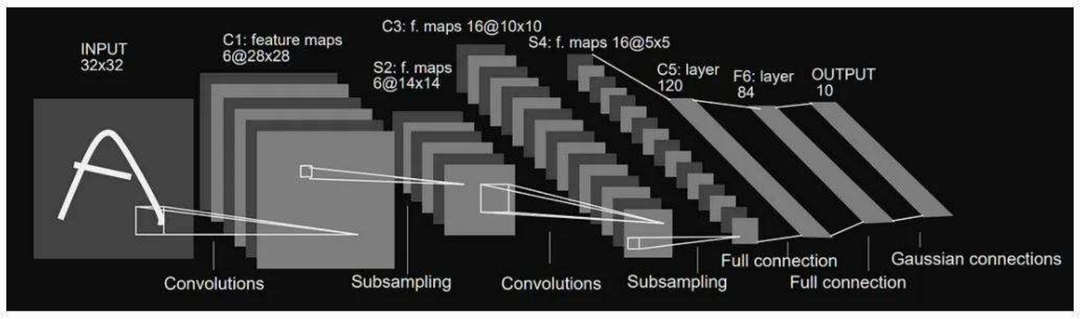
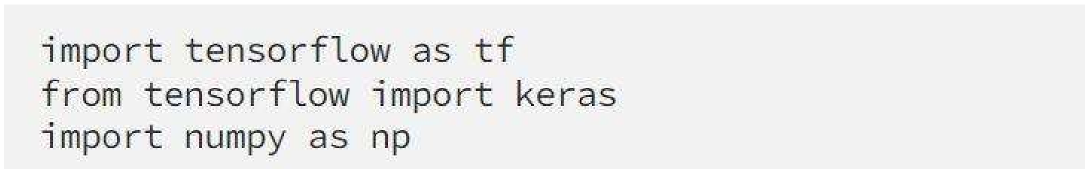


Fig. 3.1 LeNet-5 Architecture for CNN [12]

To reduce the amount of training time required, we can make some modifications such as making sure that the set of images has a mean of 0 and a standard deviation of 1. So, we have normalized the pixel value to accept something between 0 and 1 only.

The convolution layer outputs 6 feature maps and has a kernel or filter of size 5*5. A kernel or filter is a matrix when multiplied with an image, extracts the features of the image or finds the similarities. The feature maps of the first convolution layer produce images of 28*28 dimensions.

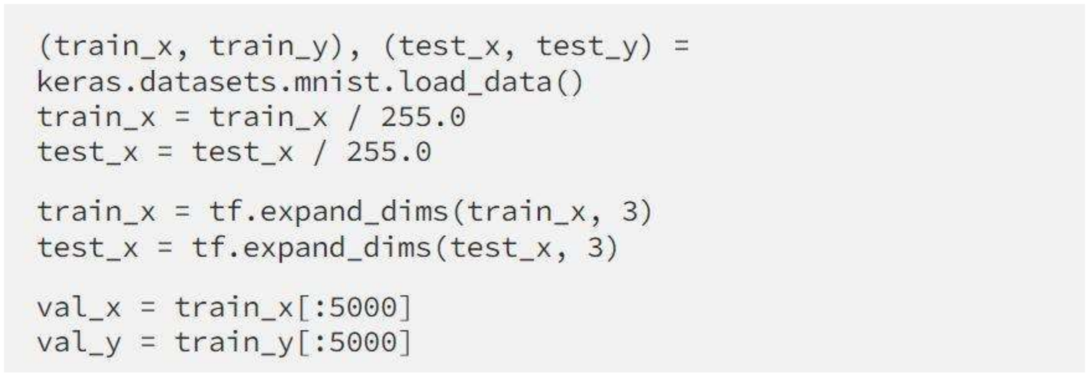
Six feature maps are also produced by subsampling layer 2 in which every map is passed as an input to the next layer. Let us now move to the implementation of the LeNet-5 architecture using TensorFlow, Keras and Numpy. Below are the import statements of the same.



```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

Fig. 3.2 Import statements for LeNet-5 Architecture [12]

Now, since we have imported keras, keras consists of a set of datasets for implementation purposes. We can load the MNIST dataset for the same. Also, we have to make different segments on our datasets for training the data, validation of data that is required while we are training the model, and finally, testing of data which is used to determine the actual accuracy of the model. Simultaneously, we need to make sure that the pixel range of our dataset is not in between 0 and 255, and it must be between 0 and 1 only for simplicity purposes.



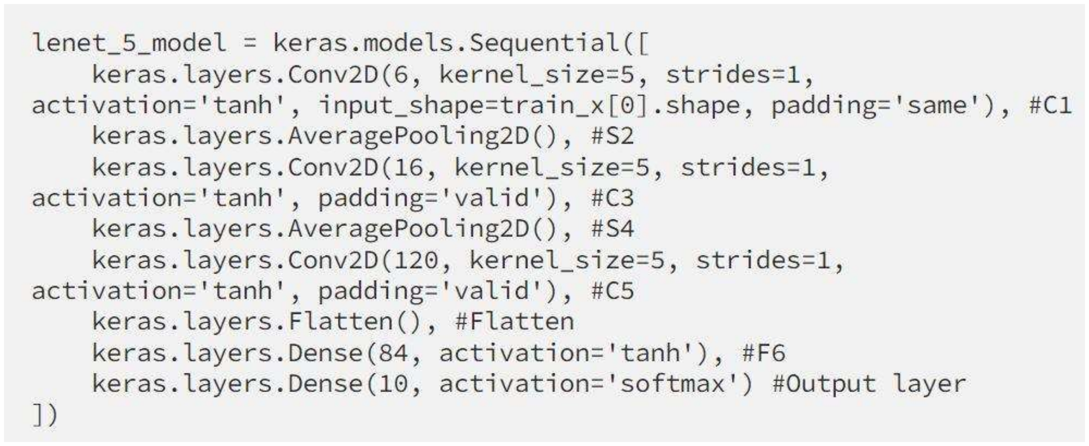
```
(train_x, train_y), (test_x, test_y) =
keras.datasets.mnist.load_data()
train_x = train_x / 255.0
test_x = test_x / 255.0

