# ABSTRACT

The project aims to create a video classification system for human actions using deep learning techniques. The dataset includes various actions like running, walking, and handwaving. The system uses a convolutional neural network model and long short-term memory layers to capture spatial and temporal features. The dataset is divided into training and testing sets, with the model trained using the training set. An early stopping mechanism prevents overfitting. The model's performance is evaluated on the test set, and the results are analyzed in terms of accuracy and loss. The final model achieves commendable accuracy, demonstrating its ability to recognize and classify human actions within video sequences. This project offers insights into deep learning models for video classification tasks and can be extended for real-world applications.

**CONTENTS**

[ABSTRACT 1](#_Toc159859950)

[CHAPTER 1 4](#_Toc159859951)

[**INTRODUCTION** 4](#_Toc159859952)

[Problem Statement 5](#_Toc159859953)

[Background 5](#_Toc159859954)

[CHAPTER 2 16](#_Toc159859955)

[**LITERATURE REVIEW** 16](#_Toc159859956)

[CHAPTER 3 19](#_Toc159859957)

[**METHODOLOGY** 19](#_Toc159859958)

[Model Structure 19](#_Toc159859959)

[Flow chart Architecture 29](#_Toc159859960)

[CHAPTER 4 30](#_Toc159859961)

[**FINDINGS** 30](#_Toc159859962)

[Model Summary 30](#_Toc159859963)

[Plotting Loss & Accuracy 32](#_Toc159859964)

[Testing on videos 36](#_Toc159859965)

[CHAPTER 5 37](#_Toc159859966)

[**DISCUSSION** 37](#_Toc159859967)

[CHAPTER 6 41](#_Toc159859968)

[**CONCLUSION** 41](#_Toc159859969)

[**REFERENCES** 42](#_Toc159859970)

**LIST OF FIGURES**

[Figure 1 CNN 6](#_Toc159587658)

[Figure 2 LAYERS IN CNN 7](#_Toc159587659)

[Figure 3 Pooling 7](#_Toc159587660)

[Figure 4 activation function 8](#_Toc159587661)

[Figure 5 loss function 9](#_Toc159587662)

[Figure 6 xception 11](#_Toc159587663)

[Figure 7 time distributed layer 14](#_Toc159587664)

[Figure 8 LSTM 15](#_Toc159587665)

[Figure 9 Two-stream architecture for video recognition 16](#_Toc159587666)

[Figure 10 Long-term Recurrent Convolutional Networks 17](#_Toc159587667)

[Figure 11 Temporal 3D ConvNet 18](#_Toc159587668)

[Figure 12: model structure 19](#_Toc159587669)

[Figure 13 KTH DATASET 20](#_Toc159587670)

[Figure 14 flow chart Architecture 29](#_Toc159587671)

[Figure 15 plotting loss and accuracy of Xception - lstm 32](#_Toc159587672)

[Figure 16 PLOTTING LOSS AND ACCURACY OF resnet50 - LSTM 33](#_Toc159587673)

[Figure 17 PLOTTING LOSS AND ACCURACY OF VGG19 - LSTM 34](#_Toc159587674)

[Figure 18 PLOTTING LOSS AND ACCURACY OF INCEPTIONV3 - LSTM 35](#_Toc159587675)

# CHAPTER 1

## **INTRODUCTION**

Human action recognition from video data is a significant task in computer vision with numerous applications such as surveillance, human-computer interaction, and activity monitoring. With the increasing availability of video data, automated methods for understanding human actions have become increasingly important.

In this project, we focus on the task of human action recognition using deep learning techniques. Specifically, we leverage a combination of convolutional neural networks (CNNs) for spatial feature extraction and recurrent neural networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, for capturing temporal dependencies within sequences of frames. Our approach aims to effectively model both spatial and temporal aspects of human actions, enabling accurate recognition even in complex scenarios.

To facilitate this task, we utilize the Xception architecture as a feature extractor for individual frames, followed by an LSTM network to model the temporal dynamics across frames within a sequence. The dataset consists of video clips categorized into different action classes such as running, walking, and handwaving. We preprocess the videos to extract sequences of frames, normalize them, and construct a suitable input format for the neural network model.

Our model is trained on a portion of the dataset and evaluated on a held-out test set to assess its performance in recognizing human actions. We utilize metrics such as accuracy to measure the model's effectiveness in correctly classifying actions from unseen video clips.

In summary, this project presents an approach to human action recognition using deep learning techniques, aiming to contribute towards more accurate and efficient automated analysis of video data in various real-world applications.

### Problem Statement

This research aims to develop a deep learning-based action recognition system for real-time human activity monitoring using video data. The primary objective is to accurately classify and recognize different activities, such as **walking, running,** and **handwaving**, from video streams.

