AIM 801: PROJECT ELECTIVE

Dibyarup Pal
MT2023090
International Institute of Information Technology, Bangalore

This report includes the work I have completed for this project elective. The workflow is as follows:

1. Literature Survey
2. Coding

Literature Survey

Papers Reviewed

- Vision based Human Activity Recognition using Deep Neural Network Framework
- Vision based Human Activity Recognition using Hybrid Deep Learning
- Human Activity Recognition Vision Based Pose Detection
- Vision-based human activity recognition using deep learning techniques
- Vision Transformer and Deep Sequence Learning for Human Activity Recognition in Surveillance Videos
- Leveraging CNN and Transfer Learning for Vision-based Human Activity Recognition
- Human Action Recognition Based on Three-Stream Network with Frame Sequence Features
- Vision-based human activity recognition for reducing building energy demand
- Vision-based Human Activity Recognition Using Local Phase Quantisation

Detailed Overview of Paper

Paper 1: Vision based Human Activity Recognition using Deep Neural Network Framework

- In this paper Vision-based Human activity recognition is accomplished using a deep learning neural network technique using Depthwise Separable Convolution (DSC) with Bidirectional Long Short-Term Memory (DSC-BLSTM). The overall DSC-BLSTM workflow is illustrated in the Fig 1.
- Dataset: **Kinetics 400** action recognition dataset of action videos, accumulated from YouTube. With 306,245 short-trimmed videos from 400 action categories. It is one of the largest and most widely used dataset in the research community for benchmarking state-of-the-art video action recognition models
- The Feature extraction is done using the MobileNetV2 convolution network using DSC is a factorized form of the standard convolution.

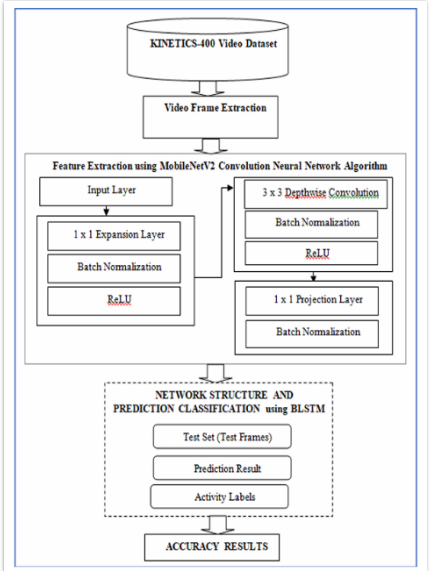


Fig 1: Proposed DSC-BLSTM Algorithm Flow Diagram

Workflow of DSC-BLSTM

- ❑ Video Frame Extraction Video Frame Extraction
- ❑ Feature Extraction using Depthwise Separable Convolution Neural Network Algorithm.

Instead of applying each filter to all the channels of the input like the standard convolution, the Depthwise Convolution (DC) layer applies one filter to one input channel, and then 1x1 Pointwise Convolution (PC) is employed to combine the outputs of the DC.

- ❑ Depthwise Separable Convolution with Bidirectional LSTM (DSC-BLSTM)

Result

The results were impressive and worked well after we used the DSC-BLSTM method in the HAR system. The result is shown as 93.8%

Methods	MC-HF-SVM	Baseline LSTM	Bidir-LSTM	DSC-BLSTM
Accuracy	89.3	90.8	91.1	**93.8**
F1-Score	89.0	90.8	91.1	**92.637**

Table I: Comparison of accuracy and F1-Score of the proposed model

Algorithm: DSC-BLSTM Pseudo code

Input: Continuous Video Frames *f*, Class Labels C
Output: Predicted activity class with accuracy score
Preparation:

1. Video Data Preparation
2. Feature Extraction using Depthwise Separable Convolution Neural Network (DSC)
3. Depthwise Separable Convolution with Bidirectional-LSTM (DSC-BLSTM)

Steps:

While (video frame)

1. Frame *f* ← Extract frames from videos

   //Fix sample duration (i.e., frames taken per loop/iteration) = 16 frames per iteration and sample size (i.e., Frame width size) = 112pixels wide
2. *M* ← Create trained model using DSC method

End While

3. **for** t = 1 to n **do** // where *n* represents number of video frames
   a. frame *f*(t) ← Read the test video frame.
   b. Apply *f*(*t*) to DSC Model // *Calculate Similarity matrix value of test frame f(t) with trained model*
   c. Predict activity Class label with frame *f*(*t*) using (DSC-BLSTM)
4. Label Predicted activity ← Result class label
5. Display predicted activity class with accuracy score

End for

Key Challenges Addressed:

- ❑ Accuracy in Activity Recognition: Traditional methods often lack high accuracy in classifying various human activities, especially from video frames. This paper tackles this by combining Depthwise Separable Convolution (DSC) for feature extraction and Bidirectional LSTM for sequential analysis, achieving better recognition performance.
- ❑ Reduction of Computational Costs: Standard convolutional models are resource-intensive, making real-time applications challenging. The DSC-BLSTM framework reduces computational costs, leveraging MobileNetV2's efficient structure with DSC, which allows for faster inference without compromising accuracy.
- ❑ Robustness Against Environmental Factors: Vision-based methods for HAR are sensitive to lighting, camera angle, and other environmental factors. The DSC-BLSTM method, tested on the extensive Kinetics-400 dataset, demonstrates robustness in recognizing activities under varying conditions, providing more generalized and reliable recognition.
- ❑ Comparison with Existing Models: The paper benchmarks the DSC-BLSTM approach against other HAR algorithms like MC-HF-SVM, Baseline LSTM, and Bidirectional LSTM, showing improved results in terms of accuracy and F1-score, thus establishing the method's effectiveness.

Paper 2: Vision based Human Activity Recognition using Hybrid Deep Learning

Dataset used:

- ❖ **KTH**: 600 videos of 25 fps with 6 different activities
- ❖ **Weizmann**: 90 videos of 30 fps with 10 different activities.

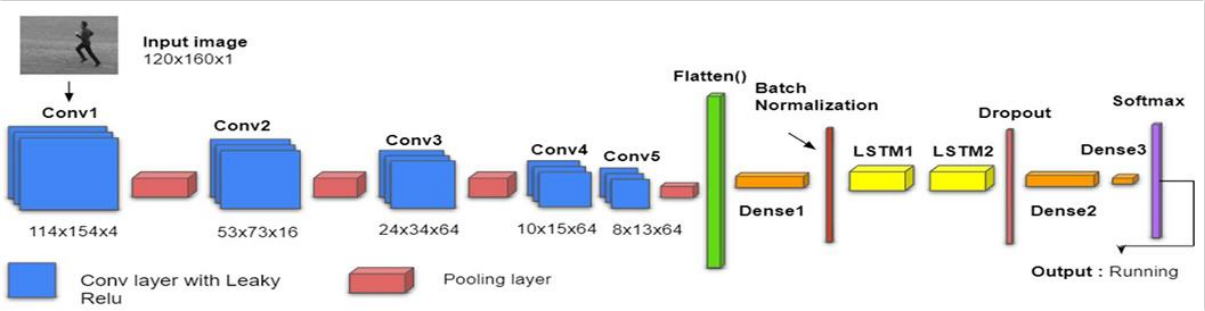


Fig 2: Hybrid CNN-LSTM architecture

This paper proposes a hybrid CNN-LSTM model (as shown in Fig 2) in which features are extracted through CNN and then fed to LSTM as a sequence by means of a time-distributed layer. This model is utilized for classifying six activities from two datasets which have shown accuracy of 96.24% and 93.39% on the KTH and Weizmann datasets, respectively.

Table II: Performance of the Deep learning models on KTH Dataset

Metrics/Models	CNN	LSTM	CNN+LSTM
Accuracy	97.70	93.53	96.24
Precision	97.57	93.65	96.50
Recall	97.74	93.60	95.49
F1 Score	97.67	93.36	96.26

Table III: Performance of the Deep learning models on Wiezmann Dataset

Metrics/Models	CNN	LSTM	CNN+LSTM
Accuracy	87.83	51.05	93.39
Precision	90.20	51.32	92.80
Recall	86.57	41.83	93.64
F1 Score	85.92	46.09	93.22

Key Challenges Addressed:

- Combining Spatial and Temporal Features: The challenge with HAR lies in effectively extracting both spatial (image-based) and temporal (sequence-based) features. CNNs perform well in extracting spatial features but struggle with temporal relationships. LSTMs capture temporal dependencies but lack spatial feature extraction capabilities. The hybrid CNN-LSTM model bridges this gap, leveraging CNN for spatial features and LSTM for sequential understanding.
- Handling Large Datasets Efficiently: Processing extensive video datasets demands high computational resources, which can result in increased training time and potential overfitting. The paper addresses this by using Leaky ReLU activation and batch normalization to stabilize training and prevent overfitting, making the model more computationally efficient.

