IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 June 2023, accepted 6 July 2023, date of publication 10 July 2023, date of current version 19 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3293813

RESEARCH ARTICLE

ViT-ReT: Vision and Recurrent Transformer Neural Networks for Human Activity Recognition in Videos

JAMES WENSEL, HAYAT ULLAH, AND ARSLAN MUNIR, (Senior Member, IEEE)
Department of Computer Science, Kansas State University, Manhattan, KS 66506, USA

Corresponding author: Arslan Munir (amunir@ksu.edu)

This work was supported in part by the Air Force Office of Scientific Research (AFOSR) under Contract FA9550-22-1-0040.

ABSTRACT Human activity recognition is an emerging and important area in computer vision which seeks to determine the activity an individual or group of individuals are performing. The applications of this field ranges from generating highlight videos in sports, to intelligent surveillance and gesture recognition. Most activity recognition systems rely on a combination of convolutional neural networks (CNNs) to perform feature extraction from the data and recurrent neural networks (RNNs) to determine the time dependent nature of the data. This paper proposes and designs two transformer neural networks for human activity recognition: a recurrent transformer (ReT), a specialized neural network used to make predictions on sequences of data, as well as a vision transformer (ViT), a transformer optimized for extracting salient features from images, to improve speed and scalability of activity recognition. We have provided an extensive comparison of the proposed transformer neural networks with the contemporary CNN and RNN-based human activity recognition models in terms of speed and accuracy for four publicly available human action datasets. Experimental results reveal that the proposed ViT-ReT framework attains a speedup of 2× over the baseline ResNet50-LSTM approach while attaining nearly the same level of accuracy. Furthermore, results show that the proposed ViT-ReT framework attains significant improvements over the state-of-the-art human action recognition methods in terms of both model accuracy and runtime for each of the datasets used in our experiments, thus verifying the suitability of the proposed ViT-ReT framework for human activity recognition in resource-constrained and real-time environments.

INDEX TERMS Transformer neural networks, human activity recognition, video analytics, recurrent neural networks, convolutional neural networks.

I. INTRODUCTION

One of the most important and fastest growing fields in machine learning is computer vision. Computer vision is the process of using a computer to interpret the real world through some form of sensor. Modern computer vision involves using artificial neural networks to determine patterns or features in the data from the sensor and then using these discovered features to inform other processes. This process can be used to learn from images or videos by treating each frame in the video as a separate image and performing processing on each frame and the video as a whole. In recent years, convolutional neural networks (CNNs) have gained significant traction in the computer vision community for image classification [1]. Popular CNN models used for image classification, such as AlexNet [2], VGG-16 [3], and ResNet [4], contain long chains of convolutional layers of different densities and filter sizes along with pooling and activation layers. As early as 2012, this task led to very deep implementations of image classifiers [2], and even more recently, deep CNNs with residual connections that provided even better results [4]. This final approach created ResNet [4], a residual network which uses a residual connection to recombine the extracted features with previous inputs at

The associate editor coordinating the review of this manuscript and approving it for publication was Alessia Saggese.

different time steps in the model. This approach helped create more robust models that could be trained more easily than deep CNNs without residual connections [4]. These models, once trained, can even be transferred to domains they were not initially trained in, via a technique called transfer learning [5]. This process has been done with relative success depending on both the dataset used for initial training and the domain of the testing data [6]. CNN models, including the ResNet classifier, have seen recent success and have been the dominant models used for computer vision tasks. However, the high model complexity of deep CNNs [7] is not ideal for all systems and shows a potential need for new model architectures. For example, for the internet of things (IoT), a network of physical devices which generates data from multiple different sources [8], there is a need to process the generated data efficiently. In many of these scenarios, the devices have limited resources to apply to the image classification or object detection tasks. Furthermore, newer trends in computing, such as edge computing [9], where the data is processed closer to the data source as opposed to cloud computing paradigm, necessitate more efficient and lightweight implementations. Consequently, there is a pressing need for developing lightweight human activity recognition models that can be executed efficiently on resource constrained IoT and edge devices.

Human activity recognition is a more complex task than image classification or object detection. Often the activities that are being classified involve some form of time dependence and cannot be determined from a single frame. For example, an alley-oop dunk, one of the most exciting highlights in all of sports, consists of the combination of a drive from a player, a pass to said player, a catch, and a dunk. There are few points, if any, at which a single frame of this sequence would give enough information to determine the action being performed, making it hard to classify this highlight. This is similarly true in soccer [10], as well as non-sports domains such as intelligent surveillance [11] where most actions consist of a combination of smaller actions that must all be similarly classified. These classifications are then combined to determine the overall action from the data. This is done by combining the features extracted by a CNN with the time-dependent feature analysis of a recurrent neural network (RNN).

The RNN maintains hidden states that allow for greater impact of time dependencies on their input data [12]. Many variants of RNNs exist, with the two most used in activity recognition being long short-term memory (LSTM) units, and gated recurrent units (GRUs). LSTMs have an internal hidden layer that contains multiple memory cells to 'remember' previous inputs. These cells inform three different output gates which serve as the output and updaters for future LSTMs [13]. GRUs are much the same but maintain fewer internal states and have two output gates instead of three [14]. However, GRUs maintain similar performances to LSTMs [15]. While RNNs have seen some success, they also suffer from the same complexity issues as CNNs. RNNs are also not obviously parallelizable, as an RNN layer contains multiple RNN units that compute the output of the layer sequentially. As a result, RNNs often have long training and prediction times [16]. This can lead to issues when operating in real-time systems or on edge devices. To fix this, 'attention mechanism' were constructed, which allow the model to 'pay attention' to certain parts of the input and ignore others. Attention was first used in conjunction with RNNs in sequence-to-sequence models but was later used on its own in development of the transformer [16]. The transformer was shown to not only be more efficient and lightweight, but to have similar performance to traditional methods when trained on a sufficiently large dataset. Our main contributions in this paper are as follows:

1) Elaborating and exploring the functionality and effectiveness of the transformer neural networks (TNNs).
2) Presenting the domain adoption framework showing the applicability of TNNs for human activity recognition task.
3) Proposing a recurrent TNN (ReT) to replace the computationally complex RNN in the typical activity recognition network chain.
4) Proposing a specialized vision TNN (ViT) to replace the computationally complex CNN feature extractor in the typical activity recognition network chain.
5) Evaluating the proposed ViT-ReT framework for human activity recognition using contemporary human activity recognition datasets and comparing the results to the state-of-the-art activity recognition models.

The rest of the paper is organized as follows. Section II discusses the general activity recognition model flow and current state-of-the-art activity recognition models. Section III gives background information regarding sequence-to-sequence models, transformers, and attention. Section IV shows the application of the different TNNs to activity recognition models. Section V describes the practical implementation of the TNNs created for this paper. Section VI describes the experiments performed on these models and Section VII describes the experimental results. Finally, Section VIII gives further discussion of these models and Section IX provides future research directions.[1]

II. RELATED WORKS

Activity recognition generally consists of two main phases: feature extraction on the input data sequence, and a combination of the extracted features into a time-dependent classifier. The standard version of this model uses a convolutional layer that is applied to each time step of the input. The input features extracted from this layer are kept in their original sequence and collectively input into a Recurrent layer (usually using LSTM units or GRUs), and then passed to a dense layer for classification [17]. This model structure is outlined in Fig. 1, and Fig. 2 expands on the CNN layer, looking directly at a single CNN block applied to a single

[1] https://github.com/JamesWensel/TranformerActivityRecognition

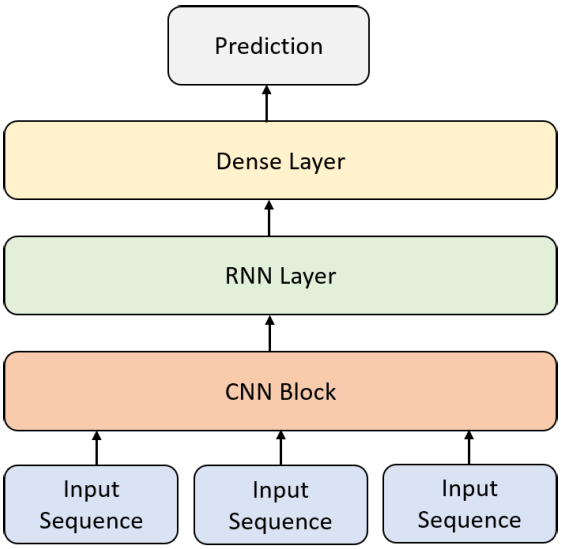


FIGURE 1. General activity recognition model flow.

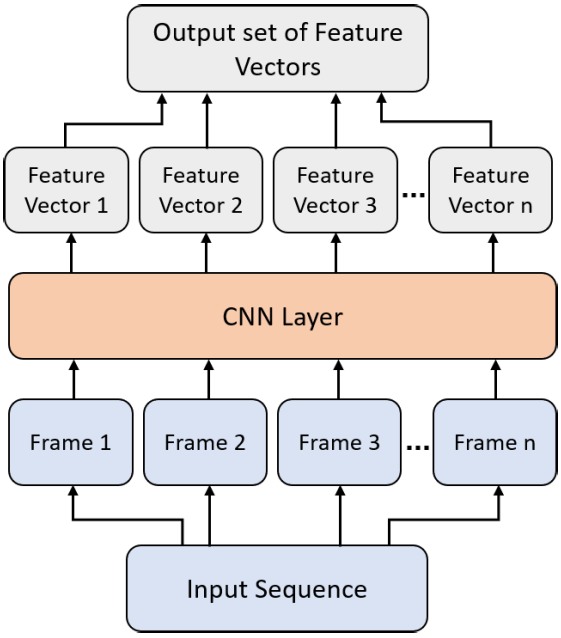


FIGURE 2. A single CNN block in the CNN layer of activity recognition model.

time step. Fig. 3 expands on the RNN layers, showing the sequential nature of RNN network architecture.

The CNN layer usually contains multiple layers of convolutional filters mixed with pooling and activation layers. These layers can be very deep for improved performance. Similarly, the RNN layer can be made up of any number of RNN units, and there are usually at least 2 RNN layers stacked on top of each other, feeding into a dense layer that can also have any number of hidden layers and internal hidden units.

While the traditional CNN and RNN model structures are very common in activity recognition, they have flaws that are inherent to the layers themselves. For convolutional layers, to get good and reliable classifications on large datasets with

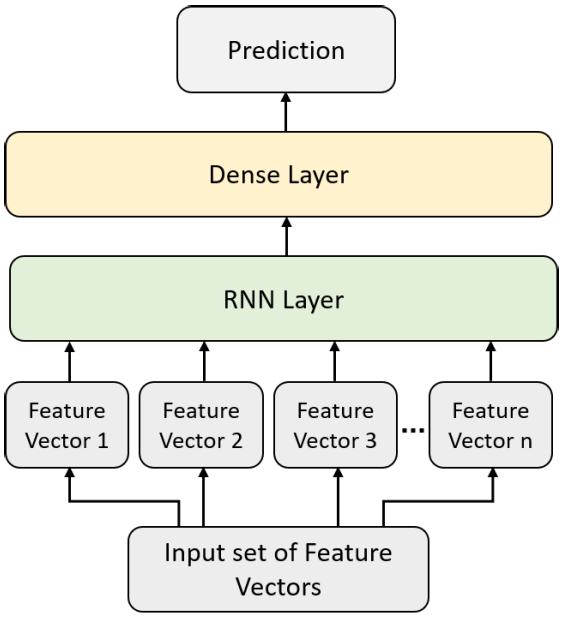


FIGURE 3. RNN and output layer of activity recognition model.

many classification categories, some form of deep and/or residual CNN is necessary [4]. This causes complexity and speed issues on devices with limited capabilities, such as IoT or edge devices. The recurrent layers have similar problems, as the final RNN in the chain is dependent on all the previous units in the layer. This means to calculate each output, all previous outputs must be calculated first, which creates a bottleneck that can impact performance [16], especially on edge devices. Sun et al. [18] have used this model structure along with an extreme learning machine (ELM), a special type of dense neural network that requires a significant number of hidden nodes compared to general dense layers. They have used the OPPORTUNITY dataset [19] to perform classification on 18 classes of gestures. Their ELM model is able to make accurate predictions on their gesture dataset, but like all general activity recognition models [10], [11], it is not suitable for resource-constrained environments or edge devices for real-time systems because of its complexity. In [20], Shotton et al. have introduced a novel approach for 3D pose recognition. Their model is capable of predicting the 3D positions of body joints using only a single depth image without relying on temporal information. Instead of directly tackling the challenging task of pose estimation, they have adopted an object recognition strategy. They have devised an intermediate representation of body parts, which transforms the complex problem of pose estimation into a more manageable per-pixel classification problem. Russo et al. [21] have updated this model structure using only deep CNNs by first splitting their video data into three streams, red, green, and blue (RGB), optical flow, and slack foreground mask to perform classification on low resolution and extremely low-resolution images. While their approach removes the RNN layers and creates a simpler model as a result, it is done by using deeper and more complex CNNs.

This results in a model that maintains the speed issues of the general activity recognition models on resource-constrained devices.

Attention has also been added to activity recognition models to improve accuracy. Ma et al. [22] have applied attention layers after feature extraction from the CNN layers and again after time series classification from GRU RNN layers. In [23], Hussain et al. have presented a hybrid approach that utilizes a vision transformer and an LSTM for recognizing human actions in surveillance videos. They have used the vision transformer for frame-level feature extraction and the multi-layer LSTM network for capturing long-term dependencies and classification of human actions. Chen et al. [24] have introduced AdaptFormer as a highly effective adaptation approach for transformers. This approach enables efficient adaptation of pre-trained vision transformers (ViTs) to a wide range of image and video tasks. It achieves improved transferability without modifying pre-trained parameters and surpasses many fully fine-tuned models in action recognition benchmarks. Its lightweight modules enable versatility, scalability, and substantial performance enhancements across multiple image and video datasets. Ranasinghe et al. [25] have introduced a self-supervised training method for video transformers that utilizes unlabeled video data. By extracting local and global spatiotemporal views of varying spatial sizes and frame rates from a given video, they have aimed to match the features of these views, and have aimed to make these features invariant to spatiotemporal variations in actions. In another work, Xing et al. [26] have conducted research on transformer models for action recognition in a semi-supervised learning (SSL) setting. They have proposed SVForemer, a novel approach that incorporates a steady pseudo-labeling framework to handle unlabeled video samples. The aim of their approach is to leverage transformer models effectively in scenarios with limited labeled data. Phong and Riberio [27] have introduced a novel model called dynamic PSO-ConvNet for action learning in videos based on their previous work in image recognition. Their approach utilizes a framework where the weight vectors of neural networks represent particles' positions in phase space. These particles share their current weight vectors and gradient estimates of the loss function. To incorporate video data, ConvNets are integrated with advanced temporal methods like transformers and RNNs. Despite their strong performance across various datasets, these models do not effectively reduce the overall complexity of activity recognition models. As a result, these models may not be suitable for resource-constrained environments that require more efficient models.