**OBJECTIVES**

1. Collect and preprocess a dataset of human activity data, including video recordings.
2. Design and implement deep learning models, such as Convolutional Neural Networks (CNNs) with recurrent layers, for action recognition.
3. Evaluate the performance of the proposed models using standard evaluation metrics.
4. Compare the performance of the used pretrained models with other models for human activity recognition.

### Background

This chapter explains the fundamental background that was used to conduct this research.  
 **CONVOLUTIONAL NEURAL NETWORK**

Information flows into the Convolutional Neural Network (CNN) via the input layer, traverses through multiple hidden layers, and ultimately reaches the output layer. The network's output is then assessed against the actual labels, measuring loss or error. To refine the model, the partial derivatives of this loss concerning the trainable weights are computed. These weights are then adjusted using backpropagation, employing diverse methods for optimization. [1]

A diagram of a process flow

Description automatically generated

Figure 1 CNN

**CNN TEMPLATE:**

Many commonly utilized hidden layers adhere to a specific structure:

**Layer Function:**

**a) Fully Connected:** These layers establish linear relationships between input and output.

**b) Convolutional Layers:** Designed for 2D (or 3D) input feature maps, these layers employ trainable weights represented by a 2D (or 3D) kernel/filter. The kernel traverses the input feature map, computing dot products with overlapping regions to generate output.

**c) Transposed Convolutional (Deconvolutional) Layer:** Primarily employed for augmenting the size of output feature maps (up sampling), this layer aims to reverse (albeit not precisely) the effects of a convolutional layer.

A diagram of a graph

Description automatically generated with medium confidence

Figure 2 LAYERS IN CNN

**Pooling:**

Pooling serves as a non-trainable layer aimed at altering the size of the feature map:

**a) Max/Average Pooling:** This operation reduces the spatial dimensions of the input layer by selecting the maximum/average value within a receptive field defined by the kernel.

**b) Unpooling:** Another non-trainable layer, unpooling, is utilized to expand the spatial dimensions of the input layer. It achieves this by placing the input pixel at a specified index within the receptive field of the output, as defined by the kernel**.**

A table with numbers and output

Description automatically generated with medium confidence

Figure 3 Pooling

**Normalization:**

Normalization is commonly applied just before activation functions to prevent unbounded activations from excessively inflating output layer values:

**a) Local Response Normalization (LRN):** This non-trainable layer square-normalizes pixel values within a local neighborhood of a feature map.

**b) Batch Normalization:** Unlike LRN, batch normalization is trainable. It normalizes data by learning scale and shift variables during training.

**Activation:**

Activation functions introduce non-linearity, enabling CNNs to effectively map complex, non-linear relationships:

**a) Non-parametric/Static functions:** Examples include linear and ReLU.

**b) Parametric functions:** ELU, tanh, sigmoid, and Leaky ReLU fall into this category.

**c) Bounded functions:** Functions like tanh and sigmoid are bounded within specific ranges.

A group of graphs showing the function of a function

Description automatically generated

Figure 4 activation function

**Loss function:**

The loss function quantifies the disparity between CNN predictions and actual labels:

**a) Regression Loss Functions**: MAE, MSE, and Huber loss are commonly used for regression tasks.

**b) Classification Loss Functions:** Cross entropy and Hinge loss are typically employed for classification tasks.

A group of graphs with numbers

Description automatically generated

Figure 5 loss function

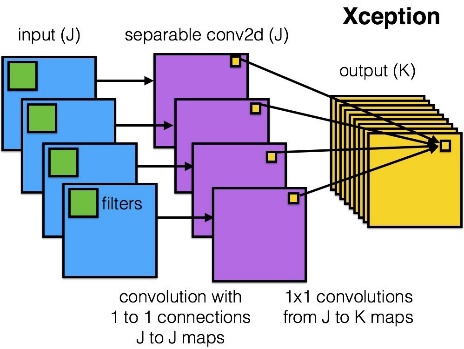
**Pre-trained a Neural Network models**

Pre-trained neural network models are trained on one task and used in another, taking an original model and dataset. They're generally used in transfer literacy, point birth, and bracket for transfer literacy, point birth, and bracket. Pre-trained models, specifically designed for image brackets, are available for colorful tasks, similar as relating the meaning of an image. These models, trained on large datasets, can be applied to any image bracket task, making them suitable for colorful bracket tasks. Pre-training offers ease of use, quick optimization, and lower data conditions compared to erecting a model from scrape. It allows for faster performance and better understanding of parameters, making it a precious tool in machine literacy. Keras operations are deep literacy models with pre-trained weights for vaticination, point birth, and fine- tuning. Weights are automatically downloaded when expressing a model and stored at keras/ models/. Models are erected according to the image data format set in the Keras configuration train at keras/keras.json, similar as" Height- range- Depth". ( https://keras.io/api/applications/ )