Paper 3: Human Activity Recognition Vision Based Pose Detection

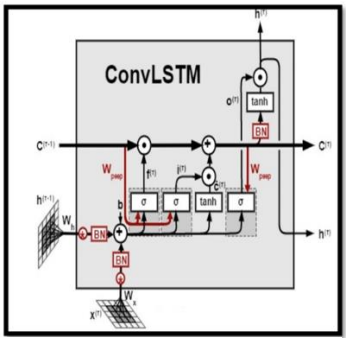
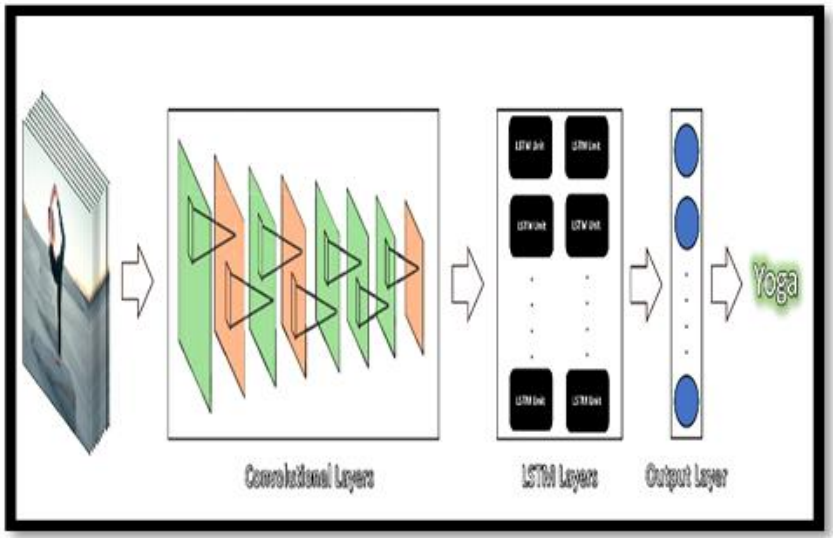


Fig 3: ConvLSTM Unit

A **ConvLSTM** cell is an LSTM network variation that contains convolutional functions inside the circuit (shown in Fig 3). It's an LSTM having convolution embedded into its structure that enables you to detect geographical data properties while taking temporal interactions into account. This method efficiently identifies geometric correlations inside each frame as well as temporal correlations between distinct frames in video categorization. Because of its convolutional design, ConvLSTM can take 3-dimensional inputs (width, height, number of channels).

A **Long-Term Recurrent Convolutional Network (LRCN)**, which integrates CNN as well as LSTM layers together into a unified framework (shown in Fig 4). To represent the time series, a deep neural network is utilised to retrieve spatial characteristics from frames, and the collected characteristics are supplied to the LSTM layer through each time interval. In this manner, the network can train spatiotemporal properties automatically in end-to-end learning and develop a reliable prediction.

Fig 4: LRCN Architecture



Dataset: **UCF101**

UCF101 has the most diversity in terms of activity, with 13320 videos clips from 101 action categories, with substantial changes in camera motions, object presentation and postures, object sizing, views, backdrop distraction, illumination changes, and much more.

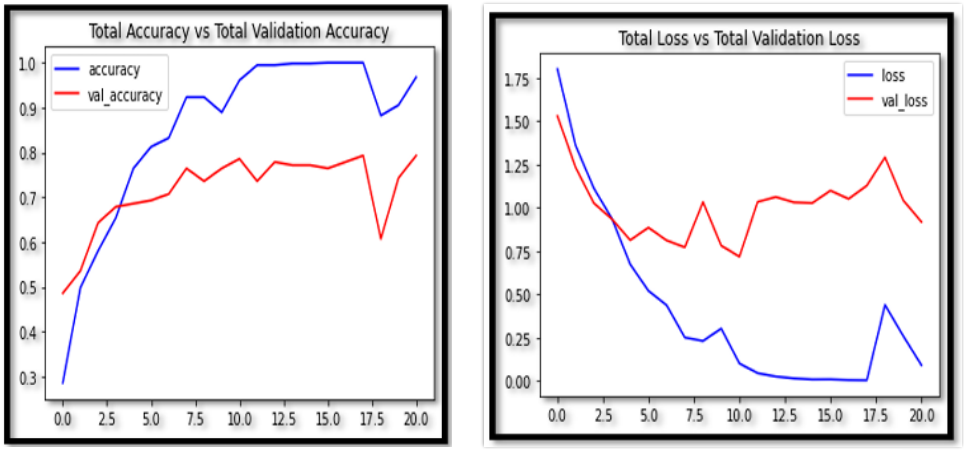


Fig 5: Accuracy and Loss of ConvLSTM Architecture

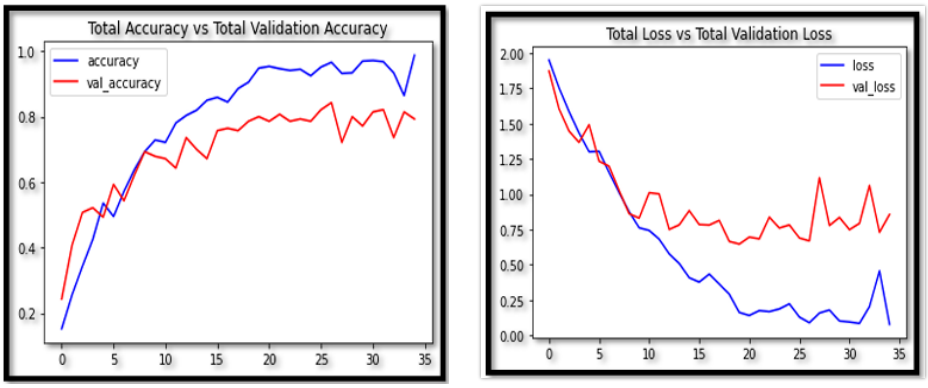


Fig 5: Accuracy and Loss of LRCN

Observations:

In this, we can see ConvLSTM has a total Validation Accuracy of 0.752 and total Accuracy of 0.936 whereas LRCN Neural Network gave a total Validation Accuracy of 0.826 and total Accuracy of 0.965.

All the figures (Fig 4 and Fig 5), clearly indicate the LRCN neural network gave 96.55% accuracy for the testing data set, this is better than the former approach.

Key challenges addressed:

- ❑ Accurate Recognition Across Diverse Activities
- ❑ Efficient Data Processing and Feature Extraction

Paper 4: Vision-based human activity recognition using deep learning techniques

The model presented in this paper uses two methodologies namely, ConvLSTM and LRCN. LSTM layers are employed for temporal detection, whereas CNN aids in the spatial extraction of the frames. ConvLSTM cells are LSTM network variants that include convolutional processes. It is an LSTM with built-in convolution, which enables it to distinguish between spatial input components and take into consideration the temporal relationship. As a result, the convolutional LSTM is capable of taking in 3d input whereas, LSTM solely could take in only 1d input. A SoftMax classifier is used for classification. The ConvLSTM2D layer's output is flattened before being given to the tanh activation function and dense layers. The MaxPooling3D and Dropout layers are employed to minimize frame dimensions and avoid overfitting of the model.

Dataset: **UCF-50** data set, comprising 50 activity classes

Result

Activity No. (Name)	Precision	Recall	F1-score
0 (Swing)	0.5	1	0.67
2 (Horse Race)	1	1	1
3 (Diving)	1	1	1
4 (Basketball)	1	0.50	1

Table IV: Performance key metrics of some activities

The performance of a few of the activities is displayed in this matrix.

Key Metrics	Performance
F1 Score	93.05%
Recall	92.5%
Precision	93.6%
Accuracy	94%

Table V: Overall Performance key metrics of the model

The accuracy of the model is 94% and the f1-score is 93.05%. Thus, it is safe to say that the convLSTM approach works well for vision-based human activity recognition.

Key Challenges Addressed:

- ❑ Reducing Computational Load: Processing video data is computationally intensive, especially for real-time applications. The model uses ConvLSTM, which integrates convolution operations within LSTM cells to manage 3D input data directly, thus reducing computational complexity while preserving temporal and spatial information.

Paper 5: Vision Transformer and Deep Sequence Learning for Human Activity Recognition in Surveillance Videos

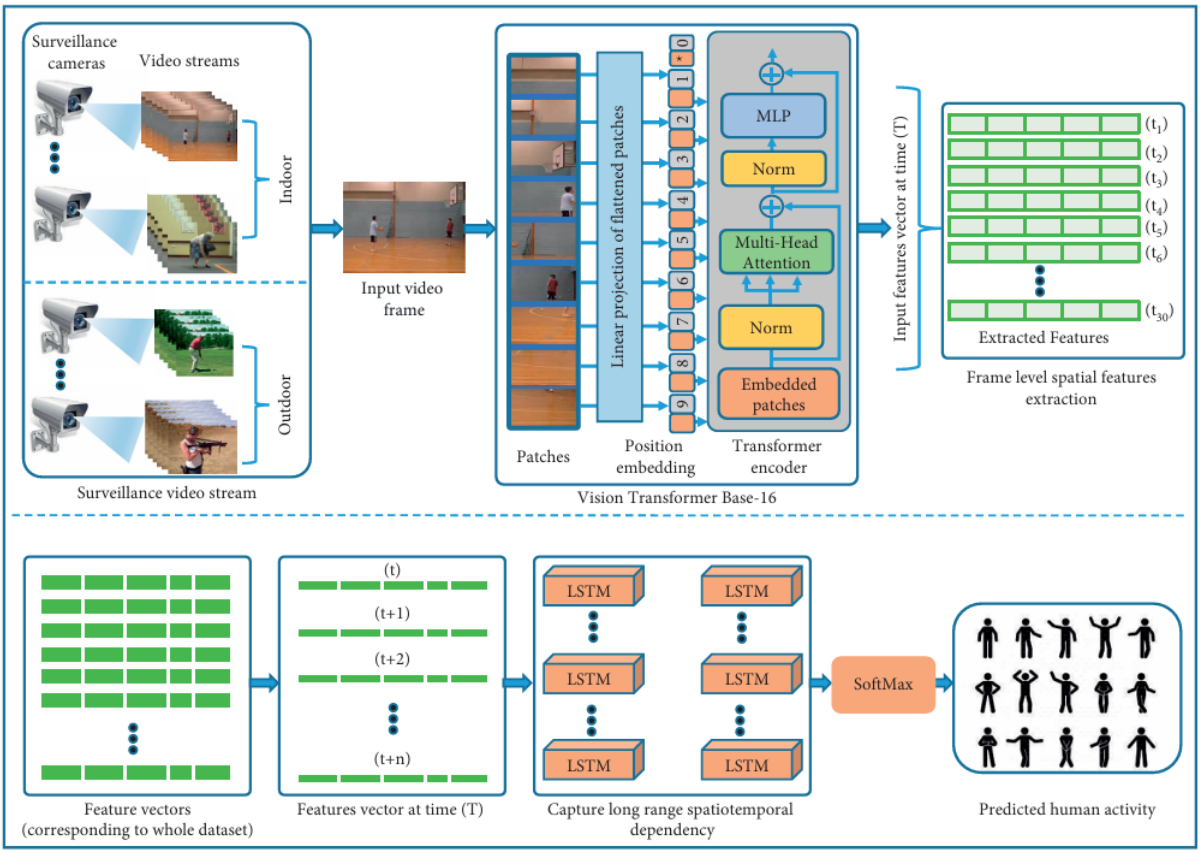


Fig 6: Human Activity recognition workflow using Transformer and Deep Sequence Learning

Dataset used:

- **UCF50:** UCF50 is a very popular HAR dataset consisting of a total of 50 classes.
- **HMDB51:** The HMDB51 dataset contains different varieties of videos related to human body movements such as facial interaction, object interaction with the body, and human interaction for body movements. There are 6766 action video clips collected from different unique sources, all the video clips belong to 51 classes.