To alleviate the issues produced when using general activity recognition models in low resource environments, we propose the use of two types of TNNs, vision transformers and recurrent transformers, to replace both the CNN and the RNN layers in activity recognition models. The proposed TNNs are based on the preliminary findings of our work presented in [28], which provides different configurations of activity recognition model architectures with two different feature extraction models (i.e., ResNet50 and ViT) and the time- or sequence-dependent activity classification models (i.e., LSTM and ReT). In this paper, we have conducted significantly more extensive evaluations of our proposed TNNs than [28] and we have enhanced the performance of our proposed TNNs considerably for human activity recognition task using fine tuning. Moreover, we have evaluated our proposed ViT-ReT framework on three additional benchmark datasets and compared the results with the state-of-the-art human action recognition methods. Since our proposed TNNs, ViT and ReT can replace both the CNN and the RNN layers in activity recognition models, respectively, our transformer models serve to reduce space and time complexity of the activity recognition models, allowing them to run more efficiently on resource-constrained devices.

III. BACKGROUND

A. POSITIONAL ENCODING

Preserving the positional information in sequential data is crucial. Reordering elements can drastically alter the meaning of a sequence, as demonstrated by the example sentence ''I had baked a cake'' versus ''I had a cake baked.'' Recurrent networks naturally preserve this information, while non-recurrent networks need to ensure its conservation [16].

The initial stage of a TNN involves converting input data into a tokenized and learned representation known as an embedding. This representation is a vector where each element corresponds to a distinct learned concept [29]. For instance, in the case of a pixel sequence, a three-element vector could represent the red, green, and blue values. Each input element is transformed into an embedded vector, with each value indicating the significance of the concepts to the original element. The embedding vector can encompass various concepts, and the values are learned during training. The size of the embedding vector is a hyperparameter specific to the problem, often set to 512 in TNNs by default [16]).

Typically, TNNs utilize an embedding for representing input data, but this alone doesn't capture positional information. To address this, a positional encoding of the same length as the embedding is added to each element. This encoding represents the importance of each concept at each position in the sequence. This positional data can be learned during training, but TNNs often use a sinusoidal positional function as it has shown similar performance to learned positional data [16]. The addition of positional encoding allows TNNs to retain and utilize positional information without residual connections. It is a crucial step in the classification process for maintaining relevant positional details [16].

B. SEQUENCE-TO-SEQUENCE MODELS

Sequence-to-sequence models are advanced neural networks that map input sequences to output sequences step by step [30]. These models maintain the meaning of the input through their own representation of the data. The

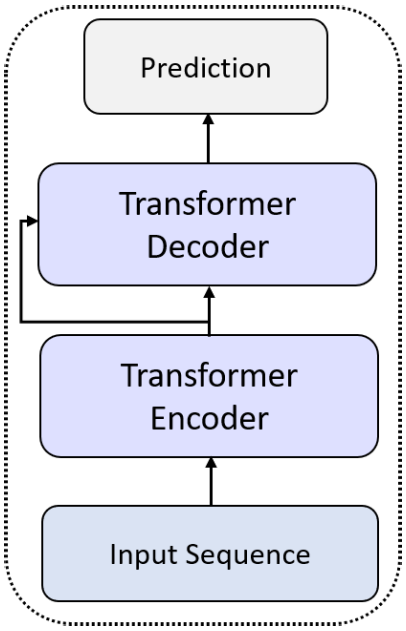


FIGURE 4. Overview of encoder-decoder structure.

encoder-decoder structure forms the basis for these models. The encoder-decoder model has two distinct parts that operate independently: an encoder and a decoder. The encoder takes an input sequence and encodes it into a vector that contains a representation of the meaning of the initial sequence. This encoding is then fed to the decoder to generate an output sequence in the desired domain [31]. Encoders can be built using RNNs to generate the encoding sequentially, while newer architectures like the *transformer* generate the entire encoding in one step using self-attention [16]. This general structure can be seen in Fig. 4. The decoder generates the output sequence by taking the encoding, and producing one element at a time. The previously generated elements are used as inputs to the decoder at the next time step, which combines these elements with the encoding to generate the next element in the sequence. This process is repeated in the decoder until the entire sequence is generated.

These sequence-to-sequence models are commonly used in natural language processing. These models involve an encoder that takes a sentence with positionally embedded words and creates an encoded representation of the sentence. The encoder's output is then passed to the decoder, which generates the target language words one at a time. The decoder utilizes the previous output words and combines them with the encoding to generate subsequent words until the entire translated sentence is formed. The encoder and decoder are trained together to complement each other. However, the basic encoder-decoder structure using RNNs faces challenges with longer sequences [32] and struggles to interpret and generate sentences longer than those used in training. These issues are addressed and improved upon by the transformer model, which will be discussed in the next subsection.

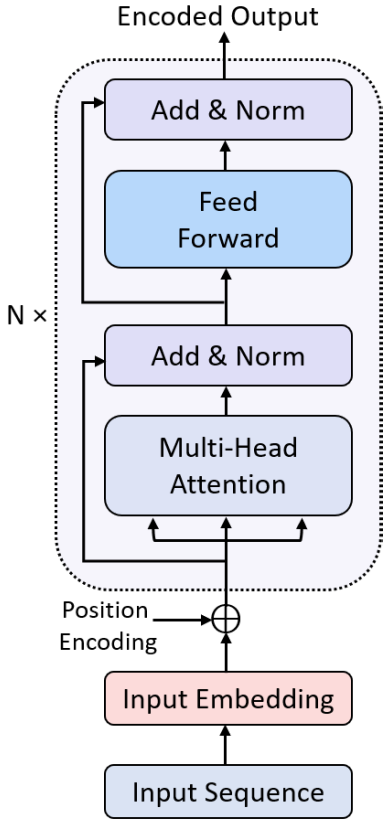


FIGURE 5. Architectural overview of transformer encoder.

Since its inception, the encoder-decoder model structure is widely used in natural language processing. Its application to activity recognition, however, may not be immediately apparent. The specific application of sequence-to-sequence models to activity recognition will be explained in detail in Section IV.

C. TRANSFORMER NEURAL NETWORK

Transformer neural network is a specialized type of sequence-to-sequence model introduced by Viswani et al. [16]. They introduced a special type of self-attention called ''scaled dot-product attention'', which replaces RNNs in both the encoder and decoder, enabling parallel processing of the entire input sequence during encoding. This results in significant speed improvements during training and prediction. While the decoder remains sequential, the use of TNNs eliminates a major bottleneck. TNN-based models, including BERT [33], exhibit robustness and achieve high accuracies similar to other sequential models.

TNNs have a structure similar to the general encoder-decoder model, but with key differences. Numerical representations are assigned to input sequences, and positional information is added for embedding. Self-attention is applied, using ''multi-headed attention'' blocks to focus on different aspects of the input sequence and preserve varying levels of meaning. The output from the attention blocks and a feed-forward network is combined using residual

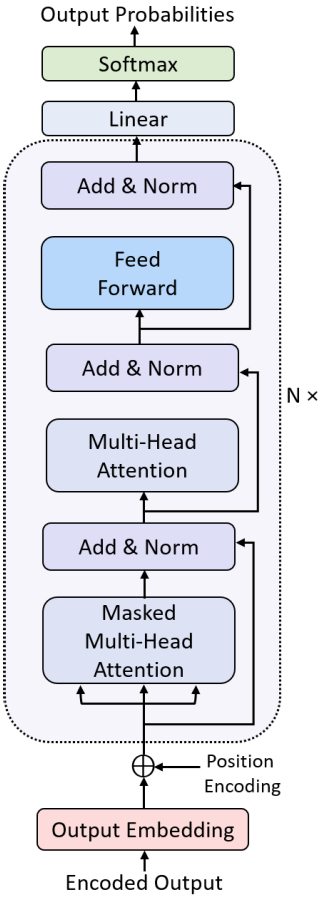


FIGURE 6. Architectural overview of transformer decoder.

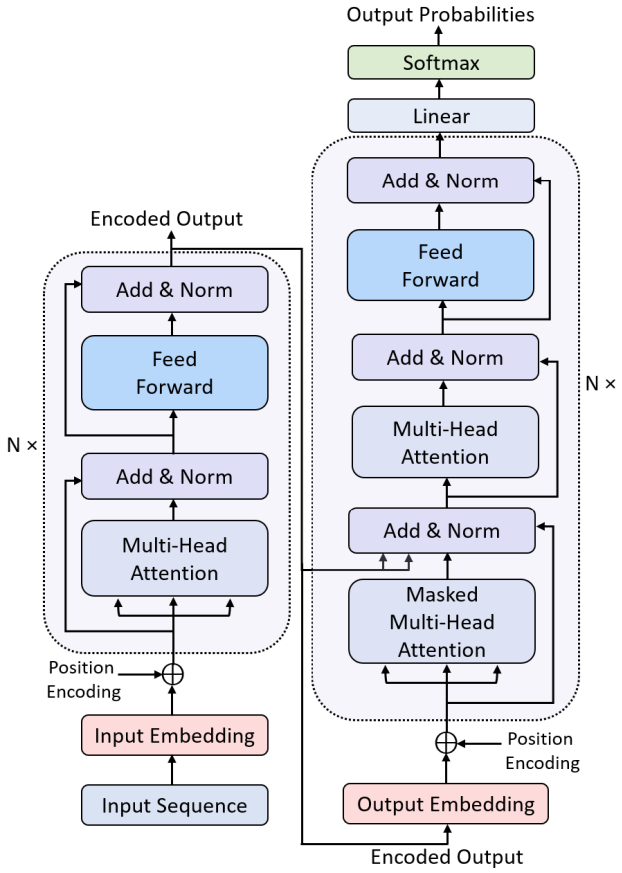


FIGURE 7. Detailed architectural overview of transformer.

connections [4]. The result is normalized and combined with the input, enhancing the encoder's robustness and learning capabilities. This structure can be visualized in Fig. 5. In the Transformer model, the decoder receives the encoded output sequences. It applies multi-headed masked attention to handle current and previous outputs, and another multi-headed attention block combines the input with the encoder's output. The output of this layer is fed to a feed-forward network, like in the encoder block. There are again residual connections around both multi-headed attention blocks and the feed-forward network that are combined with the original input and normalized. The decoder output then goes through a linear layer with softmax activation to produce a probability distribution as the output. This is then passed back to the beginning of the decoder as the input of the next layer. The decoder structure is shown in Fig. 6, and the overall structure of the transformer is depicted in Fig. 7.

TNNs have quickly become the most important neural network for sequence-to-sequence tasks. While it is clear, TNNs are extremely powerful, it is still not obvious how TNNs can be applied to video data or activity recognition as a whole. This process will be described in detail in the Section IV.

D. SELF-ATTENTION

Self-attention is a process by which a neural network learns which parts of the input it should focus on. While there are many kinds of attention, such as additive attention [32], this section will focus on scaled dot-product attention [16] as it is the basis for the transformer models created by Viswani, et al. [16]. Attention is derived from database queries. The process begins with a set of key-value pairs and involves mapping a set of queries to those keys and combining the result with the initial values to get the output. When given a set of queries Q, keys K, and values V, where Q, K, and V are all matrices, and d_k is the number of dimensions of the keys matrix, the scaled dot-product attention can be calculated by eq. (1):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

While not immediately obvious, this equation is based on the equation for cosine similarity [34]. The cosine similarity of two vectors A and B is given by eq. (2):

$$CosineSimilarity(A, B) = \frac{A \cdot B}{||A||\,||B||} \tag{2}$$

The above equation (eq. (2)) calculates the similarity between vectors A and B by taking the dot product and scaling it with their magnitudes. It yields a value between +1 and -1,

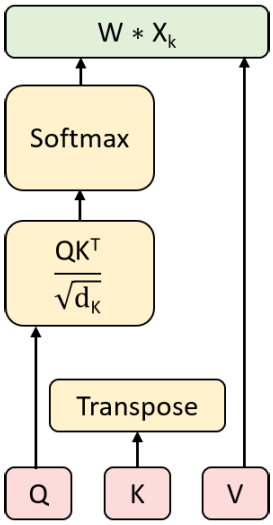


FIGURE 8. Scaled dot-product attention.

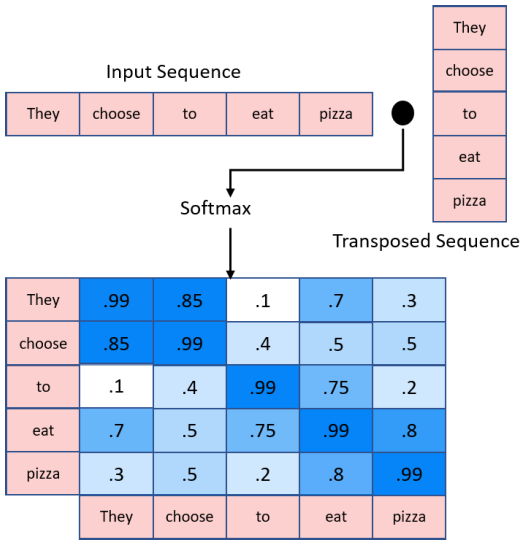


FIGURE 9. The process and results of scaled dot-product attention.