**Xception model**

Xception, a 2016 variant of the GoogLeNet architecture, significantly outperformed Inception-v3 on a large vision task. It merges GoogLeNet and ResNet ideas but replaces inception modules with a depth wise separable convolution layer. This layer, previously used in MobileNets, is not as central as in Xception, which uses a special type of layer called separable convolution. [2]

**Depthwise Separable Convolution:**

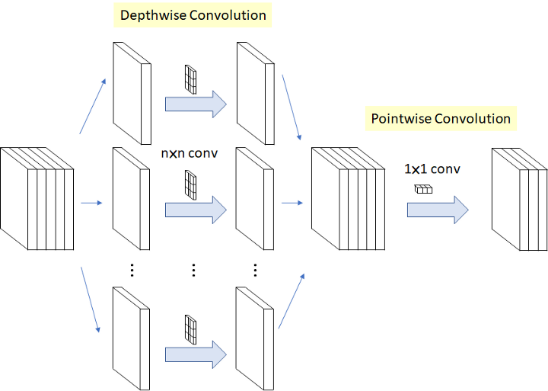


**1. Regular Convolutions:**

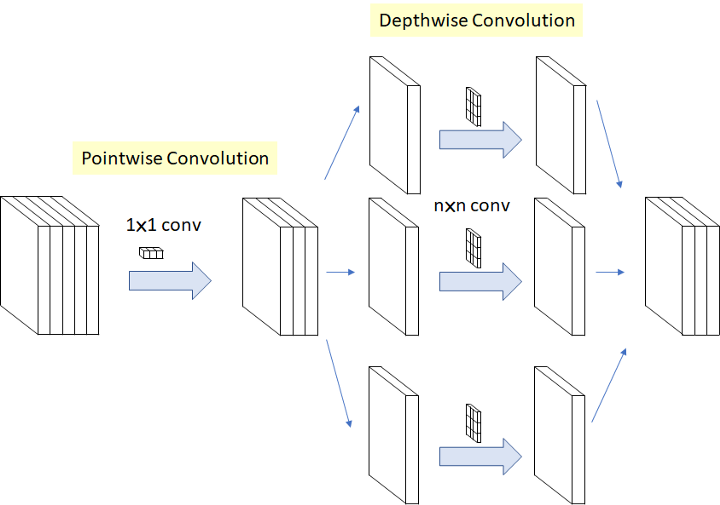
* look at both channel & spatial correlations simultaneously.

**2. Depth wise separable convolution:**

* look at channel & spatial correlations independently in successive steps.
* spatial convolution: 3x3 convolutions for each channel.
* depth wise convolution: 1x1 convolutions on concatenated channels.



**3. Modified Depth wise Separable Convolution in Xception :**



The adjusted depthwise separable convolution comprises a pointwise convolution followed by a depthwise convolution. In essence, it utilizes a pointwise convolution before executing the depthwise convolution. This adjustment is driven by the efficiency and effectiveness demonstrated by the inception module in optimizing convolutional operations.

Separable convolutions offer several advantages over regular convolutional layers, including reduced parameters, memory usage, and computational overhead. In many scenarios, they even outperform traditional convolutions. Therefore, it is advisable to consider employing separable convolutions as the default choice, except perhaps after layers with a small number of channels where their benefits may be less pronounced.