The ViT-Base-16 extracts 1000 feature vectors from each frame. Therefore, our proposed sequential learning model takes 30 frames with 1000 spatial feature vectors. Initially, the features vector contains enriched pattern information; therefore, they used 128 LSTM units to learn all possible discriminative features. Then the features space is reduced by 64 numbers of LSTM units to efficiently map the number of classes, that is, 51 and 50 classes of HMDB51 and UCF50, respectively. Furthermore, to avoid overfitting and make the network more stable during training with faster learning, they utilized a 50% dropout and batch normalization.

Layer (type)	Output shape	No. of parameters
Input data	(None, 30, 1000)	0
LSTM	(None, 30, 128)	578048
LSTM	(None, 64)	49408
Dropout	(None, 64)	0
Batch normalization	(None, 64)	256
Activation	(None, 64)	0
Dense	(None, 64)	4160
Dense	(None, 51)	3315
Activation	(None, 51)	0

Fig 7: The Output from ViT-Base-16 is fed into the LSTM followed by Dense layers

The result achieved from the proposed model:

Dataset	Precision (%)	Recall (%)	F1-score (%)
UCF50	96.18655	96.14458	96.08283
HMDB51	76.49243	73.71429	73.51059

- ❑ Limitations of CNNs for Long-Range Temporal Dependencies: This paper uses Vision Transformers, which handle longer sequences via self-attention mechanisms, enabling improved spatial feature extraction across larger temporal spans.
- ❑ Complexities in Capturing Spatial-Temporal Relationships: The model uses ViT for detailed spatial features and LSTM for temporal sequence modeling, which enables effective tracking of actions across frames.
- ❑ Background Clutter and Occlusion in Surveillance Environments: Surveillance videos often include cluttered backgrounds, occlusions, and varying scales, complicating action recognition. The model mitigates this by using ViT, which performs well even in cluttered scenes.

Paper 6: Leveraging CNN and Transfer Learning for Vision-based Human Activity Recognition

This paper focuses on using Convolutional Neural Networks (CNNs) with transfer learning to recognize human activities from video data. Specifically, the authors employ transfer learning to leverage pre-trained models on large datasets, like ImageNet, for activity classification using the Weizmann dataset, which contains video sequences of people performing various actions. The paper evaluates three CNN models—VGG-16, VGG-19, and InceptionNet-v3—and achieves the highest accuracy of 96.95% with VGG-16.

Table VI:

RESULTS ON ACTIVITY RECOGNITION BASED ON DIFFERENT CNN MODELS IN TERMS OF ACCURACY SCORE, PRECISION, RECALL, AND F1-SCORE

Model	**Accuracy (in %)**	**Precision (in %)**	**Recall (in %)**	**F1-score (in %)**
VGG-16	96.95	97.00	97.00	97.00
VGG-19	96.54	97.00	97.00	96.00
Inception-v3	95.63	96.00	96.00	96.00

Table VII:

PERFORMANCE COMPARISON USING WEIZMANN DATASET

Model	**Accuracy (in %)**
VGG-16	96.95
Cai et al. [19]	95.70
Kumar et al. [20]	95.69
Feng et al. [21]	94.10
Han et al. [22]	90.00

Key Challenges Addressed

- Data Limitations for Training Deep Models: Large-scale labelled data is typically required to train CNNs effectively, but HAR datasets are often small. The authors address this by using transfer learning, utilizing features from models pre-trained on the large ImageNet dataset, which enhances performance on the smaller Weizmann dataset.

Paper 7: Human Action Recognition Based on Three-Stream Network with Frame Sequence Features

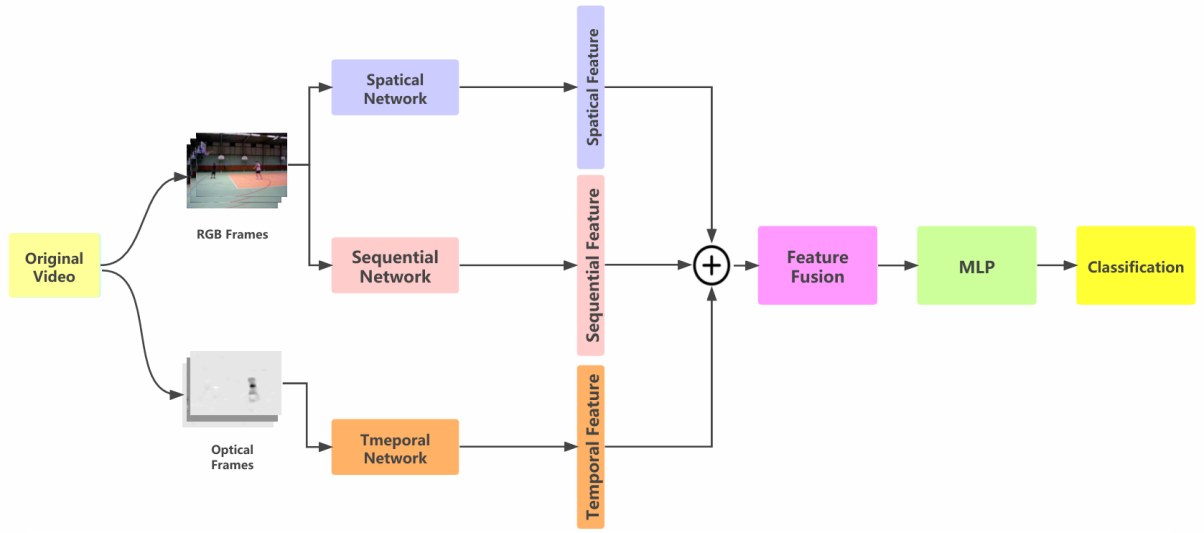


Fig 7: Three-Stream Network Architecture

This paper introduces a novel three-stream neural network for HAR. This model combines spatial, temporal, and frame sequence information to improve recognition accuracy in video-based HAR. Traditional two-stream networks capture spatial (appearance) and temporal (motion) information but often overlook the sequence of frames, which provides critical long-term and short-term motion cues.

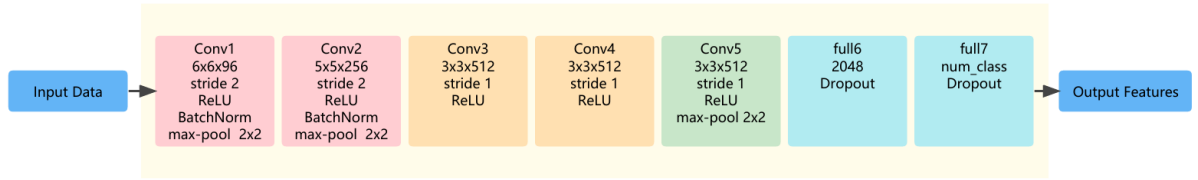


Fig 8: Architecture of Spatial and Temporal Network

Key Challenges Addressed

- **Combining Long-term and Short-term Motion Information:**

  Traditional two-stream models struggle to effectively capture the inter-frame sequence needed to understand both short-term and long-term dynamics in human actions. By adding a third stream dedicated to frame sequence features, this approach extracts both detailed, moment-to-moment spatial-temporal information and the broader action context.

- **Handling Variability in Human Motion and Complex Scenes:**

  Recognizing human actions across complex backgrounds, varying lighting conditions, and occlusions is challenging. This model incorporates spatial, temporal, and sequence information, allowing it to better generalize across diverse video environments. It was tested on challenging datasets, such as UCF11, UCF50, and HMDB51, achieving high accuracy rates of 99.17%, 97.40%, and 96.88%, respectively.

- **Efficient Feature Fusion for Action Classification:**

  Effective feature fusion from multiple streams is necessary to avoid redundancy and improve classification. This paper uses a parallel fusion of spatial, temporal, and sequence features, which proved more effective than other fusion techniques.

Table VIII:

PERFORMANCE OF DIFFERENT STREAMS ON UCF11 DATASET

Stream	Accuracy(%)
Spatial	80.37
Temporal	76.72
Sequential	82.16
Spatial+Temporal	96.71
Spatial+Sequential	97.82
Temporal+Sequential	96.43
All three stream	**99.17**

PERFORMANCE OF DIFFERENT STREAMS ON UCF50 DATASET

Stream	Accuracy(%)
Spatial	73.38
Temporal	69.89
Sequential	76.71
Spatial+Temporal	92.18
Spatial+Sequential	91.58
Temporal+Sequential	87.46
All three stream	**97.40**

PERFORMANCE OF DIFFERENT STREAMS ON HMDB51 DATASET

Stream	Accuracy(%)
Spatial	71.46
Temporal	66.85
Sequential	68.49
Spatial+Temporal	87.21
Spatial+Sequential	83.54
Temporal+Sequential	88.92
All three stream	**96.88**

- Three-Stream Architecture: The architecture separates a video into RGB frames (capturing spatial information) and optical flow frames (capturing motion information). Each type of data is processed through separate networks:
  - RGB frames are passed through a sequential network to capture frame-by-frame sequence information.
  - Optical flow frames go through a temporal network to capture motion-related features.
- Video Preprocessing: Frame Extraction, Data Structuring, Optical Flow Calculation
- Spatial and Temporal Network: The spatial and temporal networks are nearly identical 2D CNN models used to extract spatial and temporal features from the RGB and optical flow frames, respectively.
- Sequential Network: Unlike conventional approaches that rely solely on CNNs, this model incorporates a Long Short-Term Memory (LSTM) network to capture frame sequence information, benefiting from LSTM's strength in processing sequential data.
- Feature Fusion
  - Parallel Feature Fusion: The model employs parallel fusion to combine the spatial, temporal, and sequence features rather than maximum or concatenation fusion, as this method was found to be more effective for model performance.

$$f_{\text{fusion}} = \frac{f_{\text{spatial}} + f_{\text{temporal}} + f_{\text{sequential}}}{3}$$

- The fused vector is then fed into an MLP for the final classification, improving recognition accuracy by leveraging the combined feature information for a more reliable action classification.