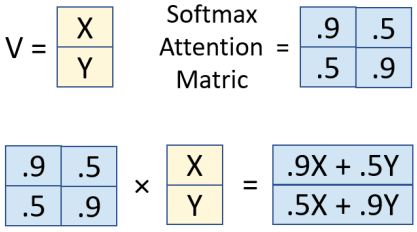


FIGURE 10. Example weighted values matrix.

resembling the cosine of the angle between the vectors. This is a simplified version of the equation presented in eq. (3).

$$\frac{QK^T}{\sqrt{d_k}} \tag{3}$$

In this case, the queries matrix and the transpose of the keys matrix serve as A and B, respectively, and instead of scaling by the product of their magnitudes, the result is scaled by the root of the number of dimensions of the keys matrix. This scaling value was chosen as an extension to dot-product attention to give it better performance as the keys matrix becomes larger. This was done because the dot product at higher dimensionalities of the keys matrix will be very large and would have a negative effect on the performance of the softmax function otherwise [16]. The result of the scaled dot-product is then passed through a softmax layer, giving us a probability matrix that is used as the weights for the values matrix. This process, including masked attention, is illustrated in Fig. 8. This can also be written to resemble typical neurons more closely, as in eq. (4):

$$y_k = Wx_k, \tag{4}$$

where

$$W = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

and

$$x_k = V$$

The encoder and decoder attention blocks have different inputs for queries, keys, and values. In the encoder, all the three matrices are identical copies of the input sequence. This might seem counterintuitive, but it is a crucial step in capturing the meaning of the input. For instance, in the sentence ''They chose to eat pizza'', each row element in the keys matrix represents a word, while the data corresponds to its learned embedding. When performing the scaled dot-product, the resulting attention matrix shows the similarity between each word pair. Applying softmax to the attention matrix yields similarity scores between 0 and 1, indicating the degree of likeness between words. For example, a score of 0.8 between ''eat'' and ''pizza'' suggests a strong association, while a score of 0.1 between ''they'' and ''to'' indicates a weaker relationship. This process as well an example softmax attention matrix is shown in Fig. 9. The softmax attention matrix from the encoder is used to weigh the values, resulting in a final matrix where each row corresponds to the original words in the input sentences. The data in each row represents a weighted dot product indicating the impact of each word on the word at that row index.The output of this layer is demonstrated in Fig. 10.

The decoder attention blocks function similarly to the encoder, but with two key differences. Firstly, the queries, keys, and values in the masked attention block are based on the current output sequence (or in the case of training, the correct output sentence). The attention matrix is masked to prevent the decoder from accessing future elements in the sequence. This ensures that during training, even if earlier predictions were incorrect, the decoder has the correct information to predict subsequent elements. The mask hides information beyond what the decoder has already generated. This simulates the practical scenario where the decoder only has access to previously generated elements when making

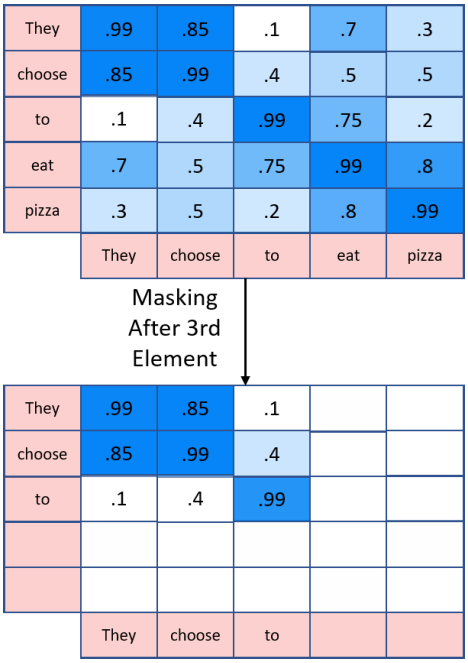


FIGURE 11. Masking of a Softmax attention matrix after the third element.

predictions. An example matrix showing the masking process can be seen in Fig. 11.

Finally, the masked attention block generates a relevance matrix containing the relevance of each word in the currently generated output sequence to every other word in the sequence. This relevance matrix will contain all contextual information that is currently known about the output sequence. This matrix is used as the values for the second attention block in the decoder, and the encoded output of the encoder block is used as the queries and keys. Using the encoded output as the queries and keys allows us to calculate a set of weights directly from the encoder output, and when those weights are applied to the relevance matrix from the first attention block, it will apply the encoded contextual information of the input sequence to all currently known contextual information about the output sequence. This gives us a contextual matrix that contains all known context from the input sequence and current output sequences. This is then combined with a residual connection, normalized, and passed through one last feedforward network, normalized one last time, and passed into a linear classification layer with softmax activation function to predict the next element in the output sequence. These final layers all apply extra preprocessing to their inputs to help improve consistency of the output predictions.

E. MULTI-HEADED ATTENTION

The attention mechanism in TNNs is "multi-headed" as it employs multiple attention blocks in parallel. Each attention head has its own learned linear projection for queries, keys, and values, which enables it to focus on different parts of the input data. This approach enhances the model's ability

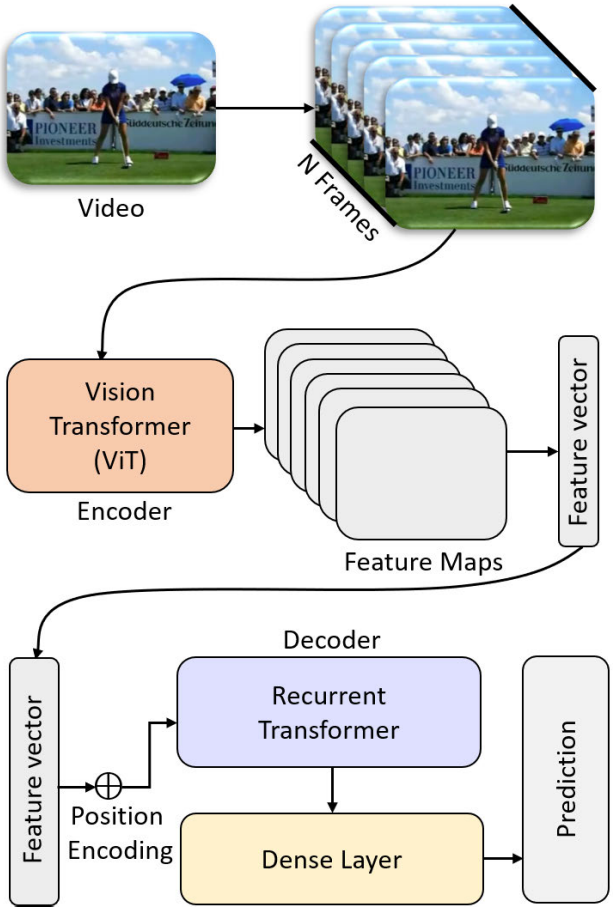


FIGURE 12. Overview of the activity recognition pipeline using the transformer for time series classification instead of the typical RNN.

to capture diverse aspects of the input sequence, leading to more accurate classifications.

For a more detailed description of both TNNs and scaled dot-product attention, see Viswani et al., [16]. Much like the TNNs, scaled dot-product attention is the current state-of-the-art attention method that is applied to many domains, and is even used beyond TNNs [35]. However, the direct application of TNNs to activity recognition is still not obvious.

IV. PROPOSED METHODOLOGY

A. RNN LAYERS

The application of TNNs for the RNN layers of the generalized activity recognition model seen in Fig. 1 and Fig. 3 is the simplest of the two applications. While TNNs were designed for, and most used for, machine translation tasks, there is nothing inherent to the model requiring word embeddings to be used as the inputs to the model. The encoder can take any sequential input, so long as the relevant positional information has been encoded in the sequence, the encoder will extrapolate out the relationships between all elements in the sequence. So, if instead of a sequence of words, the input was a sequence of frames, where each frame was embedded with the information relevant to its order in the video, the encoder would be able to find the

relationship between all frames in the video and use this to create an output containing the ''meaning'' of the video. While the application of the encoder to get meaning from the frames means video data can be used as the input to the encoder, an important question arises: What sequence will be generated by the decoder? Shockingly, the answer is none.

In the case of video data (or any time series data that does not generate an output sequence), there is no need to use the decoder at all. Instead, the output of the encoder is fed directly into a flattening layer to make the data 1-dimensional, and then to a dense layer to make predictions. Conceptually, this structure makes sense. After the multi-headed attention layer, the Encoder has created a matrix from the initial sequence that has been weighted by what the model has determined is important about each element. This is passed into a Feed-Forward layer to create an encoded output, which can be thought of as an encoded representation of the ''meaning'' of the input sequence. So instead of using said ''meaning'' to generate a sequence, this ''meaning'' can instead be used to classify the input, which is exactly what our proposed model does. Further, because our proposed model does not use the decoder, there is no bottleneck in this section. Unlike when RNNs are used to generate the final representation of the data, there is no need to wait on the completion of any chains of values waiting on previous states. Even in the application of the TNN to sequence-to-sequence models, the data gets bottlenecked at the decoder step. Since all values needed to compute the output of the encoder are present at the start of computation it is easily parallelizable, which allows for increased computation speed. This model structure can be seen in Fig. 12.

While the application of TNNs to any other time series data would follow similarly, it is not easily apparent how to apply this to non-time dependent data, such as single images, in a way that it could replace CNNs.

B. CNN LAYERS

TNNs needs to be applied to data in a sequence. However, the TNN itself does not learn anything from the positional data of the input sequence. This is all done externally by the positional embedding of the input sequence data. So, if the non-time series data was somehow made to be sequential, with the information about its locality still applied by the positional embedding, then the data could be used in the same manner as any time series data. It is by this process that the TNN is applied to the individual frames of a video, or to any other non-sequential data. To turn an image into sequence data for a TNN, it is first split into patches, or smaller subsections of the original image. These patches are then grouped in sequence to be used as the input to the TNN. This process can be seen in Fig. 13. The size of these patches is a problem dependent hyper-parameter similar to CNN filter size. For images with large object scales, larger patches may be used, but for images with much smaller scale, small patches are necessary.

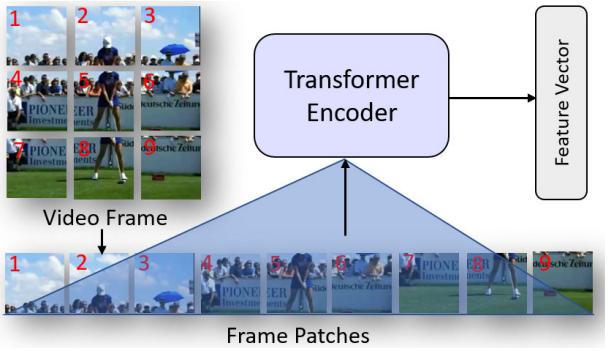


FIGURE 13. Splitting a frame into patches and passing them to a transformer for classification.

The TNN will then perform a pseudo feature extraction, by determining the internal ''meaning'' of the image, and using a dense layer to classify this output, in the same manner as for time series data. The patching of an image and using it as a sequential input to a TNN is the process taken by a vision transformer [36]. Vision transformers have been shown, when trained on a sufficiently large dataset, to perform as well as state-of-the-art deep residual CNN networks [36], while remaining more lightweight and efficient than long CNN chains. The last important step in the application of TNNs for activity recognition is utilizing the vision transformer alongside the recurrent transformer.

C. RNN AND CNN LAYERS

An important note before this section is it is not necessary to implement TNN models for both halves of the activity recognition model. The recurrent transformer (ReT) can replace the RNN layers only and train on the features extracted by a deep residual CNN more efficiently and with as much accuracy as the typical LSTM or GRU chains. Alternatively, the vision transformer (ViT) can be used to replace the deep residual CNN to perform feature extraction from the frames and pass the data to the RNN layers. Still, combining the ReT and ViT will prove to alleviate the largest amount of complexity and efficiency issues encountered by current activity recognition models.

To use a ViT, it should first be trained separately on a large image recognition dataset, such as ImageNet [37]. The ViT needs a large dataset to train, and there are no sufficiently large activity recognition datasets that would allow the ViT to achieve adequate performance if first trained in conjunction with a ReT. Once trained, it will be transferred to be the backbone of the feature extraction section of the new activity recognition model. Then, for a specific video input, a copy of the trained ViT will be applied to each frame. The output from this layer will be a series of encoded image data generated from the patches of each frame in the sequence. The encoded image data will be in the same order as it was in the initial video, and it is this sequence that will be classified by the Transformer. In the next section, we elaborate our implementation of both a ReT model and a ViT model.

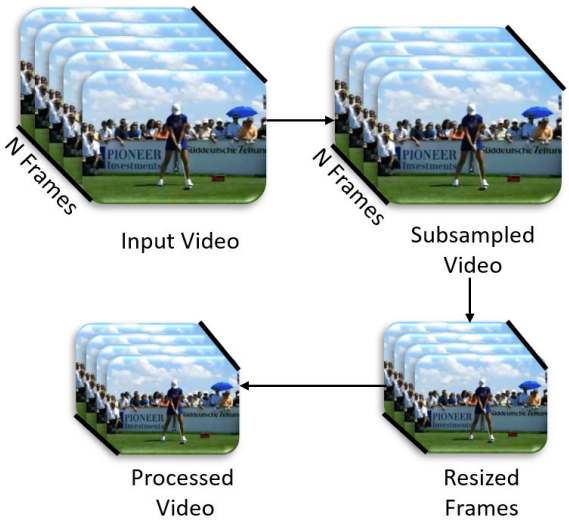


FIGURE 14. Video preprocessing steps.

V. IMPLEMENTATION

We have built our model in Python 3.8.4 using TensorFlow through the Keras API, utilizing Numpy for data manipulation and OpenCV for video processing. The implementation of our model comprises of three main stages: (i) data preprocessing, (ii) custom layers, and (iii) model controllers. This section discusses the key components of our implementations. Afterwards, the important hyperparameters for both ReT and ViT models will be discussed, followed by an explanation of the ViT used.

A. DATA PREPROCESSING

The first step in training a ViT is preprocessing the video files in the datasets to be ready to train the model. This is done by subsampling a selected number of frames from the videos to create consistent length sequences. The entire video is not used for a few reasons. As most consecutive frames in a video contain the same data as their neighboring frames, using an entire video would significantly increase training complexity with little to no tangible benefit to accuracy. Our utilization of subsampling also allows for the use of a video of any length (even if a video has more frames than the number used for the ViT) since it will be shortened to the correct sequence length for the model. The selected frames are then resized to 224×224 pixels, which is the default size for ResNet50 [4]. The ViT has a default image size of 384×384 [36], but it can be used for images of any size so only 224×224 images were used for testing to ensure the same testing conditions for both models. After resizing all selected frames, they are appended to an array containing all processed video files, and the associated training label is appended to the same index in a labels array for later training and testing. These two arrays, along with an array containing the file path to each video file are saved for later use. This process can be seen in Fig. 14.