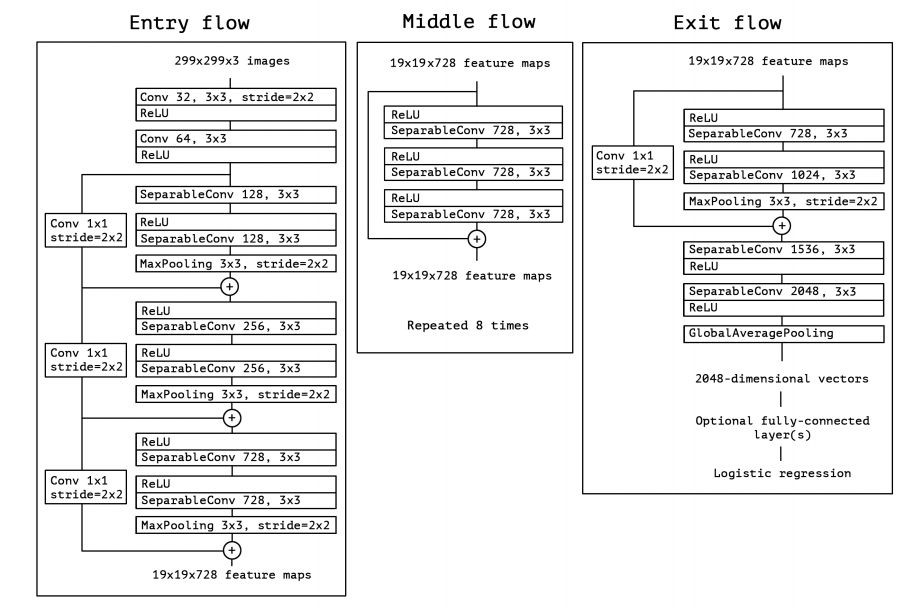


Figure 6 xception

Xception, short for "Extreme Inception," takes the core principles of the Inception architecture to an extreme level. While Inception employs 1x1 convolutions to compress the input data, Xception reverses this process. Instead of compressing the input space first, Xception applies filters to each depth map individually and then compresses the input space using a 1x1 convolution across the depth. This approach closely resembles a depthwise separable convolution, a technique utilized in neural network design as early as 2014. [3] One notable difference between Inception and Xception lies in the presence or absence of non-linearities after the initial operation. In the Inception model, both operations are succeeded by ReLU non-linearities, while Xception omits any non-linearity at this point. The Xception architecture consists of three main sections: the entry flow, the middle flow (repeated eight times), and the exit flow. Each section processes the data in a specific manner to extract meaningful features. Xception is implemented using the TensorFlow framework by Google and is trained on a powerful infrastructure consisting of 60 NVIDIA K80 GPUs. This extensive computational setup enables efficient training and handling of large-scale datasets, contributing to the effectiveness of the Xception model in various machine learning tasks.

**Recurrent Neural Network (RNN)**

RNNs are commonly used in video classification tasks due to their ability to process sequential data. CNN is used to extract high-level features from the video frames, while the RNN layer processes the temporal information in the sequence of frames. The RNN layer can be a Long Short-Term Memory (LSTM) or a Gated Recurrent Unit (GRU) layer. One of the challenges in training video classifiers is figuring out how to feed the videos to the network. One approach is to save video frames at a fixed interval until a maximum frame count is reached.

A diagram of a machine learning process

Description automatically generated

**Time Distributed data in a neural network**

Machine Learning uses Neural Networks to predict values on complex data, often involving chronological inputs like stock prices or video frames. LSTM is suitable for time-range data, but Time Distributed layer can be used to adapt each input before or after this layer, allowing for more accurate predictions on complex data. [4]

*Here, weights are****trained in the same backward pass****and not separately (because there is only one layer applied to each input). This implies that instead of attempting to "detect a cat in each frame," the model aims to identify a jumping cat by analyzing the "sequence" of frames. It applies the same layer to consecutive inputs, enabling it to recognize patterns over time rather than in isolated frames. Then the weights are tweaked.*

Convolution Neural Networks (C-NN) can be used to classify multiple images in a sequence, detecting movements, actions, and directions. To do this, a "Time Distributed" layer is needed, which can **inject a sequence as input and make predictions based on the sequence's appearance**. Keras and other frameworks can handle this data.

Time Distributed layer apply the**same layer to several inputs**. **And it produces one output per input to get the result in time.**

*“Time Distributed” will share the same weights. If you inject 5 images, the weights are not tweaked 5 times, but only once, and distributed to every blocks defined in the current Time Distributed layer.*