Paper 8: Vision-based human activity recognition for reducing building energy demand

- ❑ This paper presents a vision-based deep learning model that uses human activity recognition (HAR) to improve the energy efficiency of buildings by optimizing HVAC (Heating, Ventilation, and Air Conditioning) systems.
- ❑ The model is deployed via a camera for real-time detection and classification of activities like sitting, standing, and walking within indoor spaces, providing occupancy-based heat emission data.
- ❑ This information can dynamically adjust HVAC operations, leading to more efficient heating and cooling management, and reducing unnecessary energy consumption while maintaining thermal comfort.

Key Challenges Addressed

- Improving HVAC Responsiveness to Real-time Occupancy: Conventional HVAC systems often operate on fixed schedules, resulting in heating or cooling unoccupied spaces. The proposed HAR model integrates real-time activity data to align HVAC usage with actual room occupancy and occupant activities, reducing over-conditioning.

- Handling Complex Indoor Activity Detection: Accurate recognition of human activities (e.g., sitting vs. walking) in varied postures and movements is challenging in an office setting.

Model Selection and Transfer Learning:

- The study uses the TensorFlow Object Detection API to develop the model that supports building, training, and deploying object detection models.

- A transfer learning approach is employed, reusing knowledge from a pre-trained model (Faster R-CNN with Inception V2) trained on a large dataset (COCO). Transfer learning reduces the time and data required for training while enhancing detection accuracy.

Model Configuration (Faster R-CNN with Inception V2):

- The Faster R-CNN (Region-based CNN) with Inception V2 architecture was chosen for its ability to accurately detect objects regardless of size. Unlike other models, such as SSD MobileNet or YOLO, Faster R-CNN can better handle varied object sizes.

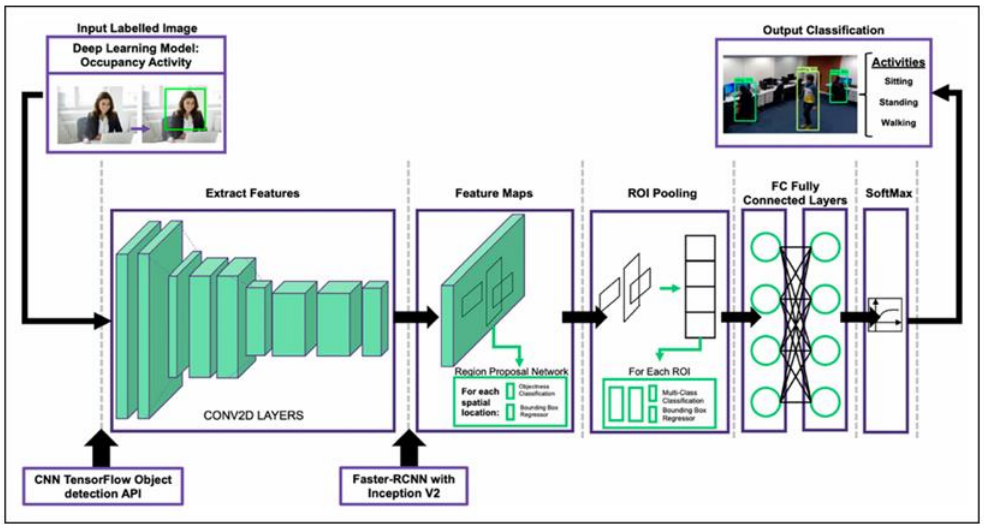


Fig 9: CNN architecture and configuration used for training the models for occupancy activity detection and recognition.

Result

Activity	Accuracy, %	Precision	Recall	F_1 score
Sitting	94.04	0.9250	0.8911	0.9077
Standing	91.43	0.9064	0.8284	0.8657
Walking	92.70	0.8643	0.9266	0.9047

Dataset used: Images representing common office activities (e.g., sitting, standing, walking) were collected and manually labelled using LabelImg software. Each activity was marked with specific labels to create a dataset for training and testing. Multiple labels were used in cases where an image had more than one occupant.

Development and Deployment of the Detection Model:

- The model uses transfer learning with a CNN to recognize activities like sitting, standing, and walking. It was trained and then deployed on a camera for real-time detection, achieving an average detection accuracy of 98.65% across all activities over a 15-minute test period.
- During the real-time tests, data was collected on the number of occupants and their activities, forming a Deep Learning-Influenced Profile (DLIP). This DLIP, reflecting dynamic occupancy patterns, was compared to static profiles, highlighting its usefulness in determining accurate occupancy-based heat emissions for HVAC adjustments.

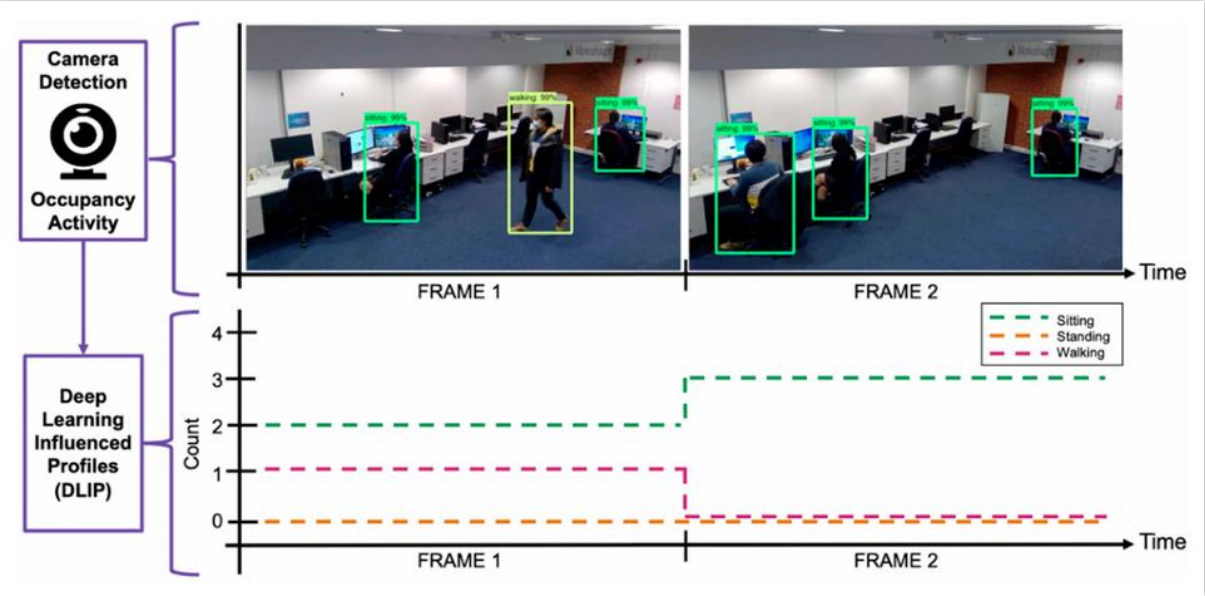


Fig 10: Process of forming the deep learning–influenced profiles from real-time occupancy activity detection and recognition using the deep learning approach.

Paper 9: Vision-based Human Activity Recognition Using Local Phase Quantization

This paper introduces a novel approach for human activity recognition (HAR) that leverages Local Phase Quantization (LPQ) for feature extraction and Support Vector Machine (SVM) for classification. The study addresses common challenges in vision-based HAR, including background clutter, illumination changes, motion blur, and occlusions. This LPQ-based method is tested on two complex datasets, UCF101 and HMDB51, achieving high accuracy rates of 99.78% and 98.67%, respectively, and surpasses other state-of-the-art approaches in computational efficiency and accuracy.

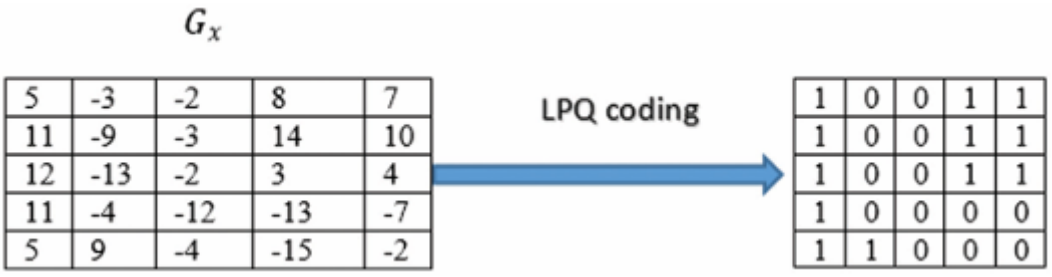


Fig 11: Process of the LPQ encoding

Key Challenges Addressed:

- Variability in Environmental Conditions: Vision-based HAR can be hindered by lighting changes, camera movement, and environmental noise. LPQ, a frequency-based feature extraction method, is resilient to these issues, maintaining accuracy across varied environments by focusing on local phase information rather than pixel intensity.
- Blur and Occlusion Robustness: Blurring caused by camera motion and scene movements is common in video data, making activity recognition challenging. The LPQ descriptor is designed to be invariant to blurring, which ensures robust feature extraction even when video quality degrades due to motion blur or occlusion.
- Efficient and Simplified Feature Extraction: Unlike traditional methods that require combining multiple feature descriptors, this study demonstrates that using LPQ alone can achieve high accuracy on complex datasets. This reduces computational demands and simplifies the feature extraction process, allowing for faster HAR without sacrificing accuracy.
- Reduced Computational Cost Compared to Deep Learning: Many deep learning models require extensive computational resources and large datasets. By using a single LPQ feature descriptor with an SVM classifier, the proposed method achieves comparable or better accuracy with significantly lower computational cost, making it more accessible for applications with limited resources.