train_x = tf.expand_dims(train_x, 3)
test_x = tf.expand_dims(test_x, 3)

val_x = train_x[:5000]
val_y = train_y[:5000]
```

Fig. 3.3 Segmentation of dataset and trimming range for LeNet-5 Architecture [12]

In the above code, we have partitioned the dataset into training and testing for x and y. Also, we have made sure that the color range is in range 0 to 1 by dividing them by 255. Next, we have expanded the dimension of training and testing. We expand the dimensions in order to make the images compatible with the network. Here, Kears provide all the necessary methods and tools to implement this particular classification model.



```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(6, kernel_size=5, strides=1,
activation='tanh', input_shape=train_x[0].shape, padding='same'), #C1
    keras.layers.AveragePooling2D(), #S2
    keras.layers.Conv2D(16, kernel_size=5, strides=1,
activation='tanh', padding='valid'), #C3
    keras.layers.AveragePooling2D(), #S4
    keras.layers.Conv2D(120, kernel_size=5, strides=1,
activation='tanh', padding='valid'), #C5
    keras.layers.Flatten(), #Flatten
    keras.layers.Dense(84, activation='tanh'), #F6
    keras.layers.Dense(10, activation='softmax') #Output layer
])
```

Fig. 3.4 Keras Section for LeNet-5 Architecture [12]

Here, the variable ‘lenet_5_model’ is an instance of a constructor from TensorFlow.keras.sequential class. Within the constructor, we are providing the kernel size or the filter size as 5, strides or the shift per pixel as 1 because we need not miss any information from the image. Next, we have put padding as same, when the padding is the same then the convolution layer makes sure that the output matrix size is the same as the input matrix size. This makes sure that all the elements of the input matrix image pass through the filter.

The Activation function is used to determine whether a particular neuron needs to be activated or not. This is mathematically calculated by a weighted sum of each neuron and the addition of bias to it. The activation function adds non-linearity to the output of a neuron. We need the output to be non-linear because a neural network if linear is just another linear regression model and hence, to avoid that we use activation functions. Next, other layers follow the same layer definitions as the first convolution layer.

In our implementation, we are utilizing keras, an average pooling constructor. We normally refrain from passing any arguments to the parameterised constructor as we need the default values with which the constructor was being called with. In this particular network, we have got two more types of layers named flattened and dense layers.

- ➢ The flattening layer is used to flatten our image matrix which is mostly in two dimensions into a single array that is in one dimension so that it can be passed as an input to other subsequence dense layers. The specific class used here can be found in Tensorflow and keras then layers then flatten.

- ➢ The final dense layer consists of 10 that map to different classes from the MNIST dataset, the resulting layer is a softmax activation function.

Now, we need to compile and build the model we have prepared. Now, in keras we are also provided with a built-in method to compile the object of the model we have built, it makes sure all the properties of the model we justified are satisfied as well as it also handles some extra options such as loss function, optimizer, and metrics.

In order to train the network, we need to calculate the difference between the actual values of the training data and predicted values from the model and this is how the loss function gets calculated.