**A diagram of a person riding a bike

Description automatically generated**

**A diagram of a person on a skateboard

Description automatically generatedA diagram of a group of people

Description automatically generated**

**A diagram of a person riding a bike

Description automatically generatedA diagram of a person riding a bike

Description automatically generated**

Figure 7 time distributed layer

**Long Short-Term Memory (LSTM)**

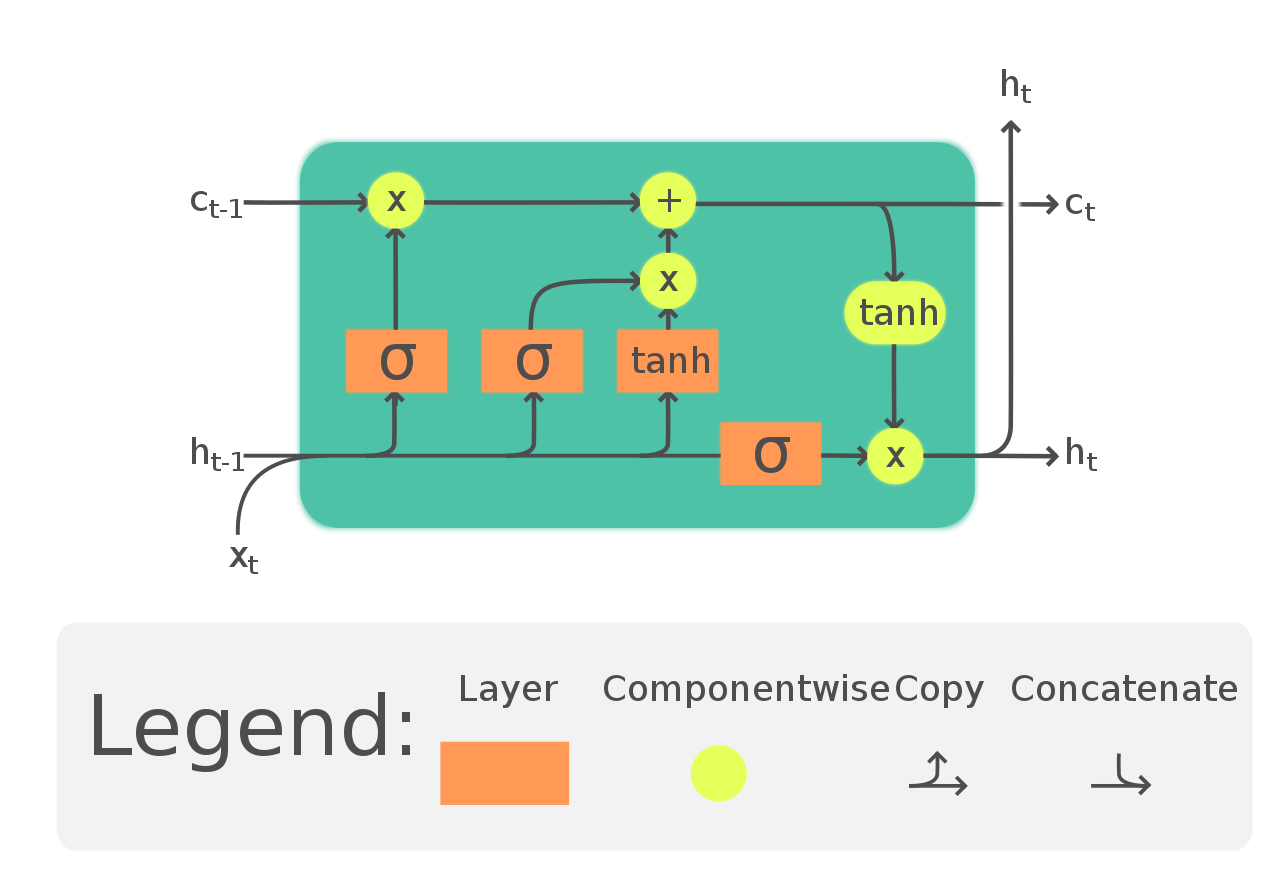
Long Short-Term Memory (LSTM) is a form of recurrent neural network (RNN) renowned for its ability to learn and retain long-term dependencies in data. LSTMs address the vanishing gradient issue inherent in conventional RNNs. They incorporate four distinct gates: the input gate, forget gate, output gate, and input modulation gate. These gates regulate the flow of information into and out of the memory cell, enabling the LSTM to selectively retain or discard information from past time steps. This capability allows LSTMs to effectively capture and utilize long-term dependencies present in the input sequence. LSTMs find applications across a wide range of tasks, including time series prediction, video analysis, handwriting recognition, speech recognition, and sentiment analysis. They stand as one of the most utilized variations of RNNs and play a prominent role in the field of artificial intelligence.

Figure 8 LSTM

# CHAPTER 2

## **LITERATURE REVIEW**

Early human action recognition research used hand-crafted features and conventional machine learning algorithms. These methods struggled to capture the complexity and variability of human actions in real-world scenarios. However, the advent of deep learning, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), has propelled the field. CNNs have demonstrated remarkable performance in image-based action recognition tasks, while RNNs excel at modeling temporal dependencies in sequential data. Recent research has focused on developing deep learning-based approaches, leveraging large-scale annotated datasets, and exploring innovative architectures like 3D convolutional networks.

The paper "Two-Stream Convolutional Networks for Action Recognition in Videos" by Karen Simonyan and Andrew Zisserman. [5] presents a novel approach to action recognition in videos using two-stream convolutional neural networks (CNNs). The authors propose a two-stream architecture that processes spatial and temporal information separately, allowing for more accurate recognition of actions in videos. The spatial stream processes individual frames of the video, while the temporal stream processes optical flow images that capture the motion between frames. The outputs of these two streams are then combined to produce a final prediction for the action being performed in the video.

A screenshot of a video game

Description automatically generated

Figure 9 Two-stream architecture for video recognition

The model proposes a ConvNets-based model for spatial and temporal recognition streams but faces issues with spatial pooling and camera motion handling.

Another paper "Long-term Recurrent Convolutional Networks for Visual Recognition and Description" by Jeff Donahue et al. [6] proposes a novel approach to visual recognition and description using long-term recurrent convolutional networks (LRCNs). The authors introduce a new architecture that combines convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to process both spatial and temporal information in videos. The LRCN model is trained end-to-end to recognize and describe actions in videos.