Performance metrics	UCF101 dataset	HMDB51 dataset
Accuracy (%)	99.78	98.67
Precision	.9978	.9867
Sensitivity	.9978	.9867
Specificity	.1000	.9997
F-measure	.9978	.9867

Table IX: The performance metrics of HAR on UCF101 and HMDB51 dataset

Methodology:

- **Feature Extraction:**
  - ✓ LPQ is especially effective for HAR, where blurring often occurs due to camera motion or movements within the scene.
  - ✓ LPQ works in the frequency domain (as opposed to spatial domain methods like Local Binary Pattern, LBP), focusing on phase information in the image to distinguish features less affected by blurring.

✓ Using Short-Term Fourier Transform (STFT), LPQ converts the image from spatial to the frequency domain, where it separates real and imaginary parts, focusing on the four lowest frequencies. This separation enables LPQ to generate a feature that remains stable despite variations in image clarity.

❑ **LPQ Encoding and Histogram Formation:**

✓ LPQ encoding works similarly to LBP in that it creates a histogram representing the local texture of an image. However, LPQ operates in the frequency domain, making it more robust to blurring. This encoding process generates a local texture feature that typically outperforms spatial-based methods like LBP and Local Ternary Pattern (LTP).

❑ **Classification with SVM:**

✓ Support Vector Machine (SVM) is used for classification. SVM is a supervised machine learning approach that aims to find an optimal hyperplane to separate data points of different classes.

✓ In this study, a one-to-one SVM classifier is used, meaning it classifies activities by maximizing the distance (or margin) between the hyperplane and the nearest data points from each class.

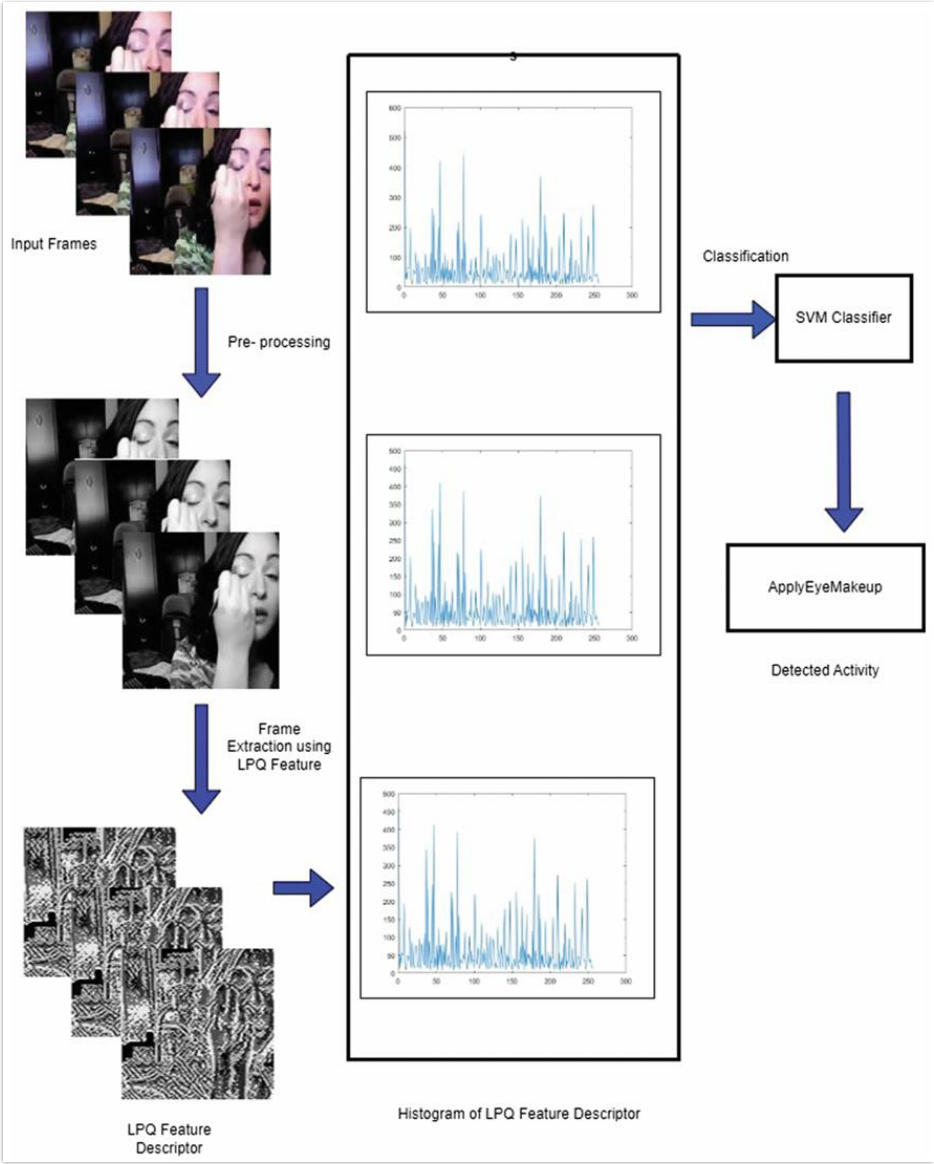


Fig 12: An overview of the proposed

Coding

After conducting a literature survey of nine papers, I implemented the approach outlined in "Human Action Recognition Based on Three-Stream Network with Frame Sequence Features" paper.

Created 4 notebooks given as follows:

1. Notebook 1: Implemented only the Spatial Network on UCF11
2. Notebook 2: Implemented the Three-Stream Network using 3D CNN on UCF11
3. Notebook 3: Implemented the Three-Stream Network using 2D CNN on UCF11
4. Notebook 4: Implemented the Three-Stream Network using 2D CNN and performed 10-fold cross-validation on UCF11

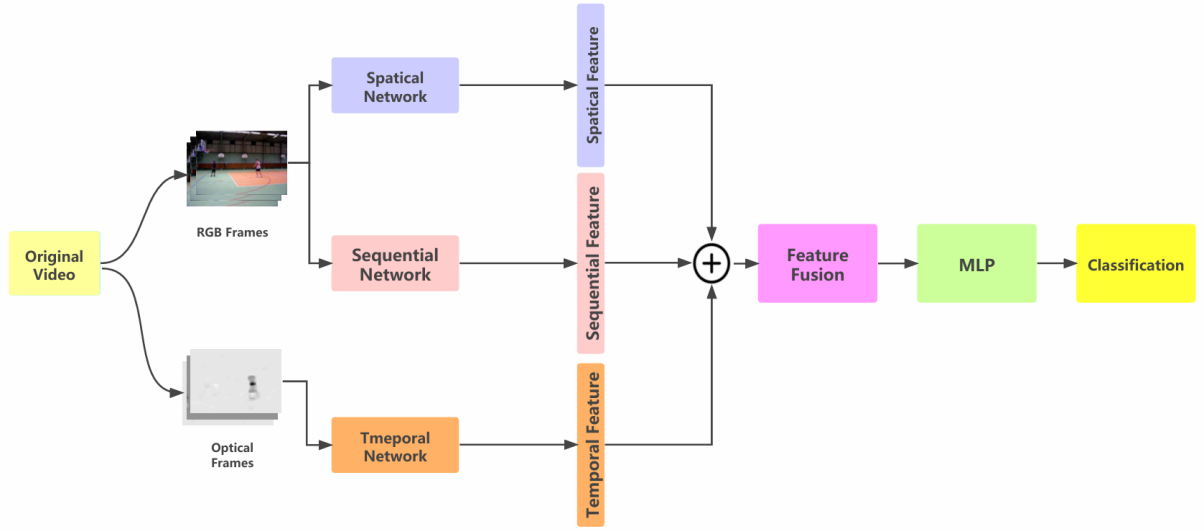


Fig 13: Implemented this Three-Stream Network for Video Classification

Note: As the paper did not provide explicit information on certain aspects, several assumptions were made during implementation. These assumptions are clearly outlined and explained in the respective Notebook.

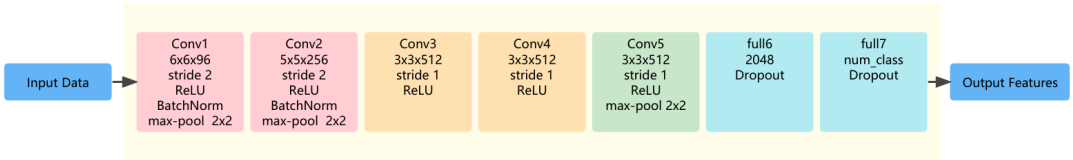


Fig 14: Implemented this CNN architecture for Spatial and Temporal Network

Notebook 1: Human Activity Recognition using only Spatial Network

Data preprocessing:

- The video data is processed for classification tasks. The create_dataset function is then defined to organize the video data. This function traverses through the dataset directory, iterating over each class folder and its subfolders to extract the file paths of video files along with their corresponding class names. Next, the collected data is converted into a pandas DataFrame, with columns videos (containing the video file paths) and Category (representing the class names).
- To prepare the data for machine learning models, label encoding is done on the class names and also one-hot encoded vector of size equal to the number of classes is generated. At the end of the process, the

DataFrame contains four key components: the video file paths, class names, numeric labels, and one-hot encoded labels, making it a structured and ready-to-use dataset for deep learning model training.