```
lenet_5_model.compile(optimizer='adam',
loss=keras.losses.sparse_categorical_crossentropy, metrics=
['accuracy'])
```

Fig. 3.5 Compilation of LeNet-5 Architecture [12]

The number of changes made to the weights inside of a network determines the loss function by an optimization algorithm. We then try to make the loss value as close to zero as possible. During training, after every epoch, we need to validate our model by the valuation of the dataset partition which was created earlier.

```
lenet_5_model.fit(train_x, train_y, epochs=5, validation_data=(val_x,
val_y))
```

Fig. 3.6 Model Fitting of LeNet-5 Architecture [12]

The above code is used to perform fitting in LeNet-5 architecture.

```
lenet_5_model.evaluate(test_x, test_y)
>> [0.04592850968674757, 0.9859]
```

Fig. 3.7 Model Evaluation of LeNet-5 Architecture [12]

We need to test our model with unseen data that was stored in the test dataset. After training the model, we can clearly understand that the model has achieved 98.59% of accuracy in our test dataset which is very much useful for such simple and similar networks.

3.2 How can we modify CNN to enhance Human Activity Recognition

CNN is one of the best Architecture out there to work with images data. But, in the case of human activity recognition we are working with video data, and we have got multiple approaches to resolve this problem. One such approach is known as the Slow Fusion approach.

The goal of the branch of research known as "human activity recognition" (HAR) is to create intelligent systems that can automatically recognise and categorize human actions based on sensor data. Applications for HAR include surveillance systems, sports performance analysis, and healthcare monitoring. Convolutional Neural Networks (CNNs) have been effectively used in HAR and have presented to be very effective in understanding visual tasks. In recent years, researchers have investigated several fusion methods for bettering HAR performance, and slow fusion is one probable strategy.

The performance of activity recognition systems can be improved by slow fusion, a fusion technique that combines several modalities or temporal knowledge. When used for human activity recognition, this refers to integrating the spatial and temporal characteristics that have been taken from various sources, such as RGB video frames and optical flow pictures, into a single CNN architecture.

The slow fusion method evaluates each modality independently before merging the features that have been acquired. It functions on a slower timeline. This method upgrades recognition accuracy by enabling the network to more smoothly capture physical location specifics and temporal dynamics of human behavior.

The slow fusion approach involves these steps for human activity recognition:

➢ **Input Processing**: Multiple modalities, including RGB frames and optical flow pictures, make up the input data. To extract pertinent information, each modality underwent a separate preprocessing stage. While optical flow images can be analyzed using methods like Farneback or Lucas-Kanade to retrieve motion-related information, RGB frames can be fed into a pre-trained CNN for spatial feature extraction.

➢ **Feature Extraction**: The CNN architecture is then utilized to extract higher-level characteristics from each of the preprocessed independent modalities. Convolutional layers are constructed on the input information in this step to detect spatial patterns and temporal dynamics.

➢ **Modality-Specific Learning**: Following that, various branches or sub-networks process the features that were taken from each modality. By adding extra layers, such as fully connected or recurrent layers, these branches learn representations particular to each modality. This permits the precise properties necessary for activity recognition to be coded for each modality.

➢ **Temporal Fusion**: Following modality-specific learning, fusion techniques are used to merge the features from several branches. Concatenating the features and feeding them into the final fusion layer is one typical strategy. This layer combines data from several modalities and develops the ability to distinguish between various actions.

➢ **Classification**: At last, in order to forecast activity, the fused features are fed through a classifier, such as a fully connected layer or a RNN. Backpropagation and gradient descent is used to optimize the network parameters once the classifier is trained using labeled activity data.

Implementing a slow fusion technique will ensure us with better performance and improvised accuracy but there are other challenges that come with this approach, some of which are listed below:

➢ **Computational Complexity**: It can be computationally demanding to process many modalities and combine their features, necessitating a lot of time and resources.
➢ **Data Synchronization**: It might be difficult to align data from various modalities in terms of timestamps, especially when working with enormous data sets. Reliable synchronization is critical to ensure temporal fusion [17].