A diagram of a person running

Description automatically generated

Figure 10 Long-term Recurrent Convolutional Networks

It explores the effectiveness of combining recurrent and convolutional neural networks for tasks involving sequences, such as activity recognition, image captioning, and video description.

Another paper by Ali Diba et al. [7] proposes a new architecture for temporal 3D ConvNets and transfer learning for video classification. The authors introduce a new video classification method called Temporal 3D ConvNet (T3D) that incorporates variable temporal convolution kernel depths and transferring knowledge from pre-trained 2D CNNs to reduce training samples needed. Their goal is to improve action recognition in videos by efficiently capturing both appearance and temporal information without the need for optical flow maps.

A diagram of a computer

Description automatically generated

Figure 11 Temporal 3D ConvNet

The significant advancements made in human action recognition research, particularly with the emergence of deep learning methodologies such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Studies like those by Simonyan and Zisserman [5], Donahue et al. [6], and Diba et al. [7] have demonstrated the efficacy of deep learning architectures in addressing the challenges of capturing spatial and temporal information in videos for accurate action recognition.

# CHAPTER 3

## **METHODOLOGY**

### Model Structure

The approach involves using Time Distributed Convolutional layers and Long Short-Term Memory (LSTM) units to capture sequential data. The model extracts detailed details from each image frame, processing each frame individually while retaining spatial information. The output is fed into LSTM units, which learn temporal dependencies, crucial for understanding chronological patterns. The model captures long-term dependencies, gaining meaningful insights from sequential data. Dense layers transform the LSTM output into the desired classification output, facilitating action identification within video sequences.

A diagram of wrapper model

Description automatically generated

Figure 12: model structure

**Dataset Preparation:**

The dataset consists of videos depicting three different actions: running, walking, and handwaving. Each video is divided into sequences of frames, with a specified sequence length (30 frames in this case). Frames are extracted from the videos, resized to a standard dimension (128x128 pixels), and normalized to values between 0 and 1.

**Dataset:**

The video dataset contains six types of human actions (handwaving, running, and walking) performed several times by 25 different subjects in 4 different scenarios - outdoors \*s1\*, outdoors with scale variation \*s2\*, outdoors with different clothes \*s3\* and indoors \*s4. The videos were captured at a frame rate of 25fps, and each frame was down sampled to the resolution of 160x120 pixels. The dataset contains 300 videos – 100 videos for each of the 3 categories.

A collage of several people in different poses

Description automatically generatedA collage of several people in different poses

Description automatically generatedA collage of several people in different poses

Description automatically generated

Figure 13 KTH DATASET

CODE [8]

Import Libraries

****

**Configuration Parameters**

****

**Data Processing:**

Frames are extracted from each video using the frames\_extraction function, which utilizes OpenCV. The extracted frames are checked to ensure that the number of frames matches the specified sequence length. Features (frames), labels (action classes), and video file paths are stored in separate lists.

**Video Frames Extraction**

****

**Function to Create a Dataset from Video Files**

****

**Dataset Splitting:**

The dataset is split into training and testing sets using a 75-25% ratio. Training features and labels are further split into mini-batches to train the model

**One-Hot Encoding Labels and Splitting Data into Train and Test Sets**

**Model Architecture:**

The model architecture combines a pre-trained Xception convolutional neural network (CNN) with Long Short-Term Memory (LSTM) layers. The Xception model is applied to each frame in the sequence, followed by global average pooling to reduce spatial dimensions. LSTM layers are employed to capture temporal dependencies in the sequence of frames. The final dense layers perform classification with softmax activation



****



**Model Training:**

Training is performed on the training dataset with a batch size of 4, for 30 epochs. An early stopping callback is utilized to prevent overfitting.



Model Evaluation:

The trained model is evaluated on the testing dataset to measure its accuracy. For each sample in the testing dataset, predictions are made and compared against the ground truth labels.

****

Calculate and Print Accuracy on Test Dataset

****

Save model:

****

**Testing on video:**

****

### Flow chart Architecture

A diagram with text and numbers

Description automatically generated with medium confidence

Pre - trained Neural Network models.