Instead of completing these preprocessing tasks before training, they could instead be applied as layers at the beginning of the model, but we choose not to do this. This is done to save time when training models, as the data processing would need to be completed each time the models are trained, and for the dataset used this takes a lot of time (sometimes over 40 minutes on its own). This also allows for more accurate data collections from the models as each layer is then able to be monitored individually instead of only being able to watch the entire model at once.

B. CUSTOM LAYER

To implement the ReT and ViT, three custom layers were created: PositionalEncoding, Encoder, and Patches. A functional wrapper, the BuildEncoder function, is also created to make creation of an encoder more seamless. Finally, a data generator is created to facilitate the use of a large dataset.

1) POSITIONAL ENCODING

This layer applies the positional data to the input sequence. It takes as an input the length of the sequence and the size of the linear project used on the input. The length of the sequence will either be the number of frames extracted from each video, or the number of patches extracted from an image, depending on whether the ReT or ViT is being used. The layer creates a positional encoding space and adds the associated positional data to each element. It also linearly projects the input to any size chosen through the linear layer (usually the size is left alone or slightly downsampled, but any could be selected).

2) ENCODER

This layer serves as the transformer. Since none of the data generates a sequential output, the decoder is not implemented. The layer first takes as input the size of the input (which is the same as the size of the linear projection from the PositionalEncoding layer). The input shape will be conserved through the encoder as it is uses the size of the final dimension of the input vector as the shape of the final dense layer in the encoder. The next input is the shape of the internal dense layer. This can be any size to allow for more hidden neurons, but is usually set to twice the input size. The next input is the number of heads for the multi-headed attention in the encoder, usually 6 but could be any value. Finally, the activation function for the internal dense layer can be passed but is by default an ReLU activation function. This layer first computes the attention matrix from the given inputs using the built in MultiHeadAttention function from Keras. The output of this is the weighted values matrix. This is then normalized and passed into the dense layer, and normalized again, at which point we have the predictions or feature extracted vector depending on which TNN is being trained.

3) PATCHES

This layer splits an image into patches. It is given an input of the length of one side of the patch in pixels (8 was used but could be any value, smaller patch sizes will create more patches). This layer uses TensorFlow's built in

patch extraction function to create patches, then reshapes the resulting tensor to maintain the correct shape of the output. If the input is a (Batch size, 64, 64, 3) shaped image tensor and patches are 8×8, it will return a tensor with shape (Batch size, 64, 192) containing all 64 created patches from the image [$192 = 8 \times 8 \times 3$]. The patches are not kept in a (8,8,3) tensor, but instead the pixel data from each pixel in the patch is placed into a 1-Dimensional tensor. This is done to eliminate future reshaping of each individual patch.

4) BUILD ENCODER

This is a functional wrapper used to build a specific TNN layer according to user specifications. It takes as input the sequence length (number of frames) and the size of the final dimension of the data tensor. It also takes as input all relevant information for the PositionalEncoding layer (embedding layer shape) and Encoder layer (dense layer shape and number of attention heads). It then creates and returns a Keras Model Transformer according to this specification. To use this model, it will then be passed an input tensor with the expected shape, which will be passed to the positional encoding. After the positional data is applied, it will be passed to the Encoder where it performs attention and creates a weighted values matrix which is normalized, passed through a dense layer, and a softmax layer to receive a classification from the model.

5) DATA GENERATOR

A generator is used to train and test the models on different datasets. After initialization, it is called instead of accessing the dataset directly. When called, it returns the batch from the datset at the current index. This allows the system to only store the current batch of data in RAM instead of the entire dataset, making more efficient use of the available resources and allowing for larger datasets to be used. Without a generator, the entire dataset must be loaded into RAM, as well as all parameters for the model that is training. This puts an unnecessary burden on the resources of the computer and does not allow for large datasets. However, there would be an increase in training and testing speed if the dataset could be loaded into RAM in its entirety as with a generator some time is required to access each subsequent batch of data.

C. MODEL CONTROLLERS

The model controllers are files that are used to either create the models, run them, test them, or output data from them. Any parameters that are not mentioned here have been mentioned previously and will be covered again in the hyperparameters subsection. The pictorial overview of the model controllers are shown in Fig. 15.

1) CREATE MODELS

The model creation file has 5 functions, each of which creates a different model. The ResNet_Model function generates an instance of the pretrained ResNet50 with ImageNet weights and global average pooling. This is applied over an input sequence using the Keras TimeDistributed wrapper, and is used to perform general feature extraction. The LSTM_Model function generates a specified number of LSTM layers following the general activity recognition RNN layer architecture. All layers will have the same specified number of LSTM units. The Vision_Transformer_Model function creates an instance of a pretrained Vision Transformer model. The Untrained_VisionTransformer_Model function creates a model ready to train for image recognition. It will have a specified number of transformers stacked on top of each other. Finally the Transformer_Model function creates a Transformer model with the specified number of transformer layers for classification of video data.

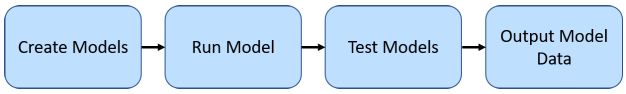


FIGURE 15. The graphical overview of the model controllers.

2) RUN MODEL

The model running file contains three functions that are used to train, test, and evaluate a model, and to make predictions. First, the FitModel function is passed a model and the data to train on, as well as the batch size and the number of epochs to use. It trains the given model with the data provided, with the specified batch size and number of training epochs. The PlotWholeModel function is a helper function that visualizes a model as well as all functional layers. When a functional layer is found, the function calls itself, passing the functional layer instead of the whole model to dispaly the model hidden in the functional layer. PredictModel is used to make predictions on a model without training by passing it a model (which is assumed to be pretrained) and a features dataset to make predictions on.

3) TEST MODELS

Testing of the models is done using 2 main files: a file that writes a test configuration JSON object (tests.json) and a file that reads the object and runs tests according to those specifications. The object contains a list of setup values and a tests to perform, indexed by the model that is being tested and the specific attribute of that model that is being tested. A list of default values for each important parameter is also included to allow for easy changing of the defaults in subsequent testings. This object is read by a TestModel file which loops over all the test types included in the configuration object file and uses the default and setup values alongside the specific model configuration being tested to create a new model and test it. The results of these tests are output and written to files for later interpretation.

4) OUTPUT MODEL DATA

The output model data file contains a function that creates progress bars over the command line for better visual representation of the progress completed by tasks. It also

contains a class definition with 13 functions. The class creates an object that is capable of storing timing and memory data before and after any event by calling an associated method for the object. This allows all other files to be much cleaner as they do not need to manually track their own data for output. It is also capable of writing all stored data to the command line, to a text file, and to a csv file. This allows for easy recording of test data for later interpretation.

D. HYPERPARAMETERS

This section contains a listing of each hyperparameter for the model, and a brief description of each, including the values used when training the model.

1) *SEQUENCE_LENGTH*: The number of frames to keep from each video (usually 20).
2) *IMAGE_HEIGHT*: The height in pixels to resize the video frames for activity recognition (usually 224).
3) *IMAGE_WIDTH*: The width in pixels to resize the video frames for activity recognition (usually 224).
4) *projection_dim*: The size of the linear projection of the input sequence by the PositionalEncoder (usually left at or as close to input size as possible)
5) *dense_dim*: The number of neurons in the TNN's feed-forward network's hidden layer (this does not change the output shape as it is later projected back to the input shape; it is normally $2\times$ input shape)
6) *activation*: The activation function for Transformer's Feed-Forward Network's hidden layer (defalts to ReLU)
7) *num_heads*: The number of heads for multi-headed attention.
8) *patch_size*: The length of one side of the patch to create in pixels (usually 8 [creates 8×8 patches from image]).
9) *LSTM_layers*: The number of LSTM layers in an LSTM model.
10) *LSTM_units*: The number of LSTM units in each layer of an LSTM model.
11) *transformer_layers*: The number of transformer layers for a recurrent transformer or vision transformer model.
12) *cateogries*: The number of video or image categories to train on (determined by the dataset being used)
13) *batch_size*: The number of examples to train on before updating the weights of the model.
14) *epochs*: The number of times to train the model using the entire training dataset.

E. VISION TRANSFORMER

To allow for quicker training and testing of models for activity recognition, we have used a pretrained implementation of the vision transformer [38]. This implementation utilizes the vision transformer weights pretrained on ImageNet by Steiner et al. [39]. We have selected these pretrained weights to ensure that both ResNet and the vision transformers are trained on the same data in the same manner as Dosovitskiy et al. [36] when they initially created the vision transformer. These model implementations are then used for activity recognition in tandem with the other model implementations as outlined in this section. The interaction between all files in this implementation can be seen in Fig. 16.

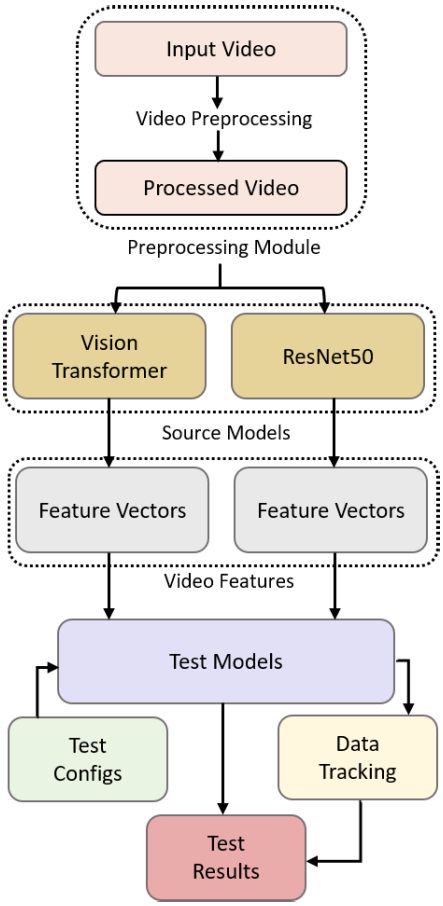


FIGURE 16. Basic dataflow of model testing.

VI. EXPERIMENTS

To compare the typical activity recognition model to the transformer activity recognition model, we have built multiple different models and have trained on various subsets of the UCF 101 dataset, which is one of the largest and most challenging activity recognition datasets. The UCF 101 dataset is a set of 13,320 human activity videos, each belonging to one of 101 different categories [40]. Each category contains approximately 130 videos, each approximately 10 seconds in length. The categories can be divided into 5 action types: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, and Sports. Besides, we have conducted extensive comparative assessments on three widely used human action recognition datasets that include YouTube Action, HMDB51, and UCF50 datasets.

In this work, we have conducted different sets of experiments to evaluate the performance of our proposed ViT-ReT framework. First, we have evaluated the performance generalization of our proposed method, where we have trained our model on different scales of data from UCF101 dataset

and presented the performance in Table 1-4. Next, we have conducted comparative analysis of our method with the state-of-the-art human action recognition methods in Table 5-8. Finally, we have conducted run-time and memory efficiency analysis experiments to assess the inference efficiency of the proposed framework in comparison with the state-of-the-art methods. To assess the performance generalization of the proposed method on different scales of training data, we have used a subset of 4, 20, and 50 action categories alongside the entire 101 action category dataset to show the models' performance and scalability over datasets of different sizes. The subset with 4 categories has 478 videos, the subset with 20 categories has 2500 videos, the subset with 50 categories has 6,567 videos, and the total dataset (101 categories) has 13,320 videos. Models are trained and optimized on each data subset. The categorical cross entropy loss and accuracy of the training are tracked and compared in the results section (Section VII). The video files are subsampled to include only 20 total frames from each video to standardize the data and reduce the overall model complexity. The frames are then resized to 224 × 224, which is the standard image size for ResNet50.

The activity recognition models proposed in this work are trained using both ResNet50 and ViT as feature extractors. Both feature extractors are pretrained using ImageNet 2012. Direct comparisons of ResNet50 and the vision transformer are not performed as extensive testing of these two models using ImageNet 2012 was performed by Dosovitskiy et al. [36]. ResNet50 and the vision transformer are trained using the same input sequence length and video frame size.

The LSTM and ReT models are trained on both ResNet50 and ViT features and optimized according to their model parameters. For the LSTM model, the number of LSTM layers and the number of LSTM units per layer are changed, while for the Transformer, the number of Transformer Layers, the size of the feature embedding, the number of neurons in the internal dense layer, and the number of attention heads are all changed. These values are initially assigned using default values, but the parameter value that optimizes loss for the current data subset is then used for subsequent tests within that subset. For example, the LSTM model was tested with 1, 2, 4, and 6 layers. If using 2 layers resulted in the smallest loss while training the 4-category data subset, then when testing the optimal number of LSTM Units in the 4-category data subset, 2 LSTM layers would be used. After testing all model parameters, the values are then set back to the original default values for the next testing set.

A breakdown of all the tests performed can be seen in Fig. 17. It shows a list of all tests performed for both the LSTM and Transformer models, as well as showing that they are repeated 8 times each. The tests for each model can be seen explicitly below:

1) LSTM
    a) Layers: 1, 2, 4, 6
    b) Units: 32, 64, 128, 256, 512

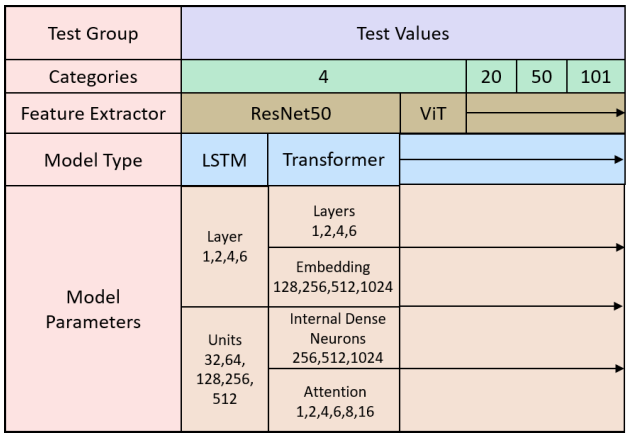


FIGURE 17. An overview of all tests performed.

2) Transformer
    a) Layers: 1, 2, 4, 6
    b) Embedding: 128, 256, 512, 1024
    c) Internal Dense Neurons: 256, 512, 1024
    d) Attention: 1, 2, 4, 8, 16

These tests are performed 8 times each. The tests are performed once for each type of feature extractor, and each feature extractor is used once for each different subset of the data. This means the LSTM will be tested 8 times (4 categories ResNet, 4 categories ViT, 20 categories ResNet, 20 categories ViT, 50 categories ResNet, 50 Categories ViT, Total dataset ResNet, and Total Dataset ViT), and the ReT will be tested 8 times in the same manner. This is done to find the model configurations with the best loss values.

