Kent State University

MIS-64061: Advanced machine learning

Fall 2023

Different Deep Learning Approaches for Human Activity Recognition

Submitted by:

Yanxi Li, yli130@kent.edu

Table of Contents

Table of figures

Abstract

The identification of human activities has become a significant focus in recent years due to its crucial role in various areas like surveillance, health, and security. Numerous deep learning models based on computer vision have been suggested to categorize human activities. This project involves a review of five approaches to human activity recognition and the implementation of four of them, incorporating the latest and most advanced methods in this field. The dataset used is UCF101[1], containing 101 categories of human activity videos. Due to computational constraints, I focused on coding implementations for seven selected categories. Among the four models tested, Long-term Recurrent Convolutional Networks (LRCN) exhibited the highest performance at 96.8%.

Introduction

Human activity recognition (HAR) has become an interesting research field in recent years for its various applications in many areas, such as healthcare, surveillance, human-human interaction, and human-computer interface.

Some studies[2,3] have classified human activities into different categories, and these studies has classified different human activities into two categories: simple activities and complicated activities. Simple activities include walking, jogging, standing, which only consider the body movement during the activity. While the complicated activities have a particular function added on the simple activities, for example, "Watching TV" includes "sitting" or "lying" on the sofa with "watching". Figure 1[4] is attached here to show the human activities classification in the thesis based on the paper[3].

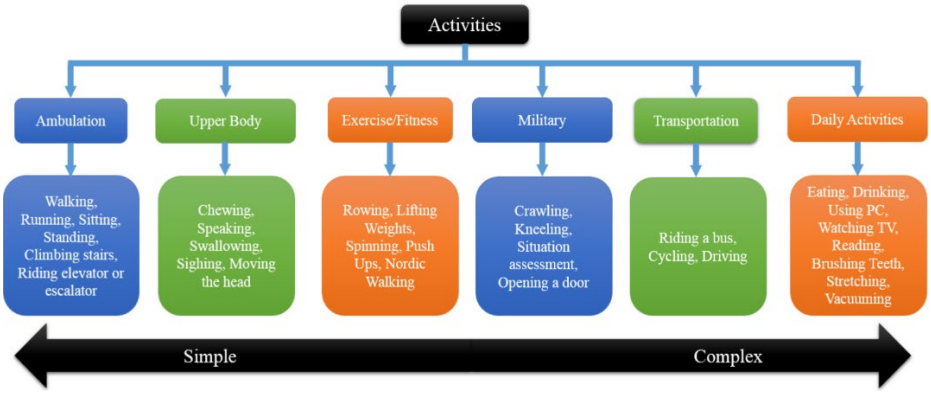


Figure 1. Classification of Human Activities by Lara

In macroscope, they are mainly two different types of methods used in human activity recognition, which are 1) Vision-based; 2) Sensor-based as shown in Figure 1[5]. Vision based is using deep learning models to classify the human activity videos captured by camera, which is what we discussed in this project. As for the sensor-based approach, it uses different kinds of

sensors to capture the human daily lives, and it can be further subdivided into three parts: wearable, object tagged, and dense sensing. In this project, I will use the first vision-based approach to do the HAR since our course is for deep learning.

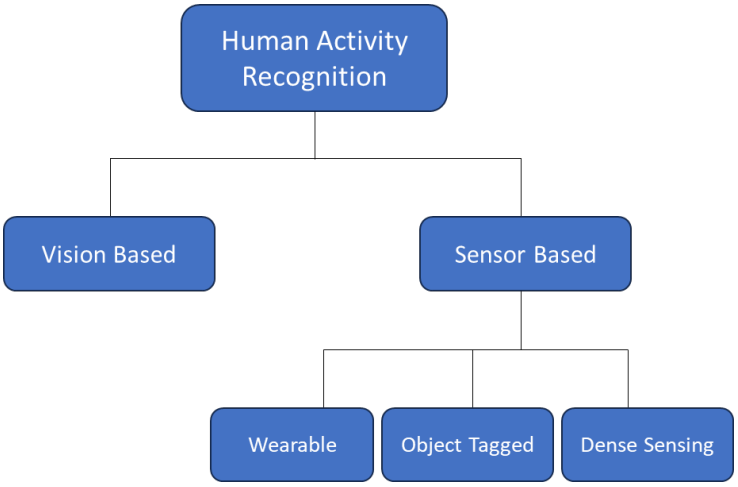


Figure 2. Classification for Human Activity Recognition methods

There are three different types of sequence model: many-to-one, one-to-many, and many-to-many and their structure is in Figure 3[6]. Video classification like the Human Activity Recognition is many-to-one since the video is taken as the several sequential images. Image description is a good example of the one-to-many model. Many-to-many, which is also named sequence-to-sequence (seq2seq) model, translation, video description, paragraph summary is the example for this category.