**InceptionV3**

**VGG19**

**Xception**

**resnet50**

A diagram of a computer

Description automatically generatedA diagram of a function

Description automatically generated

Figure 14 flow chart Architecture

# CHAPTER 4

## **FINDINGS**

### Model Summary

Xception - LSTM Model

A screenshot of a computer

Description automatically generated

ResNet50 - LSTM Model

A screenshot of a computer program

Description automatically generated

VGG19 - LSTM Model

A screenshot of a computer program

Description automatically generated

InceptionV3+ LSTM Model

A screenshot of a computer program

Description automatically generated

### Plotting Loss & Accuracy

[Xception](https://keras.io/api/applications/xception) – LSTM

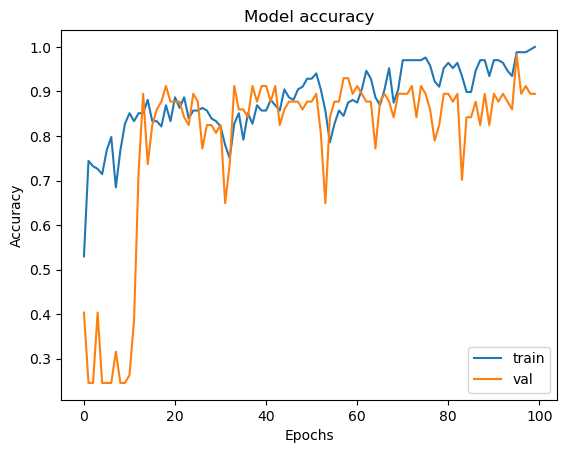
A graph with blue and orange lines

Description automatically generatedA graph of loss of a model

Description automatically generated

Figure 15 plotting loss and accuracy of Xception - lstm

ResNet50– LSTM



### 

A graph with orange and blue lines

Description automatically generated

Figure 16 PLOTTING LOSS AND ACCURACY OF resnet50 - LSTM

VGG19– LSTM

A graph with blue and orange lines

Description automatically generated

A graph with blue and orange lines

Description automatically generated

Figure 17 PLOTTING LOSS AND ACCURACY OF VGG19 - LSTM

INCEPTIONV3– LSTM

A graph with blue and orange lines

Description automatically generated

A graph of loss and loss

Description automatically generated

Figure 18 PLOTTING LOSS AND ACCURACY OF INCEPTIONV3 - LSTM

### Testing on videos

In this project the models have beeen tested on 2 different dataset which consists of walking , handwaving and running.

Testing on same dataset used in training the model

*(*[*https://www.csc.kth.se/cvap/actions/*](https://www.csc.kth.se/cvap/actions/)*)*

A person walking in a snowy area

Description automatically generated  A person walking in the snow

Description automatically generated

Testing on other datasets which is taken from online *(*[*https://www.wisdom.weizmann.ac.il/%7Evision/SpaceTimeActions.html*](https://www.wisdom.weizmann.ac.il/%7Evision/SpaceTimeActions.html) *)*

A person walking on the sidewalk

Description automatically generated A person wearing black gloves and a turtleneck

Description automatically generatedA person running on the sidewalk

Description automatically generated

# CHAPTER 5

## **DISCUSSION**

The below table shows total number of parameters, trainable parameters, non-trainable parameters, and size in MB for four different deep learning models with LSTM layers for action recognition on small-scale dataset which has been used in this project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Total Parameters** | **Trainable Parameters** | **Non-trainable Parameters** | **Size (MB)** |
| Xception-LSTM | 21,408,075 | 546,595 | 20,861,480 | 81.67 |
| ResNet50-LSTM | 24,134,307 | 24,081,187 | 53,120 | 92.07 |
| VGG19-LSTM | 20,177,763 | 153,379 | 20,024,384 | 76.97 |
| InceptionV3-LSTM | 22,349,379 | 546,595 | 21,802,784 | 85.26 |

The InceptionV3-LSTM model, with its small size of 85.26 MB, is suitable for devices with limited processing power or memory. It has the fewest trainable parameters at 546,595, making it suitable for smaller devices. The ResNet50-LSTM model has the largest number of total parameters at 24,134,307, making it suitable for more powerful devices. The VGG19-LSTM model has a moderate size at 76.97 MB and a small number of trainable parameters at 153,379. The Xception-LSTM model has a similar size but more non-trainable parameters, potentially affecting performance on certain devices.

Overall, the choice of model depends on the specific requirements of the application, such as the available processing power and memory, as well as the desired accuracy and performance. In this we have used all the models to test which model gives the highest accuracy results while testing the test dataset.

In this project, Training is performed on the training dataset with a batch size of 4, for 30 epochs. An early stopping callback is utilized to prevent overfitting, for all four models and then tested with test dataset.

The below table shows on the findings of accuracy after the training for the models on the same dataset.

|  |  |  |
| --- | --- | --- |
| **Model** | **Training** | |
| **accuracy** | **validation accuracy** |
| [Xception](https://keras.io/api/applications/xception) - LSTM | 100 | 100 |
| [VGG19](https://keras.io/api/applications/vgg/#vgg19-function) - LSTM | 95.83 | 96.49 |
| [ResNet50](https://keras.io/api/applications/resnet/#resnet50-function) - LSTM | 100 | 89.47 |
| [InceptionV3](https://keras.io/api/applications/inceptionv3) - LSTM | 100 | 98.25 |

All models have achieved high training accuracies, demonstrating that they can fit the training data effectively. However, this does not necessarily guarantee strong performance on unseen data.