After, the best model configurations will be compared based upon their loss and accuracy on each of the data subsets to show accuracy on different dataset complexities. Then the training time and throughput time will be compared for the best configurations of the LSTM and ReT. Next, the throughput time of the ViT-ReT model and the ResNet50-LSTM model will be compared. Finally, the memory usage of the two models will be compared.

In our experiments, the model is trained using a shuffled 80 percent of the dataset. After each epoch of training, the training dataset is shuffled to help fight overtraining on the training dataset. For throughput testing, each model is tested by making predictions on the entire data subset at once. It is worth noticing that, instead of creating a model that would perform the entire process end-to-end, each stage of model processing was handled separately. The data is preprocessed first and saved for later use. Secondly, neither the ReT nor ViT is implemented in parallel. This means that these tests do not utilize all the potential benefits of these models, and the models do not train or predict at the speeds that are possible if a parallel implementation is created. Parallel implementations of the transformer are the most obvious possible extension of this research which serves as a basis to show the viability of the transformer in activity recognition

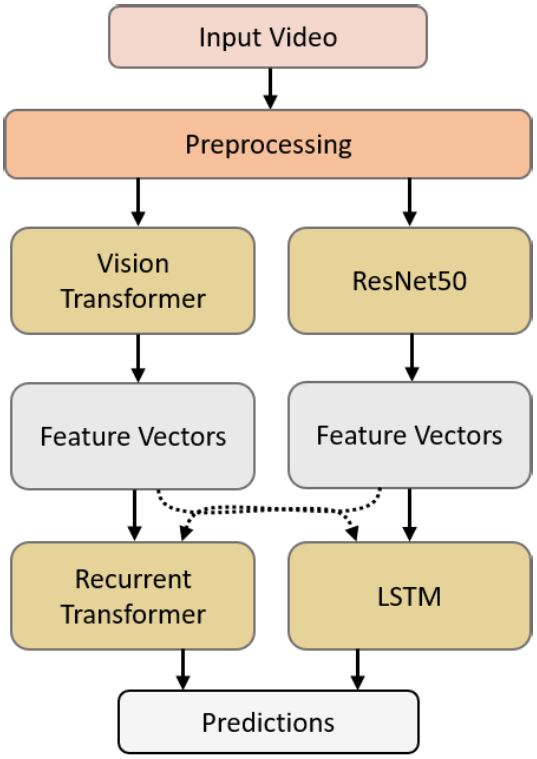


FIGURE 18. Architectural overview of the proposed ViT-ReT framework.

tasks. The overall model flow for these tests can be seen in Fig. 18.

VII. RESULTS

We have split the experimental results into eight sections. First, the accuracy of our models with the best loss values are shown for each of the data subsets. We then present comparative results of our proposed framework with the state-of-the-art methods. Afterwards, we present category-wise accuracy assessment of ViT-ReT framework on different benchmark datasets. We then present the timing data for the LSTM and ReT models, comparing their training time and throughput time for the entire data subset using our best model configuration. Subsequently, we discuss the throughput times comparing the ResNet50-LSTM model to the ViT-ReT model. Next, memory usage of ViT-ReT versus ResNet50-LSTM is discussed. Afterwards, we provide a run time comparison of ViT-ReT framework with the state-of-the-art methods. Finally, the results from all sections are discussed together. We have performed our testing using different datasets as discussed in Section VI. We have first converted the videos to sequences of 20 frames, with each frame resized to 224 × 224 pixels. All training is performed with a batch size of 4 over 50 epochs.

A. ACCURACY

For this section, we trained multiple different model configurations on different subsets of the UCF 101 dataset, and the configurations with the best loss values found are used for discussion. First, let us look at the models with the best loss values for the 4 Action dataset. The categorical cross entropy loss and accuracy for the optimal LSTM and Transformer model for each feature extractor is shown in Table 1. Best configurations:

- ResNet-LSTM: 2 LSTM Layers, 256 Units
- ResNet-ReT: 1 ReT Layer, 128-Dim Embedding, 256 Neurons, 16 Attention Heads
- ViT-LSTM: 1 LSTM Layer, 256 Units
- ViT-ReT: 1 ReT Layer, 128-Dim Embedding, 256 Neurons, 8 Attention Heads

TABLE 1. Accuracy with the 4 actions database.

4 Actions	ResNet50		ViT	
	Loss	*Accuracy*	*Loss*	*Accuracy*
LSTM	$4e^{-5}$	100%	0.50	81.25%
ReT	$2e^{-4}$	100%	0.56	81.25%

TABLE 2. Accuracy with the 20 actions database.

20 Actions	ResNet50		ViT	
	Loss	*Accuracy*	*Loss*	*Accuracy*
LSTM	0.06	98%	0.65	84.2%
ReT	0.06	100%	0.72	80.0%

From these results we can see the LSTM and ReT models perform similarly on small datasets regardless of the feature extractor used. While the models using LSTM layers had slightly smaller loss values, there was no change in accuracy between the LSTM and ReT. There was, some loss in accuracy when switching from ResNet50 to the ViT, but the loss in accuracy is the same for both LSTM and ReT. Next, the results for the 20 Action dataset are shown in Table 2. Best configurations:

- ResNet-LSTM: 1 Layer, 64 Units
- ResNet-ReT: 1 Layer, 128-Dim Embedding, 1024 Neurons, 16 Attention Heads
- ViT-LSTM: 2 Layers, 256 Units
- ViT-ReT: 1 Layer, 128-Dim Embedding, 256 Neurons, 8 Attention Heads

Here, we begin to see the impact of increased dataset complexity and size. The loss of both models has increased by more than 100x their previous loss values when using ResNet50. However, this has not come with a significant decline in accuracy. The two models are still able to achieve close to 100% accuracy with ResNet50. Interestingly, the loss of each model does not increase significantly, and the accuracy of each model does not decrease significantly from the smaller dataset when using the ViT, and even increases for the LSTM. Next, the results for the 50 Action dataset are shown in Table 3. Optimal configurations:

- ResNet-LSTM: 1 Layer, 128 Units
- ResNet-ReT: 1 Layer, 128-Dim Embedding, 1024 Neurons, 4 Attention Heads
- ViT-LSTM: 2 Layers, 512 Units

TABLE 3. Accuracy with the 50 actions database.

50 Actions	ResNet50		ViT	
	Loss	*Accuracy*	*Loss*	*Accuracy*
LSTM	0.18	94.1%	0.28	90.98%
ReT	0.27	92.5%	0.46	90.73%

TABLE 4. Accuracy with the 101 actions database.

101 Actions	ResNet50		ViT	
	Loss	*Accuracy*	*Loss*	*Accuracy*
LSTM	0.33	93.2%	0.49	89.06%
ReT	0.39	93.2%	0.63	88.14%

TABLE 5. Quantitative comparative analysis of the proposed ViT-ReT framework with the state-of-the-art action recognition methods on YouTube Action dataset.

Method	**Year**	**Accuracy** (%)
Multi-task hierarchical clustering [41]	2016	*89.7*
BT-LSTM [42]	2018	85.3
KFDI [43]	2020	79.4
Dilated CNN+BiLSTM+RB [44]	2021	89.0
Local-global features + QSVM [45]	2021	82.6
3DCNN [46]	2022	85.2
ViT-ReT (Proposed)	2023	**92.4**

TABLE 6. Quantitative comparative analysis of the proposed ViT-ReT framework with the state-of-the-art action recognition methods on HMDB51 dataset.

Method	**Year**	**Accuracy** (%)
Multi-task hierarchical clustering [41]	2016	51.7
STPP+LSTM [47]	2017	70.5
Optical flow + multi-layer LSTM [48]	2018	72.2
TSN [49]	2018	70.7
IP-LSTM [50]	2019	58.6
Deep autoencoder [51]	2019	70.3
TS-LSTM + temporal-inception [52]	2019	69.0
HATNet [53]	2019	74.8
Correlational CNN+LSTM [54]	2020	66.2
STDAN [55]	2020	56.5
DB-LSTM+SSPF [56]	2021	75.1
DS-GRU [57]	2021	72.3
TCLC [58]	2021	71.5
Evidential deep learning [59]	2021	*77.0*
ViT+LSTM [23]	2021	73.7
AdaptFormer [24]	2022	55.6
Semi-supervised temporal gradient learning [60]	2022	75.9
SVT (Linear) [25]	2022	57.8
SVT (Fine-tune) [25]	2022	67.2
SVFormer-S [26]	2023	59.7
SVFormer-B [26]	2023	68.2
ViT-ReT (Proposed)	2023	**78.4**

- ViT-ReT: 1 Layer, 128-Dim Embedding, 512 Neurons, 2 Attention Heads

As should be expected, further increasing dataset complexity leads to a larger loss and lower accuracies for both models when given the same training configuration. However, the ReT and LSTM models are still performing very good with ResNet50 as a feature extractor. Even with 50 categories and over 6,500 total samples, the ResNet50-LSTM and ResNet50-ReT models are still able to maintain good accuracy. Results in Table 3 indicate that the accuracies of ViT-LSTM and ViT-ReT models are close to ResNet50-LSTM and ResNet50-ReT models. Specifically, ViT-LSTM model is within 3.3% accuracy of ResNet50-LSTM model and ViT-ReT model is within 1.9% accuracy of ResNet50-ReT model. Furthermore, ViT-ReT model is within 3.6% accuracy of ResNet50-LSTM model. Finally, the total dataset results are shown in Table 4. Optimal configuration:

- ResNet-LSTM: 2 Layer, 64 Units
- ResNet-ReT: 1 Layer, 128-Dim Embedding, 1024 Neurons, 4 Attention Heads
- ViT-LSTM: 2 Layers, 512 Units
- ViT-ReT: 1 Layer, 128-Dim Embedding, 512 Neurons, 2 Attention Heads

With the final increase in complexity, we continue to see the same trends as with the other datasets. The LSTM and ReT perform almost identically on all tasks. Results in Table 4 indicate that ViT-LSTM model is within 4.4% accuracy of ResNet50-LSTM model and ViT-ReT model is within 5.4% accuracy of ResNet50-ReT model. Furthermore, ViT-ReT model is within 5.4% accuracy of ResNet50-LSTM model. These results indicate that ViT and ReT models perform quite close to ResNet50 and LSTM model, and transformers can be used in place of typical CNN and LSTM configurations. While the vision transformer did not perform as well as ResNet50 for some datasets, its accuracy is still adequate for many tasks and shows promise with a better implementation.

Now that it has been show that transformers perform with comparable accuracy to current methods, the next section will examine their training and prediction speeds.

B. COMPARISON WITH STATE-OF-THE-ART METHODS ON BENCHMARK DATASETS

The proposed ViT-ReT framework is extensively evaluated on benchmark human action video datasets (including YouTube action [61], HMDB51 [62], UCF50 [63], and UCF101 [64] dataset) and the comparative analyses with the state-of-the-art human activity recognition methods are presented in Table 5, 6, 7 and 8, respectively, where the best accuracy score is reported in bold and runner-up in italics. Furthermore, the training history of the proposed ViT-ReT framework and other baseline methods on each dataset are shown in Fig. 19. From Fig. 19, it can be noticed that the proposed ViT-ReT and ViT-LSTM seem to struggle in the first half of training (in first 40 epochs); however, they perform better in later epochs by improving training accuracy.

The quantitative results obtained by the proposed ViT-ReT framework and state-of-the-art methods on YouTube action dataset are given in Table 5. From the results in Table 5, it can

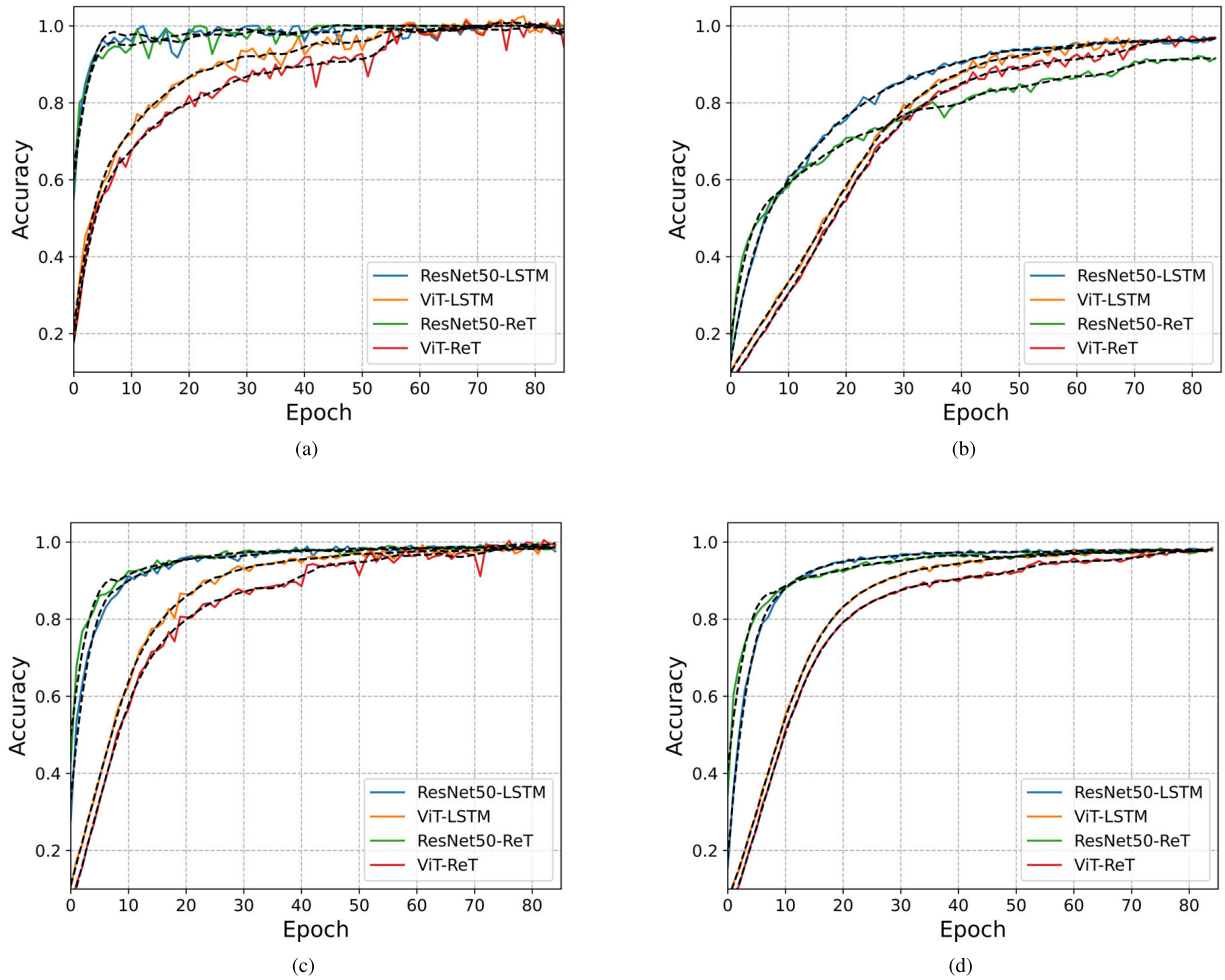


FIGURE 19. Training history of the proposed ViT-ReT along with other experimented baseline methods (including ResNet50+LSTM, ViT+LSTM, and ResNet50+ReT) on four benchmark human action video datasets: (a) YouTube Action dataset, (b) HMDB51 dataset, (c) UCF50 dataset, and (d) UCF101 dataset.