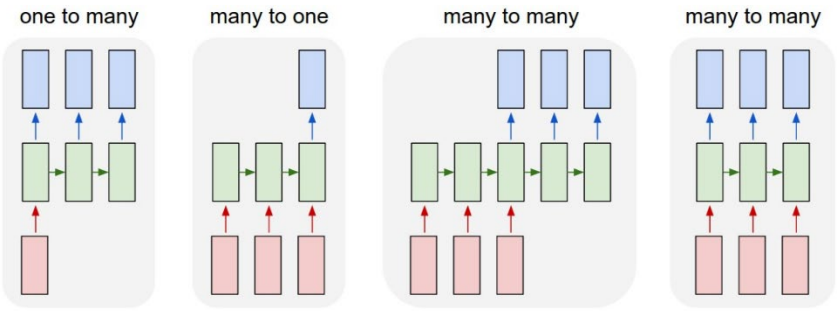


Figure 3. Different types of sequence model

In this project, I crafted four distinct deep learning models based on vision to accomplish Human Activity Recognition. The four models include 3D-CNN (convolutional neural network), Long-term Recurrent Convolutional Networks (LRCN), ConvLstm2D, and a combination of CNN with Transformer.

The remaining sections of this report are outlined as follows: Chapter 2 details the methodology for each approach, Chapter 3 focuses on the UCF101 dataset, Chapter 4 delves into the model-building process, Chapter 5 presents the results and discussion, and the final segment, Chapter 6, encapsulates the conclusion for this project.

Deep Learning in Polymer field

I will summarize the paper I went through used deep learning in polymer field as well since my background is polymer.

Vandans and the coworkers[7] used deep learning to identify the knot types. They used the simulation to generate millions of different polymer chains, the knot type in these conformations is 0, 3_1, 4_1, 5_1, and 5_2 (Figure 4[7]). In deep learning, we can take the five knot types as five categories. The study shows that the sequential model (LSTM) performs better than the fully connected model and the confusion matrix is attached in Figure 5[7].

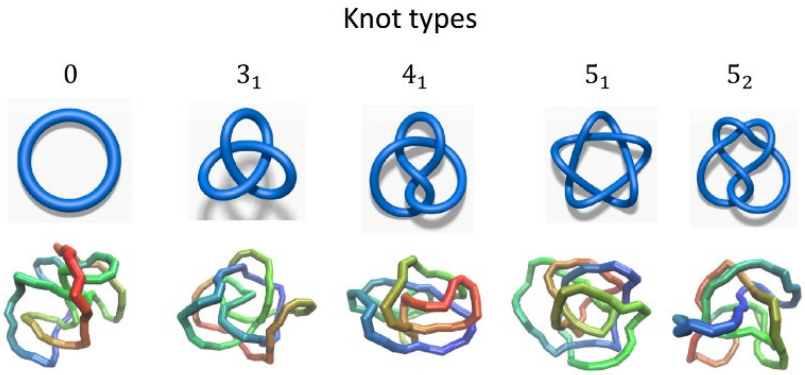


Figure 4. Five different knot types

LSTM are suitable for the sequence and time-series data. It motivates the researcher to utilize this architecture since knotting is a sequential property of the chain.