- Further, the dataset is split into training, validation, and test sets, ensuring that 72% of the data is used for training, 8% for validation, and 20% for testing, thereby providing distinct subsets for model development and evaluation.

Video Preprocessing:

Two major preprocessing steps are extracting frames, and calculating optical flow.

- It begins with the extraction of frames, which extracts a sequence of 10 evenly spaced frames from a video file. Using OpenCV, the function calculates the number of frames to skip to ensure that 10 frames are evenly distributed across the video. It captures these frames and saves them as individual image files in a specified folder.

  Assumption: Considered 10 evenly spaced frames from the video for further processing.

- Finally, the frames and labels are converted into PyTorch tensors. The RGB frames are stacked along the time axis to form 3D tensors, while the labels are represented as one-hot encoded tensors.

- This implementation enables efficient batch loading and preprocessing using PyTorch's DataLoader, making it suitable for training models that leverage both RGB appearance features and motion information captured through optical flow. The combination of these features ensures the model can effectively analyze spatiotemporal data.

- Visualization of extracted frames from a video

Fig 15: consecutive 5 evenly spaced RGB frames

- Preprocessing:
  - **RGB Frames:** Resized to 128×128, normalized using ImageNet statistics.

Architecture:

Implements the spatial network (as shown in Fig 4) for video classification,

- The **CNNStream3D** Network is a 3D CNN, this architecture is suited for tasks like action recognition or video-based activity classification. It processes input tensors through convolutional, pooling, and fully connected layers to extract spatiotemporal features and classify them into target classes.

  Assumption: Implemented the CNN network using 3D CNN.

Training:

This code trains a video classification model using RGB frames, evaluates its performance, and saves the best-performing model dynamically. The key steps are:

1. **Training Function:** The train function is responsible for the model's training and validation phases. It computes the loss using the defined function, optimizes model parameters using the specified optimizer, and evaluates performance metrics on both training and validation sets. Furthermore, it measures training time per epoch, saves the model with the highest validation accuracy, and handles dynamic saving of the best model for deployment or further evaluation.

   Assumption: 20 epochs

2. **Safe Collate:** Filters out invalid batches with insufficient frames during data loading.
3. **Datasets and DataLoaders:** Prepares data for training, validation, and testing with specified transformations and safe collation. Training batches are shuffled for better generalization.
4. **Model Setup:**
    - Moved the model to GPU
    - The optimization process employs the ***Adam optimizer*** with a learning rate of ***0.00125***, and the ***Categorical Cross-Entropy loss*** function is used to measure model performance."

Results:

1. **Training and Validation Accuracy:**
    - The accuracy graph indicates relatively slow progress, with validation accuracy outperforming training accuracy. This suggests possible underfitting, where the model is unable to effectively capture the underlying patterns in the training data.
    - After 20 epochs, the training accuracy remains low (~25%), while validation accuracy improves to approximately **37%**.

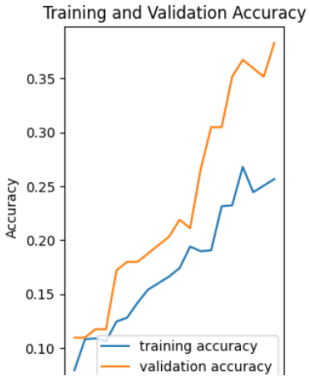


Fig 16: Training and validation Accuracy for all epochs

2. **Training Time:** (Approx.) 1 hour and 41 minutes
3. **Testing Accuracy:** The model achieved a **Test Accuracy of 37.62%.**
4. **Confusion Matrix Analysis:**

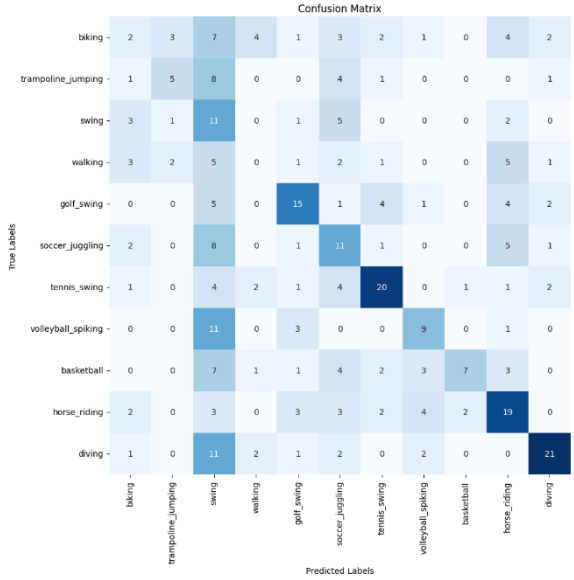


Fig 17: Confusion matrix showing the performance of the test set

- The confusion matrix reveals significant misclassifications across most activity classes.
- For example:
    - **Swing** and **Volleyball Spiking** are heavily misclassified across multiple classes.
    - Activities like **Walking** and **Biking** show no strong class dominance.
    - **Diving** and **Tennis Swing** show relatively better performance with more correct predictions compared to other classes.
- Misclassifications occur frequently between similar activities, e.g., **soccer juggling** and **volleyball spiking** or **horse riding** and **biking**.

5. **Key Takeaways:**

    - The model's performance indicated a need for more training epochs, which was addressed in subsequent Notebook.

Notebook 2: Human Activity Recognition based on Three-Stream Network using 3D-CNN

Data preprocessing:

- The video data is processed for classification tasks. The create_dataset function is then defined to organize the video data. This function traverses through the dataset directory, iterating over each class folder and its subfolders to extract the file paths of video files along with their corresponding class names. Next, the collected data is converted into a pandas DataFrame, with columns videos (containing the video file paths) and Category (representing the class names).
- To prepare the data for machine learning models, label encoding is done on the class names and also one-hot encoded vector of size equal to the number of classes is generated. At the end of the process, the DataFrame contains four key components: the video file paths, class names, numeric labels, and one-hot encoded labels, making it a structured and ready-to-use dataset for deep learning model training.
- Further, the dataset is split into training, validation, and test sets, ensuring that 72% of the data is used for training, 8% for validation, and 20% for testing, thereby providing distinct subsets for model development and evaluation.

Video Preprocessing:

Two major preprocessing steps are extracting frames, and calculating optical flow.

- It begins with the extraction of frames, which extracts a sequence of 10 evenly spaced frames from a video file. Using OpenCV, the function calculates the number of frames to skip to ensure that 10 frames are evenly distributed across the video. It captures these frames and saves them as individual image files in a specified folder.

  Assumption: Considered 10 evenly spaced frames from the video for further processing.

- The next part calculates optical flow, which computes the optical flow between consecutive frames. Optical flow captures motion information by analysing changes between grayscale versions of the frames. Using **Farneback's** algorithm (cv2.calcOpticalFlowFarneback), the function generates a list of flow matrices that describe the motion between frame pairs.
- Finally, the frames and labels are converted into PyTorch tensors. The RGB frames and optical flow data are stacked along the time axis to form 3D tensors, while the labels are represented as one-hot encoded tensors.
- This implementation enables efficient batch loading and preprocessing using PyTorch's DataLoader, making it suitable for training models that leverage both RGB appearance features and motion information captured through optical flow. The combination of these features ensures the model can effectively analyze spatiotemporal data.
- Visualization of extracted frames from a video

Fig 18: Consecutive 5 evenly space RGB frames

- Visualization of corresponding computed Optical Flow frames

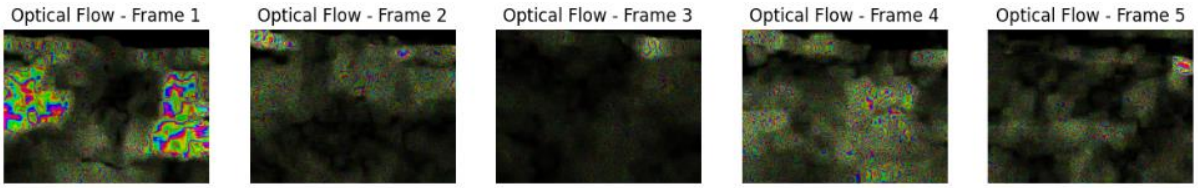


Fig 19: Consecutive 5 corresponding computed optical flow frames

- Preprocessing:
  - **RGB Frames:** Resized to 128×128, normalized using ImageNet statistics.
  - **Optical Flow:** Resized and normalized using channel-specific statistics.

Architecture:

Implements the three-stream network for video classification (as shown in Fig 13), designed to capture complementary features from spatial, temporal, and sequential dimensions of video data. It combines Convolutional Neural Networks (CNNs) for spatial and motion feature extraction with a Long Short-Term Memory (LSTM) network for sequential feature extraction.

The modular design of the Three-stream network is implemented and explained in a step-by-step manner.

- The **CNNStream** class defines the convolutional neural network (as shown in Fig 14) used for both spatial and temporal streams. It consists of five convolutional blocks, each combining convolutional layers, batch normalization, ReLU activation, and max-pooling to extract hierarchical features. The spatial stream processes RGB frames to extract appearance features, while the temporal stream processes two-channel optical flow data to capture motion information.

  Assumption: Implemented the CNN network using 3D CNN.

- The **LSTMStream** class models sequential dependencies in the data using an LSTM network. It processes video sequences by flattening the spatial dimensions of the input frames and feeding them into an LSTM layer. The output from the final time step is passed through a fully connected layer to align the feature dimension with that of the CNN.

  Assumption: Given the absence of a detailed LSTM network architecture in the paper, a single-layer LSTM was implemented for this task.

- The **ThreeStreamNetwork** class integrates the *spatial*, *temporal*, and *sequential* streams into a unified architecture. The outputs from the three streams are fused by averaging their feature representations across the time dimension, creating a single representation that combines appearance, motion, and sequence information. A fully connected layer refines these fused features, followed by a final classification layer that predicts the class probabilities.