Due to these issues, it gets difficult to proceed with slow fusion methodology and hence, here we have another approach to handle similar scenarios. In the new approach, we use Long Short Term Memory (LSTM) along with CNN. Usually, our datasets consist of various sets of videos and labels assigned to them, so we need to work with videos rather than images. But, a video is also a set of frames or images that is compiled together.

The idea is to take each of these frames or images and determine the activity performed in every case and return the classification that is occurred maximum. As we are aware that

CNN is one of the best architectures out there to work with image classification, it becomes necessary to utilize it. But taking the maximum occurred classification may fail in various instances such as standing up can be determined as sitting down or vice versa, hence, we need to supply it with some memory which would enhance the performance of the architecture.

CHAPTER-4
RESULTS ANALYSIS

To achieve close to zero or no feature mapping, it is recommended to use RNNs with LSTM cells. The neural network can now receive data directly and act like a black box in order to model the problem properly. Various different human activity recognition datasets require an enormous amount of effort and processing for feature engineering, the approach being used here is anyways simpler.

A RNN architecture called LSTM was developed in order to fix the deficiencies of standard RNNs in determining and safeguarding long-term associations in sequential input from consumers.

4.1 Why use LSTM?

The "vanishing gradient" problem, where the gradients exponentially decrease as they are backpropagated through time, affects standard RNNs and makes it challenging for the network to learn and remember information from far-off time steps. This issue is addressed by LSTM, which includes a memory cell and the input gate, forget gate and output gate.

An LSTM's memory unit acts as a storage component that has the ability to deliberately keep or forget knowledge over time. The input gate directs the flow of fresh data into the memory cell, assisting the network to prioritize the data that needs to be maintained. The network may eliminate unnecessary or obsolete information by using the forget gate, which decides which data should be removed from the memory cell. The output gate, which also determines which information is output by the LSTM, also controls the flow of information from the memory cell to the following time step.

The LSTMs' potential to cautiously archive and retrieve documentation everywhere lengthy sequences is made achievable by these gating mechanisms, which are managed by sigmoid and element-wise multiplication processes. LSTMs have the ability to efficiently capture and retain long-term dependencies in sequential data by retaining key data while omitting irrelevant or redundant data.

An LSTM's design enables it to recognise patterns that span several time steps and learn and adapt to sequences with variable time delays. Because comprehending long-term dependencies and context is essential for tasks like natural language processing, speech recognition, time series analysis, and human activity recognition, LSTMs are ideally suited for these kinds of applications.

As a result of including a memory cell and gating methods to selectively store, forget, and output information, LSTM is an RNN variation that addresses the vanishing gradient problem and is able to capture long-term dependencies in sequential data.

4.2 Comparison of different techniques for Human Activity Recognition

Serial No.	**Title**	**Dataset**	**Approach**	**Limitation**	**Result**
1	A Hybrid Approach for Human Activity Recognition with Support Vector Machine and 1D Convolutional Neural Network	UCI HAR dataset	1D CNN, random forest, SVM	Low powered IC deployments in real time for wearable devices.	97.71% accuracy
2	Smartphone Based Human Activity Recognition with Feature Selection and Dense Neural Network	UCI HAR dataset	ANN, SVM	Can be improvised in feature selection	95.79% accuracy
3	Implementation of an Anomalous Human Activity Recognition System	UCF 10 Crime	CNN, Adaptive video comprehension	Such events rarely occur in real scenarios and hence performance with regular videos was only measured without anomalies which are why the alarming rate was	This method was better compared to other similar available approaches with respect to accuracy and training time

				decreased.	
4	Activity recognition in Manufacturing : The roles of motion capture and sEMG+inertial wearables in detecting fine vs. gross motion	UCSD MIT Human Motion Dataset	K nearest neighbors, Linear Discriminant Analysis	Stored only arm motion details.	Fine grain motion classification was more difficult with respect to gross activity
5	Human Activity Recognition for Healthcare using Smartphones	UCI	ANN, Multilayer Perceptron	Lowest accuracy was 80% by gyroscope	Highest accuracy was 92% by accelerometer