Comparing validation accuracies to training accuracies reveals that Xception + LSTM has minimal overfitting (both at 100%), while ResNet50 + LSTM exhibits moderate overfitting (training accuracy of 100%, yet validation accuracy of only 89.47%) .

The Xception LSTM model has 100% training and validation accuracies, but this may indicate overfitting. VGG19 and ResNet50 models show slight discrepancies, with VGG19 showing slightly lower validation accuracy. InceptionV3 has high training accuracy (100% and 98.25%), indicating good generalization capability with minimal overfitting. However, the model's performance on unseen data may be affected.

The Xception LSTM model has concerns about overfitting, while VGG19 and ResNet50 models show better generalization. ResNet50 has a significant drop in validation accuracy, possibly due to model complexity or insufficient regularization techniques. InceptionV3 has high accuracies and minimal overfitting, indicating good generalization capability.

When choosing a model for deployment, consider both training accuracy and performance on unseen data. Models with lower training accuracies but smaller gaps between training and validation are better for generalization.

The below table shows the accuracy of test dataset for the models for the dataset used while training the model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Xception  - LSTM | ResNet50 - LSTM | VGG19 - LSTM | InceptionV3 - LSTM |
| **Walking** | 100.00% | 70.59% | 85.29% | 88.24% |
| **Running** | 97.56% | 95.12% | 95.12% | 95.12% |
| **Handwaving** | 100.00% | 100.00% | 100.00% | 100.00% |
| Accuracy | 99.05% | 88.57% | 93.33% | 94.29% |

The below table shows the accuracy of test dataset for the models for the unseen dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Xception  + LSTM | ResNet50 + LSTM | VGG19 + LSTM | InceptionV3 + LSTM |
| **Walking** | 11.11% | 44.44% | 0.00% | 88.89% |
| **Running** | 100.00% | 0.00% | 22.22% | 0.00% |
| **Handwaving** | 77.78% | 66.67% | 100.00% | 100.00% |
| Accuracy | 64.29 | 35.71% | 39.29% | 60.71% |

The performance discrepancy between the test and unseen datasets of Xception, ResNet50, VGG19, and InceptionV3 LSTM models indicates potential overfitting issues. Xception - LSTM achieves high accuracy on the test dataset, but a significant drop in accuracy on the unseen dataset suggests it struggles to generalize well.

The results suggest that the models may have overfitted the training dataset, and their performance may not generalize well to new data.

# CHAPTER 6

## **CONCLUSION**

This thesis emphasizes the importance of selecting the right CNN architecture when integrating LSTM networks for human action recognition tasks. The study reveals that the architectural intricacies of CNN models significantly impact the performance of the combined architecture. Deeper architectures like Xception and InceptionV3 are effective in capturing spatial and temporal features within action sequences, leading to superior recognition accuracy. The research contributes to the field of deep learning by revealing the interplay between CNN and LSTM components in action recognition systems.

Future work could focus on improving the models' generalization performance by using techniques such as data augmentation, transfer learning, or regularization. Additionally, the models' performance could be evaluated on larger and more diverse datasets to assess their suitability for real-world applications.

# **REFERENCES**

|  |  |
| --- | --- |
| [1] | A. Anwar, "Cheat Sheets for Machine Learning and Data Science," 15 January 2024. [Online]. Available: https://sites.google.com/view/datascience-cheat-sheets#h.w5n247jxgdas. |
| [2] | F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *IEEE Xplore,* 2017. |
| [3] | A. K.. 🇵🇸, "Xception — With Depthwise Separable Convolution," 1 January 2024. [Online]. Available: https://www.linkedin.com/pulse/xception-depthwise-separable-convolution-ayoub-kirouane/. |
| [4] | P. Ferlet, "How to work with Time Distributed data in a neural network," 3 January 2024. [Online]. Available: https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00. |
| [5] | K. Simonyan and A. Zisserman, "Two-Stream Convolutional Networks," pp. 2-3, 2014. |
| [6] | L. A. H. M. R. S. V. S. G. K. S. T. D. Jeff Donahue, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," 2016. |
| [7] | M. F. V. S. A. H. K. Ali Diba, "Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification," 2017. |
| [8] | K. RAAGULBHARATWAJ, "notebook89873cc695," 12 December 2023. [Online]. Available: https://www.kaggle.com/code/raagulbharatwajk/notebook89873cc695. |
| [9] | V. Maskara, "Literature Survey: Human Action Recognition," 18 Febuary 2024. [Online]. Available: https://www.maskaravivek.com/post/survey-of-human-action-recognition/. |