Training:

This code trains a video classification model using RGB and optical flow data, evaluates its performance, and saves the best-performing model dynamically. The key steps are:

5. **Training Function:** The train function is responsible for the model's training and validation phases. It computes the loss using the defined function, optimizes model parameters using the specified optimizer, and evaluates performance metrics on both training and validation sets. Furthermore, it measures training time per epoch, saves the model with the highest validation accuracy, and handles dynamic saving of the best model for deployment or further evaluation.

Assumption: 100 epochs

6. **Safe Collate:** Filters out invalid batches with insufficient frames during data loading.
7. **Datasets and DataLoaders:** Prepares data for training, validation, and testing with specified transformations and safe collation. Training batches are shuffled for better generalization.
8. **Model Setup:**
    - Moved the model to GPU
    - The optimization process employs the ***Adam optimizer*** with a learning rate of ***0.00125***, and the ***Categorical Cross-Entropy loss*** function is used to measure model performance."

Results

1. **Training and Validation Accuracy:**
    - The training and validation accuracy plot shows that the model achieved steady improvement during the training process, reaching around **98%** training accuracy after 100 epochs.
    - However, the Maximum validation accuracy achieved is **77.34%**, indicating some degree of overfitting.
2. **Training Time:** (Approx.) 9 hours
3. **Testing Accuracy:** The model achieved a **Test Accuracy of 64.58%.**
4. **Confusion Matrix Analysis:**

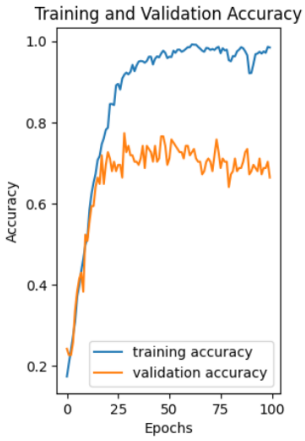


Fig 20: Training and validation Accuracy for all epochs

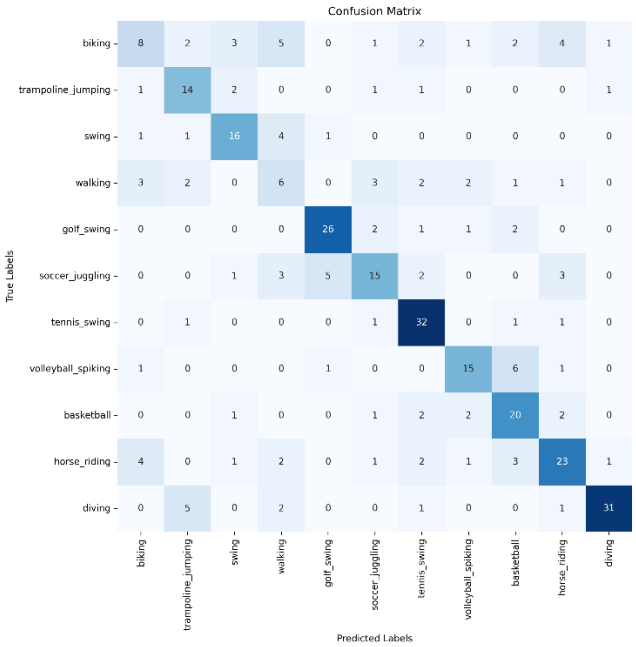


Fig 21: Confusion matrix showing the performance of the test set

- The confusion matrix reveals the following key observations:
    1. Class Performance:
        - "Diving" and "Tennis Swing" are well-classified, achieving high precision and recall with minimal misclassification.
        - "Golf Swing" also performed well, with 31 correct predictions.

- "Trampoline Jumping" and "Horse Riding" faced confusion with other classes, particularly "Biking" and "Walking."

2. Misclassification Issues:
    - "Biking" was frequently misclassified as "Swing" and "Horse Riding," showing confusion among motion-related activities.
    - Misclassifications are relatively frequent in visually or motionally similar activities, such as "Swing" versus "Volleyball Spiking" and "Soccer Juggling."

5. **Key Takeaways:**
    - While implementing the architecture outlined in the paper, certain parameters had to be assumed due to their absence in the original work. Unfortunately, the achieved performance did not match the reported results.
    - To address these discrepancies, adjustments were made to both the assumptions and the architecture, incorporating a 2D-CNN approach as used in subsequent notebook.

Notebook 3: Human Activity Recognition based on Three-Stream Network using 2D-CNN

Data preprocessing:

- The video data is processed for classification tasks. The create_dataset function is then defined to organize the video data. This function traverses through the dataset directory, iterating over each class folder and its subfolders to extract the file paths of video files along with their corresponding class names. Next, the collected data is converted into a pandas DataFrame, with columns videos (containing the video file paths) and Category (representing the class names).
- To prepare the data for machine learning models, label encoding is done on the class names and also one-hot encoded vector of size equal to the number of classes is generated. At the end of the process, the DataFrame contains four key components: the video file paths, class names, numeric labels, and one-hot encoded labels, making it a structured and ready-to-use dataset for deep learning model training.
- Further, the dataset is split into training, validation, and test sets, ensuring that 72% of the data is used for training, 8% for validation, and 20% for testing, thereby providing distinct subsets for model development and evaluation.

Video Preprocessing:

Two major preprocessing steps are extracting frames, and calculating optical flow.

- It begins with the extraction of frames, which extracts a sequence of 10 evenly spaced frames from a video file. Using OpenCV, the function calculates the number of frames to skip to ensure that 10 frames are evenly distributed across the video. It captures these frames and saves them as individual image files in a specified folder.

  Assumption: Considered 10 evenly spaced frames from the video for further processing.
- The next part calculates optical flow, which computes the optical flow between consecutive frames. Optical flow captures motion information by analysing changes between grayscale versions of the frames. Using **Farneback's** algorithm (cv2.calcOpticalFlowFarneback), the function generates a list of flow matrices that describe the motion between frame pairs.
- Finally, the frames and labels are converted into PyTorch tensors. The RGB frames and optical flow data are stacked along the time axis to form 3D tensors, while the labels are represented as one-hot encoded tensors.
- This implementation enables efficient batch loading and preprocessing using PyTorch's DataLoader, making it suitable for training models that leverage both RGB appearance features and motion

information captured through optical flow. The combination of these features ensures the model can effectively analyze spatiotemporal data.

- Preprocessing:
  - **RGB Frames:** Resized to 128×128, normalized using ImageNet statistics.
  - **Optical Flow:** Resized and normalized using channel-specific statistics.

Architecture:

Implements the three-stream network for video classification (as shown in Fig 13), designed to capture complementary features from spatial, temporal, and sequential dimensions of video data. It combines Convolutional Neural Networks (CNNs) for spatial and motion feature extraction with a Long Short-Term Memory (LSTM) network for sequential feature extraction.

The modular design of the Three-stream network is implemented and explained in a step-by-step manner.

- The **CNNStream** class defines the convolutional neural network (as shown in Fig 14) used for both spatial and temporal streams. It consists of five convolutional blocks, each combining convolutional layers, batch normalization, ReLU activation, and max-pooling to extract hierarchical features. The spatial stream processes RGB frames to extract appearance features, while the temporal stream processes two-channel optical flow data to capture motion information.

  Assumption: Implemented the CNN network using 2D CNN.

- The **LSTMStream** class models sequential dependencies in the data using an LSTM network. It processes video sequences by flattening the spatial dimensions of the input frames and feeding them into an LSTM layer. The output from the final time step is passed through a fully connected layer to align the feature dimension with that of the CNN.

  Assumption: Given the absence of a detailed LSTM network architecture in the paper, a single-layer LSTM was implemented for this task.

- The **ThreeStreamNetwork** class integrates the *spatial*, *temporal*, and *sequential* streams into a unified architecture. The outputs from the three streams are fused by averaging their feature representations across the time dimension, creating a single representation that combines appearance, motion, and sequence information. A fully connected layer refines these fused features, followed by a final classification layer that predicts the class probabilities.

Training:

This code trains a video classification model using RGB and optical flow data, evaluates its performance, and saves the best-performing model dynamically. The key steps are:

9. **Training Function:** The train function is responsible for the model's training and validation phases. It computes the loss using the defined function, optimizes model parameters using the specified optimizer, and evaluates performance metrics on both training and validation sets. Furthermore, it measures training time per epoch, saves the model with the highest validation accuracy, and handles dynamic saving of the best model for deployment or further evaluation.

    Assumption: 50 epochs

10. **Safe Collate:** Filters out invalid batches with insufficient frames during data loading.
11. **Datasets and DataLoaders:** Prepares data for training, validation, and testing with specified transformations and safe collation. Training batches are shuffled for better generalization.

12. **Model Setup:**
    - Moved the model to GPU
    - The optimization process employs the ***Adam optimizer*** with a learning rate of ***0.00125***, and the ***Categorical Cross-Entropy loss*** function is used to measure model performance."

Results:

1. **Training and Validation Accuracy**
    - The accuracy curve shows that the model's training accuracy steadily increased and plateaued at nearly **100%**, indicating effective learning on the training set.
    - The validation accuracy reached a maximum of **81.25%**, but some fluctuations suggest slight overfitting. Despite this, the validation curve aligns closely with the training curve, demonstrating stable generalization.