6	Human Physical Activity Recognition Based on Computer Vision with Deep Learning Model	CAD 60	CNN, Multi layer perceptron	Only the skeleton model was used upon input and hence, there are chances for improvement	81.8% accuracy
7	Deep Learning for Human Activity Recognition: A Resource Efficient Implementation of LowPower Devices	WISDM	CNN, DL, Neural Network, Linear Classifier	Computation time and spectrogram generation were slightly different on different electronic devices.	The lowest accuracy was 95.1%

Table 4.1. Comparison of different HAR activities in last few years [18]

CHAPTER-5
CONCLUSION

Recognition of human action is a challenging subject. Security, augmented reality, and healthcare are a few important uses for the field. These applications can also be extended to a variety of other fields. We also know that one of the most effective feature selection algorithms at the moment is CNN, which is followed by activity categorization. The protection of wildlife is one of the other sectors where activity recognition is applicable. It is possible to train the application to recognise when an assault on an animal is taking place or not. A further use for such a system would be in sports, where it is important for players to maintain accurate postures. Automation is cheaper and less prone to mistakes than always needing a coach to oversee things. In the future, we'll perform an organised review of the literature that includes further research on machine learning and DL methods.

REFERENCES

[1] C. M. Bishop, “Neural networks and their applications,” *Review of Scientific Instruments*, vol. 65, no. 6, pp. 1803–1832, Jun. 1994, doi: https://doi.org/10.1063/1.1144830.

[2] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[3] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, “Deep learning for sensor-based activity recognition: A survey,” *Pattern Recognition Letters*, vol. 119, pp. 3–11, Mar. 2019, doi: https://doi.org/10.1016/j.patrec.2018.02.010.

[4] Z. Sun, Q. Ke, H. Rahmani, M. Bennamoun, G. Wang, and J. Liu, “Human Action Recognition From Various Data Modalities: A Review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2022, doi: https://doi.org/10.1109/tpami.2022.3183112.

[5] T. PlÖtz, “Applying Machine Learning for Sensor Data Analysis in Interactive Systems,” *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–25, Jul. 2021, doi: https://doi.org/10.1145/3459666.

[6] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu, “Deep Learning for Sensor-based Human Activity Recognition,” *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–40, May 2021, doi: https://doi.org/10.1145/3447744.

[7] Q. Zhang, “Convolutional Neural Networks,” *3rd International Conference on Electromechanical Control Technology and Transportation*, 2018, doi: https://doi.org/10.5220/0006972204340439.

[8] P. Choudhari, “Understanding ‘convolution’ operations in CNN,” *Analytics Vidhya*, May 21, 2020. https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4 (accessed April 24, 2023).

[9] “Edge Detection Convolution Intuition,” *Signal Processing Stack Exchange*. https://dsp.stackexchange.com/questions/76405/edge-detection-convolution-intuition (accessed April 4, 2023).

[10] A. K. Pandey, “Convolution, Padding, Stride, and Pooling in CNN,” *Analytics Vidhya*, Jan. 24, 2021. https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-

13dc1f3ada26#:~:text=Stride%20is%20the%20number%20of,%2F%20%F0%9D%91%A0%7D%20%2B%201%5D (accessed March 2, 2023).

[11] datahacker.rs, “#004 CNN Padding,” *Master Data Science*, Nov. 01, 2018. https://datahacker.rs/what-is-padding-cnn/ (accessed May 1, 2023).

[12] R. Alake, “Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning),” *Medium*, Jun. 25, 2020. https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342 (accessed May 20, 2023).

[13] “Recurrent Neural Network Tutorial,” *Simplilearn.com*. https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn (accessed March 4, 2023).

[14] S. Ranasinghe, F. Al Machot, and H. C. Mayr, “A review on applications of activity recognition systems with regard to performance and evaluation,” *International Journal of Distributed Sensor Networks*, vol. 12, no. 8, p. 155014771666552, Aug. 2016, doi: https://doi.org/10.1177/1550147716665520.

[15] O. C. Ann and L. B. Theng, “Human activity recognition: A review,” *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*, Nov. 2014, doi: https://doi.org/10.1109/iccsce.2014.7072750.

[16] A. Zharovskikh, “Human Activity Recognition: Everything You Should Know,” *InData Labs*, Mar. 31, 2022. https://indatalabs.com/blog/human-activity-recognition (accessed Feb 26, 2023).

[17] M. Ehatisham-Ul-Haq *et al.*, “Robust Human Activity Recognition Using Multimodal Feature-Level Fusion,” *IEEE Access*, vol. 7, pp. 60736–60751, 2019, doi: https://doi.org/10.1109/access.2019.2913393.

[18] Y. Daga and S. Meena, “Applications of Human Activity Recognition in Different Fields: A Review,” *IEEE Xplore*, Dec. 01, 2022. https://ieeexplore.ieee.org/document/9986408

[19] “Artificial Neural Networks and its Applications,” *GeeksforGeeks*, Jun. 24, 2020. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/ (accessed May 12, 2023).