TABLE 7. Quantitative comparative analysis of the proposed ViT-ReT framework with the state-of-the-art action recognition methods on UCF50 dataset.

Method	Year	Accuracy (%)
Multi-task hierarchical clustering [41]	2016	93.2
Deep autoencoder [51]	2019	*96.4*
Ensemble model with swarm-based optimization [65]	2021	92.2
DS-GRU [57]	2021	95.2
Local-global feature+QSVM [45]	2021	69.4
ViT+LSTM [23]	2021	96.1
ViT-ReT (Proposed)	2023	**97.1**

be observed that the proposed ViT-ReT framework attains the best accuracy of 92.4% as compared to the other state-of-the-art action recognition methods among which the multi-task hierarchical clustering [41] method achieves a runner-up accuracy of 89.7%. Rest of the methods that include BT-LSTM [42], KFDI [43], Dilated CNN+BiLSTM+RB [44], Local-global features + QSVM [44], and 3DCNN [46] obtain accuracies of 85.3%, 79.4%, 89.0%, 82.6%, and 85.2%, respectively. For HMDB51 dataset, which is a very challenging human action video dataset, the proposed ViT-ReT dominates the state-of-the-art approaches by achieving the upmost accuracy as given in Table 6. Results in Table 6 indicate that the proposed ViT-ReT obtains an accuracy of 78.4% followed by the runner-up Evidential deep learning [59] having an accuracy of 77.0%. The multi-task hierarchical clustering method obtains the lowest accuracy of 51.7% followed by AdaptFormer [24] having the second-lowest accuracy of 55.6%. Thus, ViT-ReT framework provides an improvement in accuracy of 36.7% over the multi-task hierarchical clustering method for the HMDB51 dataset. The rest of comparative methods that include STPP+LSTM [47], Optical flow + multi-layer LSTM [48], TSN [49], IP-LSTM [50], Deep autoencoder [51], TS-LSTM + temporal-inception [52], HAT-Net [53], Correlational CNN+LSTM [54], STDAN [55], DB-LSTM+SSPF [56], DS-GRU [57], TCLC [58], Vit+LSTM [23], Semi-supervised temporal gradient learning [60], SVT (Linear) [25], SVT (Fine-tune) [25], SVFormer-S [26], and SVFormer-S [26] obtain accuracies of 70.5%, 72.2%, 70.7%, 58.6%, 70.3%, 69.0%, 74.8%, 66.2%,

TABLE 8. Quantitative comparative analysis of the proposed ViT-ReT with the state-of-the-art action recognition methods on UCF101 dataset.

Method	Year	Accuracy (%)
Multi-task hierarchical clustering [41]	2016	76.3
Saliency-aware 3DCNN with LSTM [66]	2016	84.0
Spatiotemporal multiplier networks [67]	2017	87.0
Long-term temporal convolutions [68]	2017	82.4
Bidirectional-LSTM [69]	2017	92.8
Videolstm [70]	2018	89.2
Two-stream convnets [71]	2018	84.9
TS-LSTM + temporal-inception [52]	2019	91.1
Hybrid deep learning [72]	2020	89.3
SVT (Linear) [25]	2022	90.8
SVT (Fine-tune) [25]	2022	*93.7*
ConvNet Transformer [27]	2023	86.1
SVFormer-S [26]	2023	79.1
SVFormer-B [26]	2023	86.7
ViT-ReT (Proposed)	2023	**94.7**

56.5%, 75.1%, 72.3%, 71.5%, 73.7%, 75.9%, 57.8%, 67.2%, 59.7%, and 68.2%, respectively.

The quantitative results assessment with the state-of-the-art methods on UCF50 dataset are listed in Table 7. From the listed comparative assessment, it can be noticed that the ViT-ReT improves the accuracy as compared to the best existing method, that is, Deep autoencoder [51] by achieving an accuracy of 97.1%, whereas the Deep autoencoder [51] has an accuracy of 96.5%. The local-global feature+QSVM [45] achieves the lowest accuracy of 69.4%, whereas the rest of the methods that include multi-task hierarchical clustering [41], ensemble model with swarm-based optimization [65], DS-GRU [57], and Vit+LSTM [23] obtain accuracies of 93.2%, 92.2%, 95.2%, and 96.1%, respectively. Finally, the performance comparison of activity recognition methods on UCF101 dataset is presented in Table 8. The results listed in Table 8 show that the ViT-ReT dominates the state-of-the-art methods by achieving the best accuracy of 94.7% followed by SVT (Fine-tune) [25] which attains the runner-up accuracy of 93.7%. Multi-task hierarchical clustering [41] exhibits the lowest accuracy of 76.3% followed by the Long-term temporal convolutions [68] with the second-lowest accuracy of 82.4%. Rest of the comparative methods that include saliency-aware 3DCNN with LSTM [41], Spatiotemporal multiplier networks [67], Bidiectional-LSTM [69], Videolstm [70], Two-stream convnets [71], TS-LSTM + temporal-inception [52], hybrid deep learning [72], SVT (Linear) [25], ConvNet Transformer [27], SVFormer-S [26], and SVFormer-B [26] attain accuracies of 84.0%, 87.0%, 92.8%, 89.2%, 84.9%, 91.1%, 89.3%, 90.8%, 86.1%, 79.1%, and 86.7, respectively. Overall, based on the conducted comparative analysis, the proposed ViT-ReT performs better than the state-of-the-art methods on each experimented dataset, which shows the effectiveness and robustness of the proposed ViT-ReT over existing mainstream human action/activity recognition methods.

TABLE 9. Timing results with the 4 actions database.

4 Actions	Timing Results	
	Training	*Throughput*
LSTM	48.85s	0.98s
ReT	37.70s	0.19s

TABLE 10. Timing results with the 20 actions database.

20 Actions	Timing Results	
	Training	*Throughput*
LSTM	2:37.00	0.51s
ReT	3:04.68	0.80s

C. CATEGORY-WISE ACCURACY ASSESSMENT ON DIFFERENT BENCHMARK DATASETS

This section presents the results obtained by assessing the proposed ViT-ReT approach using the category-wise accuracy metric (capturing the model performance for individual classes in datasets for the experimental settings).Further, we have also assessed ViT-ReT using category-wise accuracy metric for the test sets of each dataset, where the obtained results are shown in Fig. 20. From the visual results in Fig. 20, it can be noticed that ViT-ReT obtains around 92% accuracy for all classes of YouTube Action, UCF50, and UCF101 datasets. For the HMDB51 dataset, which is one of the challenging human actions video datasets, our method obtains around 80% accuracy for most of the classes. The obtained category-wise accuracy validates the generalization of ViT-ReT for recognizing human actions in videos. Thus, these extensive experiments on benchmark human action recognition datasets and the obtained results confirm the effectiveness and robustness of ViT-ReT for human action/activity recognition task.

D. CLASSIFICATION TIME

In this section, we show, for each UCF 101 subset, the timing data for the LSTM and ReT models, comparing their training time and throughput time for the entire data subset using our best model configuration. Of note, only timing data using ResNet50 will be shown. Since the model was implemented in a way that allows for direct timing of the LSTM or Transformer layers, it is not necessary to look at both feature extractors as the processing time of the Transformer and LSMT will be the same, or very close, for the different feature extractors. First, we will look at the timing data for the 4 Action subset. Timing data for this subset can be seen in Table 9.

With this simple dataset, containing only 478 total videos and 4 categories, the ReT is faster than the LSTM. The ReT is able to train 1.30× faster than the LSTM models, and it is able to process the entire dataset 4.79× faster. This shows that, for smaller data subsets, the ReT model is significantly

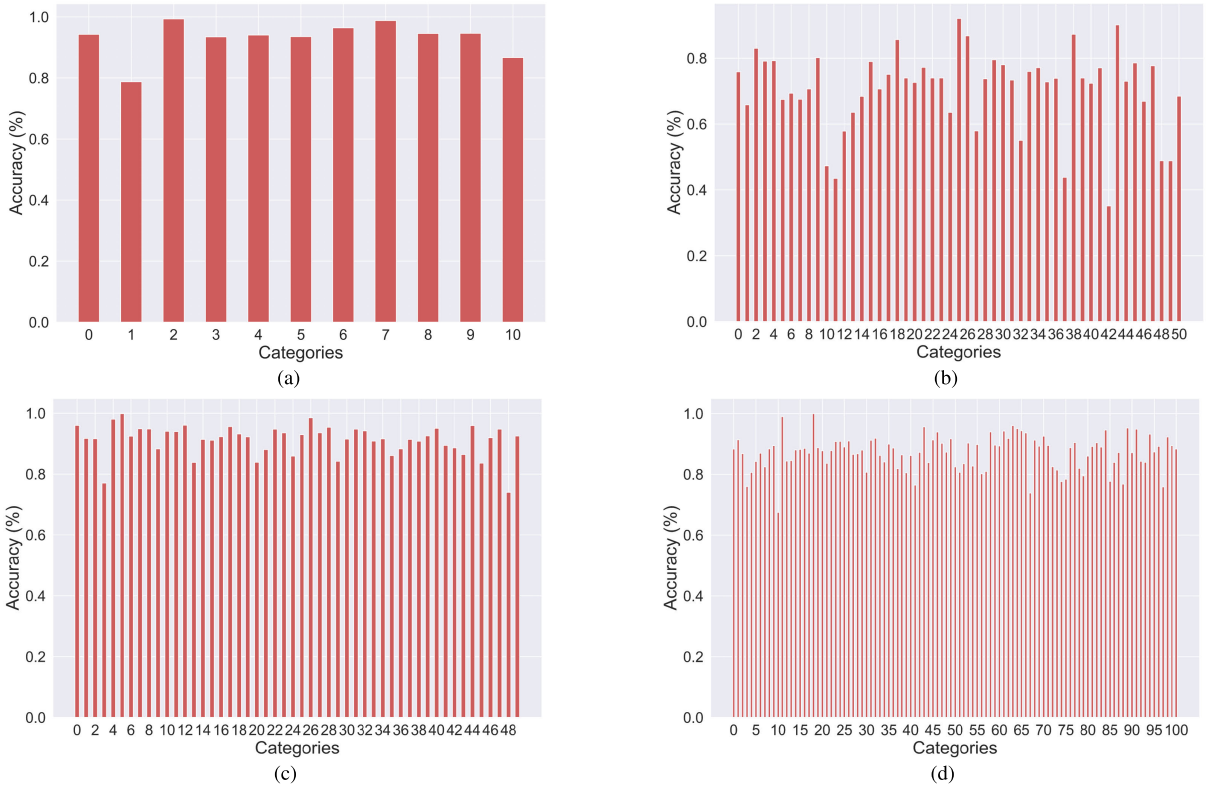


FIGURE 20. Category-wise accuracy performance of the proposed ViT-ReT framework on the test sets of four experimented datasets: (a) YouTube Action dataset, (b) HMDB51 dataset, (c) UCF50 dataset, and (d) UCF101 dataset.

TABLE 11. Timing results with the 50 actions database.

50 Actions	Timing Results	
	Training	*Throughput*
LSTM	6:57.90	0.80s
ReT	8:16.60	0.80s

TABLE 12. Timing results with the 101 actions database.

101 Actions	Timing Results	
	Training	*Throughput*
LSTM	21:01.75	1.72s
ReT	16:16.01	1.42s

faster than the LSTM model. For the next data subset, timing data can be seen in Table 10.

In the case of the slightly more complex data, 2500 videos with 20 action categories, the LSTM is marginally faster. The training of the ReT was 1.17× slower than the LSTM, and it processed the entire dataset 1.5× slower. Timing results for the 50-category dataset can be seen in Table 11.

Again, as the complexity increases, 6,567 videos with 50 categories, the training time for the ReT begins to lag behind the training time of the LSTM, training 1.15× slower. However, once trained it is able to process the entire dataset in essentially the same amount of time. While it may take more time to train the ReT models to the same level as the LSTM models, once it is trained it runs just as fast as the LSTM models for a medium complexity dataset. Finally, the total dataset timing data is shown in Table 12.

With the final and most complex dataset, 13,320 videos and 101 categories, the ReT performs much better than the LSTM model. It was able to train 1.29x faster, and able to process the entire dataset 1.21× faster. For the most complex dataset, the ReT is simply faster than using a RNN model. These results show the promise of the ReT for activity recognition tasks. The ReT performs just as accurately while being just as fast or faster than the LSTM model in most tasks. In the next section, a throughput analysis of the entire model structures is performed, comparing the ViT-ReT model to the traditional ResNet50-LSTM model.

E. TOTAL THROUGHPUT TIME

To measure model throughput, our best performing ReT and LSTM configurations for the total dataset are used. Once trained, the ViT is used for feature extraction with the ReT and ResNet50 is used for the feature extraction with the LSTM (ViT-ReT and ResNet50-LSTM). These models are then passed a randomly selected subset of the validation data to make predictions. The time for each prediction was tracked and used to calculate throughput. Each video file contains 20 frames. Time data is shown in Table 13.

As shown above, the ViT-ReT model is significantly faster than ResNet50-LSTM model. In most case, the ViT-ReT is 2× faster than the ResNet50-LSTM model. This speedup can

TABLE 13. Throughput timing results for different number of files.