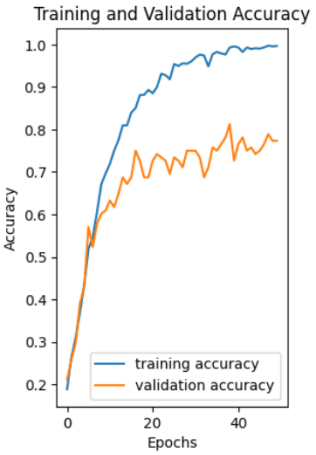


Fig 22: Training and validation Accuracy for all epochs

2. **Training Time:** (Approx.) 4 hours and 50 minutes
3. **Testing Accuracy:** The model achieved a **Test Accuracy of 79.62%**, demonstrating its ability to generalize well on unseen data.
4. **Confusion Matrix Analysis**

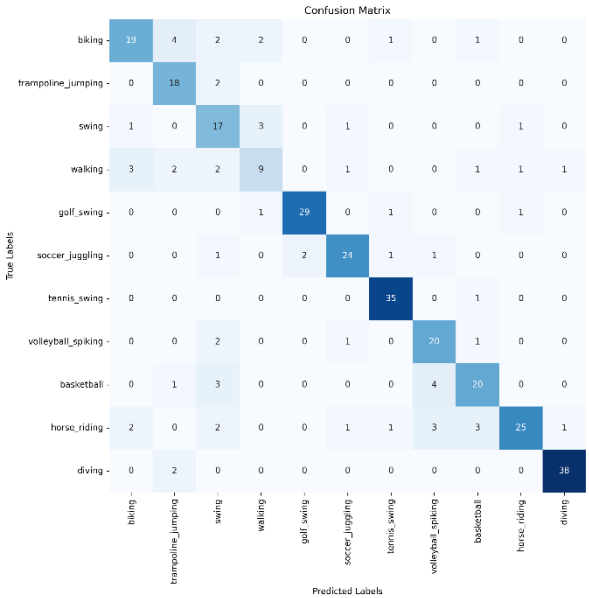


Fig 23: Confusion matrix showing the performance of the Test set

    - The confusion matrix highlights the performance across 11 activity classes.
    - Classes like **tennis_swing (35/36)** and **diving (38/40)** were classified with near-perfect accuracy, indicating strong performance in these activities.
    - Misclassifications were observed in **biking** and **horse_riding**.

5. **Key Takeaways:**
    - Unfortunately, the current modifications, including the revised assumptions and architecture, have not yielded the performance levels reported in the original paper.
    - Additionally, the paper mentions the use of cross-validation, which has been implemented in the subsequent notebook.

Notebook 4: Human Activity Recognition based on Three-Stream Network using 2D-CNN and performed 10-fold cross-validation

Data preprocessing:

- The video data is processed for classification tasks. The create_dataset function is then defined to organize the video data. This function traverses through the dataset directory, iterating over each class folder and its subfolders to extract the file paths of video files along with their corresponding class names. Next, the collected data is converted into a pandas DataFrame, with columns videos (containing the video file paths) and Category (representing the class names).
- To prepare the data for machine learning models, label encoding is done on the class names and also one-hot encoded vector of size equal to the number of classes is generated. At the end of the process, the DataFrame contains four key components: the video file paths, class names, numeric labels, and one-hot encoded labels, making it a structured and ready-to-use dataset for deep learning model training.
- Further, the dataset is split into training, validation, and test sets, ensuring that 72% of the data is used for training, 8% for validation, and 20% for testing, thereby providing distinct subsets for model development and evaluation.

Video Preprocessing:

Two major preprocessing steps are extracting frames, and calculating optical flow.

- It begins with the extraction of frames, which extracts a sequence of 10 evenly spaced frames from a video file. Using OpenCV, the function calculates the number of frames to skip to ensure that 10 frames are evenly distributed across the video. It captures these frames and saves them as individual image files in a specified folder.

  Assumption: Considered 10 evenly spaced frames from the video for further processing.
- The next part calculates optical flow, which computes the optical flow between consecutive frames. Optical flow captures motion information by analysing changes between grayscale versions of the frames. Using **Farneback's** algorithm (cv2.calcOpticalFlowFarneback), the function generates a list of flow matrices that describe the motion between frame pairs.
- Finally, the frames and labels are converted into PyTorch tensors. The RGB frames and optical flow data are stacked along the time axis to form 3D tensors, while the labels are represented as one-hot encoded tensors.
- This implementation enables efficient batch loading and preprocessing using PyTorch's DataLoader, making it suitable for training models that leverage both RGB appearance features and motion information captured through optical flow. The combination of these features ensures the model can effectively analyze spatiotemporal data.
- Preprocessing:
  - **RGB Frames:** Resized to 128×128, normalized using ImageNet statistics.
  - **Optical Flow:** Resized and normalized using channel-specific statistics.

Architecture:

Implements the three-stream network for video classification (as shown in Fig 13), designed to capture complementary features from spatial, temporal, and sequential dimensions of video data. It combines Convolutional Neural Networks (CNNs) for spatial and motion feature extraction with a Long Short-Term Memory (LSTM) network for sequential feature extraction.

The modular design of the Three-stream network is implemented and explained in a step-by-step manner.

- The **CNNStream** class defines the convolutional neural network (as shown in Fig 14) used for both spatial and temporal streams. It consists of five convolutional blocks, each combining convolutional layers, batch normalization, ReLU activation, and max-pooling to extract hierarchical features. The spatial stream processes RGB frames to extract appearance features, while the temporal stream processes two-channel optical flow data to capture motion information.

  Assumption: Implemented the CNN network using 2D CNN.

- The **LSTMStream** class models sequential dependencies in the data using an LSTM network. It processes video sequences by flattening the spatial dimensions of the input frames and feeding them into an LSTM layer. The output from the final time step is passed through a fully connected layer to align the feature dimension with that of the CNN.

  Assumption: Given the absence of a detailed LSTM network architecture in the paper, a single-layer LSTM was implemented for this task.

The **ThreeStreamNetwork** class integrates the *spatial*, *temporal*, and *sequential* streams into a unified architecture. The outputs from the three streams are fused by averaging their feature representations across the time dimension, creating a single representation that combines appearance, motion, and sequence information. A fully connected layer refines these fused features, followed by a final classification layer that predicts the class probabilities.

Training:

The code implements a robust k-fold cross-validation pipeline for training and evaluating a three-stream neural network on a video-based dataset. The workflow is modular, breaking the process into separate steps for dataset preparation, model training, evaluation, and cross-validation.

1. The ***get_kfold_data*** function leverages Stratified K-Fold Cross-Validation from sklearn to split the dataset into k folds while maintaining a balanced distribution of labels across each fold. For each fold, the function creates training and testing splits and appends them to a list for use during cross-validation. This ensures that the model is tested on all portions of the data, reducing bias and variance in the evaluation process.
    - In accordance with the paper's recommendations, **10-fold cross-validation** is utilized to assess the model's generalization performance.
2. The ***train_model*** function handles the training process for the neural network. It trains the model for a specified number of epochs using mini-batch gradient descent. For each batch, the inputs—RGB frames and optical flow data—and their corresponding labels are moved to the appropriate computation device (CPU or GPU). The model performs a forward pass, and the loss is calculated using the specified loss function. The loss is then backpropagated to compute gradients, which are used by the Adam optimizer to update model parameters.

   Assumption: 10 epochs in each fold (due to constraints in GPU hours)
3. The ***evaluate_model*** function performs model evaluation on the test set. This function outputs the average loss and accuracy over the test set, which are crucial metrics for assessing model performance.
4. **Model Setup:**
    - Moved the model to GPU
    - The optimization process employs the Adam optimizer with a learning rate of 0.000125, and the Categorical Cross-Entropy loss function is used to measure model performance."

This pipeline ensures a thorough evaluation of the model's performance across multiple subsets of the data, which is particularly important for video classification tasks. By testing on multiple folds, the process reduces overfitting and provides a more reliable estimate of the model's accuracy and loss.

Results:

- To evaluate the performance of the model, conducted **10-fold cross-validation** and plotted the testing accuracies across each fold, as shown in Figure 24. The individual accuracies for each fold vary, highlighting both the model's performance fluctuations and its robustness under different splits of the data.
- The testing accuracy across the 10 folds ranged from **0.590** to **0.794**, demonstrating variability in performance. Notably, the model achieved its **highest** accuracy of **0.794** in **fold 8**, while the **lowest** accuracy of **0.590** occurred in **fold 9**.
- Despite the fluctuations, the **average testing accuracy** across all folds was calculated to be **0.7076** (indicated by the blue dashed line), which provides a reliable measure of the model's generalization capability.
- **Training Time:** (Approx.) 9 hours and 25 minutes

The observed variability suggests that while the model performs well on average, certain splits of the data may present more challenging scenarios, potentially due to data imbalance or noise.

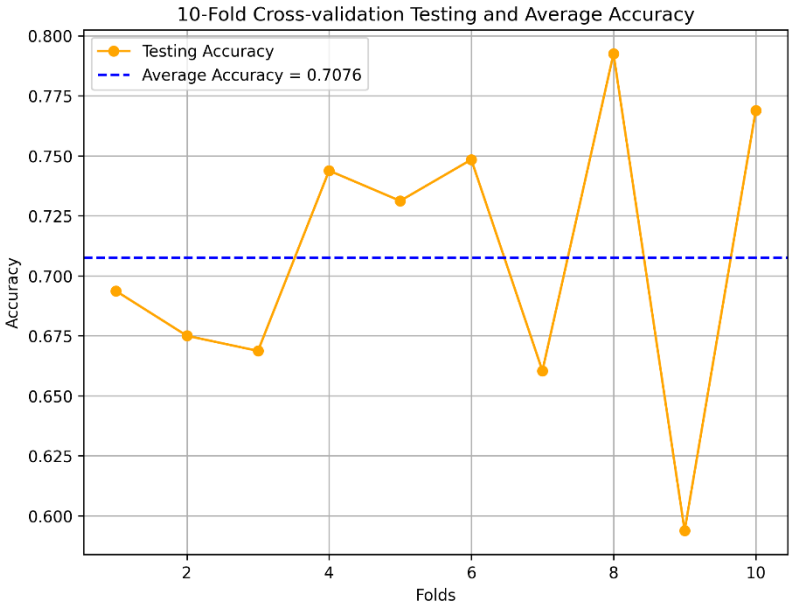


Fig 24: Line plot showing the performance of the model across 10 folds and average accuracy

Future Work

In future work, I aim to enhance the architecture by incorporating Vision Transformers to capture long-range temporal dependencies. Additionally, I will strive to further improve the model's accuracy.