Method	Files Predicted			
	1	*100*	*1000*	*10000*
ViT-ReT	1.32s	4.07s	35.77s	5:59.25
ResNet50-LSTM	2.69s	9.38s	1:10.43	11:52.03

TABLE 14. Throughput FPS results for different number of files.

Method	Files Predicted			
	1	*100*	*1000*	*10000*
ViT-ReT	15.15	491	559	557
ResNet50-LSTM	7.43	213	283	281
ViT-ReT Speedup	2×	2.24×	1.98×	1.98×

TABLE 15. Memory (MB) usage of ViT-ReT vs ResNet50-LSTM.

Method	Action Categories			
	4	*20*	*50*	*101*
ViT-ReT	1021.15	1019.28	1015.84	1021.66
ResNet50-LSTM	1121.42	1122.46	1117.06	1121.5
Memory Efficiency Gained	1.10×	1.10×	1.10×	1.10×

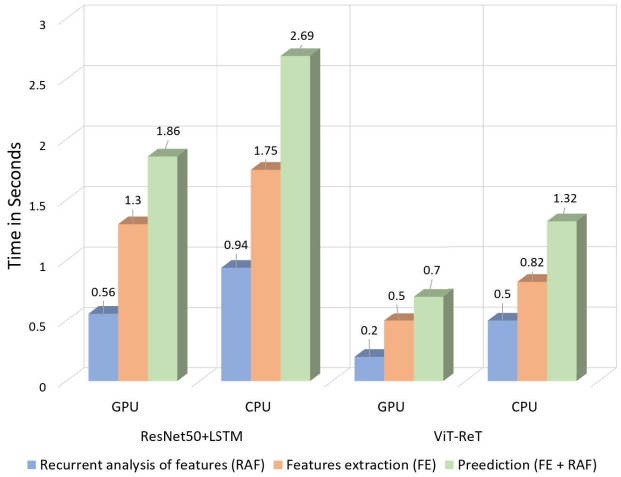


FIGURE 21. Run time analysis (in seconds) of our proposed ViT-ReT framework for features extraction, recurrent analysis of features and prediction (including both features extraction and recurrent analysis of features) on both GPU and CPU execution environments.

be seen in Table 14, along with the frames per second (FPS) for the two models with each different number of files.

These results clearly show that the ViT-ReT model is significantly faster, in all cases approximately 2× faster, than the ResNet50-LSTM model. This is a clear example of the promise transformers have in activity recognition. Regardless of the complexity of the dataset, the ViT-ReT is faster. It is important to note that, in the case of the entire model flow, the ResNet50-LSTM model is still more accurate than the ViT-ReT model, but with a better implementation of the ViT, this accuracy disparity would most likely be overcome. In the next section, the memory used in the creation of both models is compared.

F. MEMORY

Since it has been shown that the ReT is just as accurate while being faster than the LSTM, the next metric to consider is the memory usage of the two different models. To track this, the process's memory usage was tracked before and after creation of the models using the best model configuration for each data subset. The difference in the memory usage measured before and after each task shows how much memory performing a certain action with a specific model will take. The memory data can be seen in Table 15. As is shown, the ViT-ReT model is more memory efficient than the ResNet50-LSTM model while still being faster. On average, the transformer model is 1.1x more memory efficient than the ResNet50-LSTM model. Even using multiple different model configurations, the transformer models are more memory efficient. In the next section, our results from all testing sections will be discussed in detail.

G. RUN TIME COMPARISON WITH THE STATE-OF-THE-ART METHODS

To assess the feasibility of our proposed ViT-ReT for real-time applications (i.e., real-time human action detection and recognition in video streams), we have computed the runtime of the proposed ViT-ReT for the human action recognition task in terms of frames per second (FPS), seconds per frame (SPF), and in seconds for action/activity prediction. In this round of experiments, we have tested our proposed ViT-ReT and the baseline ResNet50-LSTM method on action video clips having FPS between 25 and 30, both on GPU and CPU platforms. The obtained results for the proposed ViT-ReT and the baseline ResNet50-LSTM method for run time in seconds, FPS, and SPS are depicted in Fig. 21, Fig. 22, and Fig. 23, respectively. For the better run time assessment of the proposed ViT-ReT and the baseline ResNet50-LSTM, we have estimated the run time of specific tasks (i.e., features extraction, recurrent analysis of features, and prediction (including both features extraction and recurrent analysis of features) as well as overall run time in seconds, FPS, and SPF as shown in Fig. 21, Fig. 22, and Fig. 23, respectively. From the depicted results, it can be noticed that the proposed ViT-ReT is 2× faster than the baseline ResNet50+LSTM method across all experiments on both GPU and CPU. This speedup can also be observed in task specific run times, which validate the computational efficiency of the proposed ViT-ReT over baseline ResNet50-LSTM method.

In addition to comparison with the baseline method (ResNet50+LSTM), we have also compared the obtained run time results with the state-of-the-art methods (from Section VII-B), where the obtained run time comparison results are presented in Table 16. The run time comparison results in Table 16 indicate that the proposed ViT-ReT prevails over the state-of-the-art methods by obtaining the best FPS and SPF scores on both GPU (FPS of 28 and SPF of 0.035) and CPU (FPS of 15 and SPF of 0.066) platforms, followed by the Optical flow + multilayer LSTM [48] method, which attains the second best FPS and SPF scores for both GPU (FPS of 25 and SPF of 0.040) and CPU (FPS of 5.4 and SPF of 0.180) platforms. The Videolstm [70]

TABLE 16. Run time analysis of our proposed ViT-ReT framework with the state-of-the-art human action recognition methods.

Method	Seconds per Frame (SPF)		Frames per second (FPS)	
	GPU	CPU	GPU	CPU
Bidirectional LSTM [69]	0.057	-	20	-
Optical flow + multi-layer LSTM [48]	*0.040*	*0.180*	*25*	*5.4*
TSN [49]	0.071	-	14	-
Videolstm [70]	0.094	-	10.6	-
IP-LSTM [50]	0.043	-	23.2	-
Deep autoencoder [51]	0.043	0.52	24	1.5
DS-GRU [57]	0.040	-	25	-
ViT-ReT (Proposed)	**0.035**	**0.066**	**28**	**15**

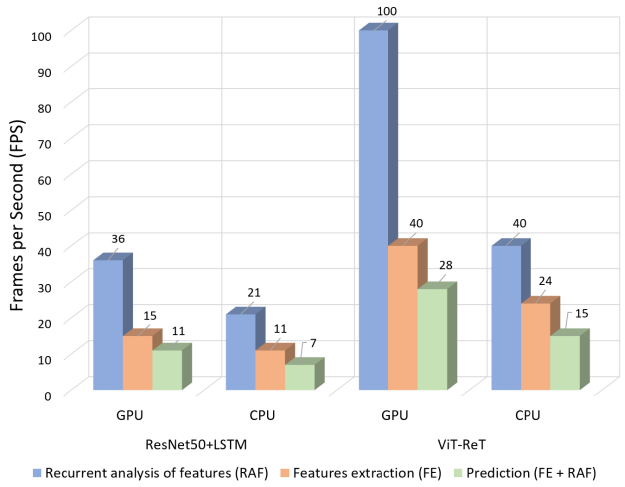


FIGURE 22. The estimated FPS values of our proposed ViT-ReT framework for features extraction, recurrent analysis of features, and prediction (including both features extraction and recurrent analysis of features) on both GPU and CPU execution environments.

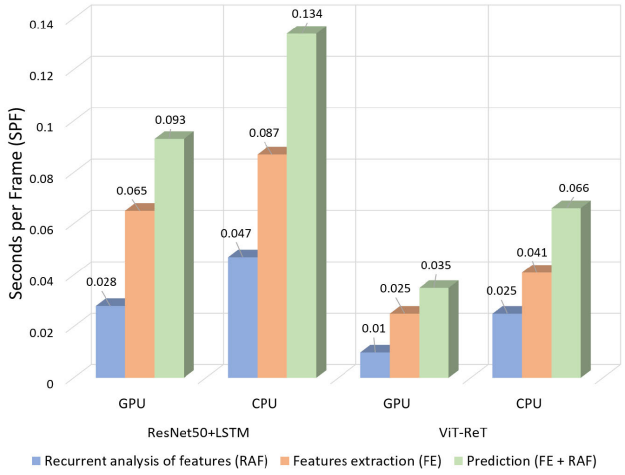


FIGURE 23. The estimated SPF values of our proposed ViT-ReT framework for features extraction, recurrent analysis of features, and prediction (including both features extraction and recurrent analysis of features) on both GPU and CPU execution environments.

method achieves the lowest FPS and SPF of 10.6 and 0.094, respectively, on the GPU platform followed by TSN [49] method with the second lowest FPs and SPF of 14 and 0.071, respectively, on the GPU platform. Results in Table 16 indicate that the ViT-ReT provides an improvement in run time of 38.2% and 36.4%, on average, in terms of FPS and SPF, respectively, on the GPU platform as compared to the state-of-the-art human action recognition methods.

Moreover, it is also worth mentioning here that overall storage requirements of our proposed ViT-ReT is only 28.5 MB (ViT 25 MB and ReT 3.5 MB) and is on average 1.1× more memory efficient than other baseline methods (Section VII-F). The trained model with these storage requirements can be easily run on devices with limited computational resources such as Raspberry pi, arduino, and other embedded devices. The obtained run time (FPS and SPF) and model storage requirements validate the effectiveness and suitability of the proposed ViT-ReT for human action/activity recognition in resource-constrained and real-time environments.

H. DISCUSSION

The most striking result is the ability of the ReT model to maintain similar accuracy with the LSTM model while being faster and more memory efficient. In all cases, the ReT model exhibits comparable cross entropy loss and accuracy to the LSTM model. This makes TNN models ideal not only for general use, but also for use in edge devices with limited resources. The transformer, even if not implemented in parallel, can use resources more efficiently by using less memory and by using that memory for less time than a traditional RNN activity recognition model without any significant drop in performance. Clearly, an implementation that utilizes any amount of parallelization to perform more quick matrix multiplication (the backbone of multi-headed attention) would allow for even better use of resources. Overall, the transformer has shown itself to be a better and more versatile model than the traditional RNNs when used for activity recognition.

For the ViT model, it maintains decent accuracy, but was significantly behind its deep CNN counterpart in accuracy. This is most likely due to the specific implementation of the ViT used and the amount of training performed for it. As shown by Dosovitskiy et al. [36], it can perform comparable to other state-of-the-art deep CNN models, and even more efficiently in some cases. With a better implementation it is still possible to achieve accuracies close to or better than state-of-the-art deep CNNs for activity recognition. However, the ViT, when used in conjunction with the ReT, creates a model that is faster and more memory efficient than a traditional model using ResNet50 and LSTM

classifiers. This serves to show that, with a better trained implementation, there is a potential for the ViT to make significant improvements over current deep CNNs.

VIII. CONCLUSION

Activity recognition is a challenging research area in computer vision that involves recognizing the actions taken by humans with different sensors. Contemporary activity recognition models rely on multiple very dense and complex networks which make these models unsuitable for resource-constrained edge devices. Deep residual CNNs and RNN chains, the backbone of modern activity recognition models, suffer from complexity and speed issues, making real-time applications or computations in limited resource environments difficult. There is a need to advance activity recognition models beyond what currently exists, and the transformer neural networks (TNNs) provide a propitious alternative. The TNNs are one of the most promising neural networks created in the recent years. TNNs have revolutionized sequence-to-sequence modeling and shown that it is possible to create very accurate and lightweight models with little more than matrix multiplications and fully connected artificial neural networks. This paper sought to show the potential applications of TNNs beyond sequence-to-sequence models. It was also shown that there is potential for Transformers to create significant improvements in computer vision and activity recognition by replacing both sets of complex networks that models currently rely on. The recurrent transformer (ReT), and its extension the vision transformer (ViT), show promise in these domains, with the potential to significantly improve on current state-of-the-art models. Their application into nonsequential and non-time dependent datasets could prove to make huge improvements in a wide variety of machine learning tasks.

Our results have shown that the ReT can make predictions that maintain similar accuracy to traditional RNN classifiers used for activity recognition while being faster and more memory efficient. When used in conjunction with a ViT for feature extraction, there is a significant speedup over current deep CNN and RNN activity recognition models, as well as a notable improvement in memory efficiency. For instance, the proposed ViT-ReT framework achieves a speedup of 2× over baseline CNN-RNN methods (e.g., ResNet50-LSTM) while obtaining the same level of accuracy. Moreover, the proposed ViT-ReT framework outperforms the state-of-the-art methods on four publicly available human action datasets (that include YouTube action, UCF50, UCF101, and HMDB51 datasets) in terms of both model precision and runtime. These significant improvements in model accuracy (up to 52.64%) and inference time (up to 38.2% on average) offered by our proposed ViT-ReT validate its effectiveness and efficiency over existing mainstream human action recognition methods.

IX. FUTURE RESEARCH DIRECTIONS

There are three immediate directions for future research that extends this paper. The first is creating a more accurate Vision Transformer and completing Vision Transformer Focused testing to show its application to the activity recognition Field. This would show without a doubt the utility of the Transformer in all fields, not only sequence-to-sequence problem fields. It can also serve to significantly improve the model efficiency of activity recognition models. With a better vision transformer implementation, testing should focus on speed and memory efficiency while maintaining accuracy, as is done in this paper. The tests discussed in this paper can then be extended to including training and testing utilizing the entire activity recognition model end to end including all preprocessing and feature extraction to show the overall improvements made to the activity recognition task by both the recurrent and vision transformer models.

Second, creating fully parallel implementation of both the recurrent and vision transformer networks would show how powerful TNNs can be when they are able to fully utilize the maximum potential of their architecture. TNNs are already faster and more memory efficient than the RNN networks currently in use without a parallel implementation that makes use of fast matrix multiplication to drastically increase model speed. With better parallel implementations, TNN activity recognition models can be used in many different resource-constrained environments and/or real-time systems.

Finally, we believe it may be possible to create a TNN for activity recognition that performs both feature extraction and classification within a single model, removing the need for a deep CNN or ViT altogether. This would be accomplished by using the embedding layers to perform a pseudo-feature extraction before data processing begins in the ReT. With the right embedding and preprocessing steps, it may be possible to achieve similar accuracy without using CNNs, instead using what may be a significantly more lightweight and efficient overall model implementation. This could potentially lead to creating the fastest and most memory-efficient activity recognition model in existence.

ACKNOWLEDGMENT

The authors would like to thank Dr. Erik Blasch for his guidance and support on this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force, Air Force Research Laboratory (AFRL), and/or AFOSR.

REFERENCES

[1] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3642–3649. [Online]. Available: http://openmp.org/wp/

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[5] A. J. Storkey, "When training and test sets are different: Characterizing learning transfer," in *Dataset Shift in Machine Learning*. Cambridge, MA, USA: MIT Press, 2008, pp. 2–28.

[6] F. Timme, J. Kerdels, and G. Peters, "On the robustness of convolutional neural networks regarding transformed input images," in *Proc. 12th Int. Joint Conf. Comput. Intell.*, 2020, pp. 396–403.

[7] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of GPU-based convolutional neural networks," in *Proc. 18th Euromicro Conf. Parallel, Distrib. Netw.-based Process.*, Feb. 2010, pp. 317–324.

[8] A. Munir, P. Kansakar, and S. U. Khan, "IFCIoT: Integrated fog cloud IoT: A novel architectural paradigm for the future Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 6, no. 3, pp. 74–82, Jul. 2017.

[9] A. Munir, E. Blasch, J. Kwon, J. Kong, and A. Aved, "Artificial intelligence and data fusion at the edge," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 36, no. 7, pp. 62–78, Jul. 2021.

[10] K. Muhammad, H. Ullah, M. S. Obaidat, A. Ullah, A. Munir, M. Sajjad, and V. H. C. de Albuquerque, "AI-driven salient soccer events recognition framework for next-generation IoT-enabled environments," *IEEE Internet Things J.*, vol. 10, no. 3, pp. 2202–2214, Feb. 2023.

[11] J. Qin, L. Liu, Z. Zhang, Y. Wang, and L. Shao, "Compressive sequential learning for action similarity labeling," *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 756–769, Feb. 2016.

[12] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[13] S. S. Norouzi, A. Akbari, and B. Nasersharif, "Language modeling using part-of-speech and long short-term memory networks," in *Proc. 9th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2019, pp. 182–187.

[14] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proc. 8th Workshop Syntax, Semantics Struct. Stat. Transl. (SSST)*, 2014, pp. 103–111.

[15] J. C. Heck and F. M. Salem, "Simplified minimal gated unit variations for recurrent neural networks," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 1593–1596.

[16] A. Viswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. 31st Conf. Neural Inf. Process. Systmes (NIPS)*, Long Beach, CA, USA, 2017, pp. 1–11.

[17] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, Apr. 2017, pp. 351–360.

[18] J. Sun, Y. Fu, S. Li, J. He, C. Xu, and L. Tan, "Sequential human activity recognition based on deep convolutional network and extreme learning machine using wearable sensors," *J. Sensors*, vol. 2018, pp. 1–10, Sep. 2018.

[19] D. Roggen, A. Calatroni, L.-V. Nguyen-Dinh, R. Chavarriaga, and H. Sagha, "Opportunity activity recognition," UCI Mach. Learn. Repository, 2012, doi: 10.24432/C5M027.

[20] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proc. CVPR*, Jun. 2011, pp. 1297–1304.

[21] P. Russo, S. Ticca, E. Alati, and F. Pirri, "Learning to see through a few pixels: Multi streams network for extreme low-resolution action recognition," *IEEE Access*, vol. 9, pp. 12019–12026, 2021.

[22] H. Ma, W. Li, X. Zhang, S. Gao, and S. Lu, "AttnSense: Multi-level attention mechanism for multimodal human activity recognition," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 3109–3115.

[23] A. Hussain, T. Hussain, W. Ullah, and S. W. Baik, "Vision transformer and deep sequence learning for human activity recognition in surveillance videos," *Comput. Intell. Neurosci.*, vol. 2022, pp. 1–10, Apr. 2022.

[24] S. Chen, C. Ge, Z. Tong, J. Wang, Y. Song, J. Wang, and P. Luo, "AdaptFormer: Adapting vision transformers for scalable visual recognition," 2022, *arXiv:2205.13535*.

[25] K. Ranasinghe, M. Naseer, S. Khan, F. S. Khan, and M. S. Ryoo, "Self-supervised video transformer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 2864–2874.

[26] Z. Xing, Q. Dai, H. Hu, J. Chen, Z. Wu, and Y.-G. Jiang, "SVFormer: Semi-supervised video transformer for action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Apr. 2023, p. 18816–18826.

[27] N. H. Phong and B. Ribeiro, "Video action recognition collaborative learning with dynamics via PSO-ConvNet transformer," 2023, *arXiv:2302.09187*.

[28] J. Wensel, "Transformer neural networks for human activity recognition," Ph.D. dissertation, Dept. Comput. Sci., Kansas State Univ., Manhattan, KS, USA, 2022.

[29] K. Patel and P. Bhattacharyya, "Towards lower bounds on number of dimensions for word embeddings," in *Proc. 8th Int. Joint Conf. Natural Lang. Process.*, Nov. 2017, pp. 31–36.

[30] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[31] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.

[32] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2015, pp. 1–15.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.

[34] Hedu—Math of Intelligence. (Dec. 2020). *Visual Guide to Transformer Neural Networks (Episode 2) Multi-Head & Self-Attention*. YouTube. [Online]. Available: https://www.youtube.com/watch?v=mMa2PmYJlCo

[35] B. Debnath, M. O'Brient, S. Kumar, and A. Behera, "Attention-driven body pose encoding for human activity recognition," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 5897–5904.

[36] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenbor, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image worth 16 × 16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–22.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[38] F. Morales. (2021). *ViT-Keras*. [Online]. Available: https://github.com/faustomorales/vit-keras

[39] A. P. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your ViT? Data, augmentation, and regularization in vision transformers," *Trans. Mach. Learn. Res.*, May 2022.

[40] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human action classes from videos in the wild," Univ. Central Florida, Orlando, FL, USA, Tech. Rep., 2012.

[41] A.-A. Liu, Y.-T. Su, W.-Z. Nie, and M. Kankanhalli, "Hierarchical clustering multi-task learning for joint human action grouping and recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 1, pp. 102–114, Jan. 2017.

[42] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9378–9387.

[43] M. Riahi, M. Eslami, S. H. Safavi, and F. T. Azar, "Human activity recognition using improved dynamic image," *IET Image Process.*, vol. 14, no. 13, pp. 3223–3231, Nov. 2020.

[44] K. Muhammad, Mustaqeem, A. Ullah, A. S. Imran, M. Sajjad, M. S. Kiran, G. Sannino, and V. H. C. de Albuquerque, "Human action recognition using attention based LSTM network with dilated CNN features," *Future Gener. Comput. Syst.*, vol. 125, pp. 820–830, Dec. 2021.

[45] S. Al-Obaidi, H. Al-Khafaji, and C. Abhayaratne, "Making sense of neuromorphic event data for human action recognition," *IEEE Access*, vol. 9, pp. 82686–82700, 2021.

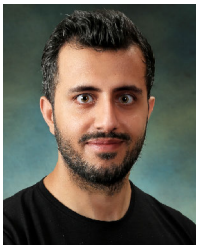
[46] R. Vrskova, R. Hudec, P. Kamencay, and P. Sykora, "Human activity classification using the 3DCNN architecture," *Appl. Sci.*, vol. 12, no. 2, p. 931, Jan. 2022.

[47] X. Wang, L. Gao, P. Wang, X. Sun, and X. Liu, "Two-stream 3-D Convnet fusion for action recognition in videos with arbitrary size and length," *IEEE Trans. Multimedia*, vol. 20, no. 3, pp. 634–644, Mar. 2018.

[48] A. Ullah, K. Muhammad, J. Del Ser, S. W. Baik, and V. H. C. de Albuquerque, "Activity recognition using temporal optical flow convolutional features and multilayer LSTM," *IEEE Trans. Ind. Electron.*, vol. 66, no. 12, pp. 9692–9702, Dec. 2019.

[49] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks for action recognition in videos," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 11, pp. 2740–2755, Nov. 2019.
[50] S. Yu, L. Xie, L. Liu, and D. Xia, "Learning long-term temporal features with deep neural networks for human action recognition," *IEEE Access*, vol. 8, pp. 1840–1850, 2020.
[51] A. Ullah, K. Muhammad, I. U. Haq, and S. W. Baik, "Action recognition using optimized deep autoencoder and CNN for surveillance data streams of non-stationary environments," *Future Gener. Comput. Syst.*, vol. 96, pp. 386–397, Jul. 2019.
[52] C.-Y. Ma, M.-H. Chen, Z. Kira, and G. AlRegib, "TS-LSTM and temporal-inception: Exploiting spatiotemporal dynamics for activity recognition," *Signal Process., Image Commun.*, vol. 71, pp. 76–87, Feb. 2019.
[53] A. Diba, M. Fayyaz, V. Sharma, M. Paluri, J. Gall, R. Stiefelhagen, and L. Van Gool, "Holistic large scale holistic video understanding," 2019, *arXiv:1904.11451*.
[54] M. Majd and R. Safabakhsh, "Correlational convolutional LSTM for human action recognition," *Neurocomputing*, vol. 396, pp. 224–229, Jul. 2020.
[55] Z. Zhang, Z. Lv, C. Gan, and Q. Zhu, "Human action recognition using convolutional LSTM and fully-connected LSTM with different attentions," *Neurocomputing*, vol. 410, pp. 304–316, Oct. 2020.
[56] J.-Y. He, X. Wu, Z.-Q. Cheng, Z. Yuan, and Y.-G. Jiang, "DB-LSTM: Densely-connected bi-directional LSTM for human action recognition," *Neurocomputing*, vol. 444, pp. 319–331, Jul. 2021.
[57] A. Ullah, K. Muhammad, W. Ding, V. Palade, I. U. Haq, and S. W. Baik, "Efficient activity recognition using lightweight CNN and DS-GRU network for surveillance applications," *Appl. Soft Comput.*, vol. 103, May 2021, Art. no. 107102.
[58] L. Zhu, H. Fan, Y. Luo, M. Xu, and Y. Yang, "Temporal cross-layer correlation mining for action recognition," *IEEE Trans. Multimedia*, vol. 24, pp. 668–676, 2022.
[59] W. Bao, Q. Yu, and Y. Kong, "Evidential deep learning for open set action recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 13329–13338.
[60] J. Xiao, L. Jing, L. Zhang, J. He, Q. She, Z. Zhou, A. Yuille, and Y. Li, "Learning from temporal gradient for semi-supervised action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 3242–3252.
[61] J. Liu, J. Luo, and M. Shah, "Recognizing realistic actions from videos 'in the wild,'" in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 1996–2003.
[62] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2556–2563.
[63] K. K. Reddy and M. Shah, "Recognizing 50 human action categories of web videos," *Mach. Vis. Appl.*, vol. 24, no. 5, pp. 971–981, Jul. 2013.
[64] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," 2012, *arXiv:1212.0402*.
[65] L. Zhang, C. P. Lim, and Y. Yu, "Intelligent human action recognition using an ensemble model of evolving deep networks with swarm-based optimization," *Knowl.-Based Syst.*, vol. 220, May 2021, Art. no. 106918.
[66] X. Wang, L. Gao, J. Song, and H. Shen, "Beyond frame-level CNN: Saliency-aware 3-D CNN with LSTM for video action recognition," *IEEE Signal Process. Lett.*, vol. 24, no. 4, pp. 510–514, Apr. 2017.
[67] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Spatiotemporal multiplier networks for video action recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7445–7454.
[68] G. Varol, I. Laptev, and C. Schmid, "Long-term temporal convolutions for action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1510–1517, Jun. 2018.
[69] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action recognition in video sequences using deep bi-directional LSTM with CNN features," *IEEE Access*, vol. 6, pp. 1155–1166, 2018.
[70] Z. Li, K. Gavrilyuk, E. Gavves, M. Jain, and C. G. M. Snoek, "VideoLSTM convolves, attends and flows for action recognition," *Comput. Vis. Image Understand.*, vol. 166, pp. 41–50, Jan. 2018.
[71] Y. Han, P. Zhang, T. Zhuo, W. Huang, and Y. Zhang, "Going deeper with two-stream ConvNets for action recognition in video surveillance," *Pattern Recognit. Lett.*, vol. 107, pp. 83–90, May 2018.
[72] N. Jaouedi, N. Boujnah, and M. S. Bouhlel, "A new hybrid deep learning model for human action recognition," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 32, no. 4, pp. 447–453, May 2020.

JAMES WENSEL received the B.S. degree in computer engineering and the M.S. degree in computer science from Kansas State University, in 2020 and 2022, respectively, where he is currently pursuing the Ph.D. degree in computer science. His master's research was focused on the use of transformer neural networks in human activity recognition. His research interests include computer vision, activity recognition, transformer neural networks, and reinforcement learning.



HAYAT ULLAH received the bachelor's degree in computer science from Islamia College Peshawar, Pakistan, in 2018, and the master's degree in computer science from Sejong University, Seoul, Republic of Korea, in 2021. He is currently pursuing the Ph.D. degree in computer science with Kansas State University, Mahattan, KS, USA. He is a Research Assistant with the Intelligent Systems, Computer Architecture, Analytics, and Security (ISCAAS) Laboratory, Kansas State University, exclusively involved in multi-model human actions modeling and activity recognition. He has published several articles in well-reputed journals, that include IEEE Internet of Things Journal and IEEE Transactions on Image Processing. His research interests include image processing, object detection, image segmentation, video analytics, deep learning applications in surveillance, and image enhancement.

ARSLAN MUNIR (Senior Member, IEEE) received the M.A.Sc. degree in electrical and computer engineering from The University of British Columbia (UBC), Vancouver, BC, Canada, in 2007, and the Ph.D. degree in electrical and computer engineering from the University of Florida (UF), Gainesville, FL, USA, in 2012.

From 2007 to 2008, he was a Software Development Engineer with the Embedded Systems Division, Mentor Graphics Corporation. From May 2012 to June 2014, he was a Postdoctoral Research Associate with the Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. He is currently an Associate Professor with the Department of Computer Science, Kansas State University. His current research interests include embedded and cyber-physical systems, secure and trustworthy systems, parallel computing, artificial intelligence, and computer vision. He received many academic awards, including the Doctoral Fellowship from the Natural Sciences and Engineering Research Council (NSERC) of Canada. He also received gold medals for best performance in electrical engineering and the gold medals and academic roll of honor for securing rank one in pre-engineering provincial examinations (out of approximately 300,000 candidates).