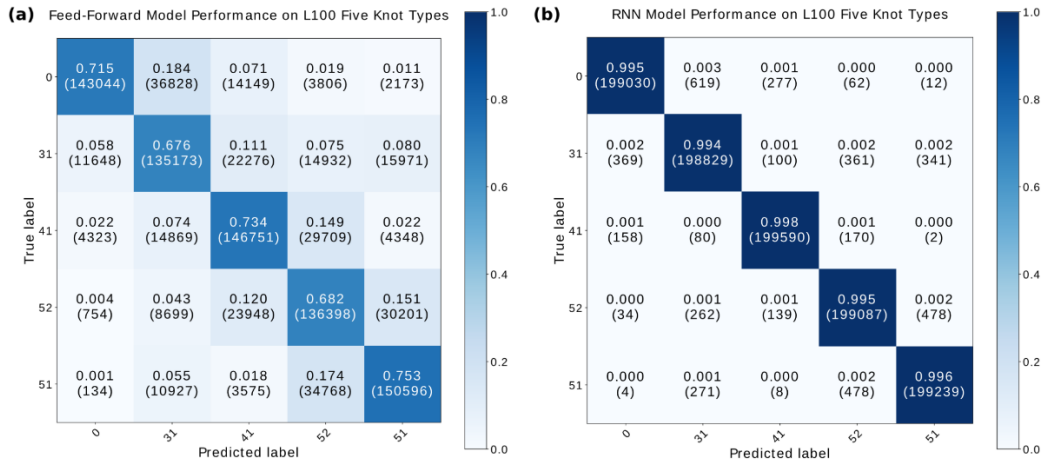


Figure 5. Confusion matrix for the two models

Wei[8] published a deep learning method which could not only identify the polymer chain state (Figure 6) but also the transition point. The simple fully connected model is used here, so the advantage for this method is without any order parameter and the calculation of heat capacity.

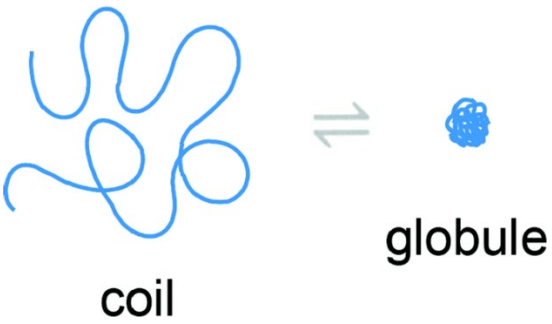


Figure 6. Polymer chain state in the solution

In simulation, polymer conformation is represented by 3D spatial coordinates for the n connected monomers. Researchers need to decide how to feed this information into the neural network. Here, instead of adopt image recognition idea to deal with the input data, they directly feed the 3D coordinates into the model.

Methodology

Flow diagram

During the class we learned how to deal with the image dataset, but what about the video dataset? We can consider each video file as a sequence of images! So, the first step for this project is to extract several series frames in the video. After getting the sequence of images in each video, we need to preprocess the image into pixels. The whole flow of the process is shown in Figure 3.

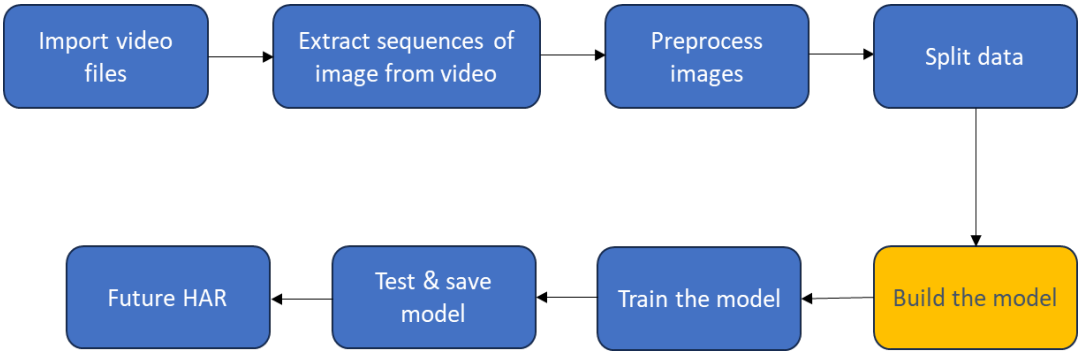


Figure 7. Flow diagram of the whole process

Common Deep Learning Single Architecture

Convolutional Neural Network (CNN)

CNN is mainly used in the field of image processing based on convolutional layers. The convolutional layers can do feature extraction for images. The polling layer added after the convolutional layer can reduce the size of the feature map and save the computational cost. Final layer is the fully connected layer which can help learn the non-linear combinations of the high-level features represented by the output from the previous layer. The simplified CNN structure flow is shown in Figure 4[9].

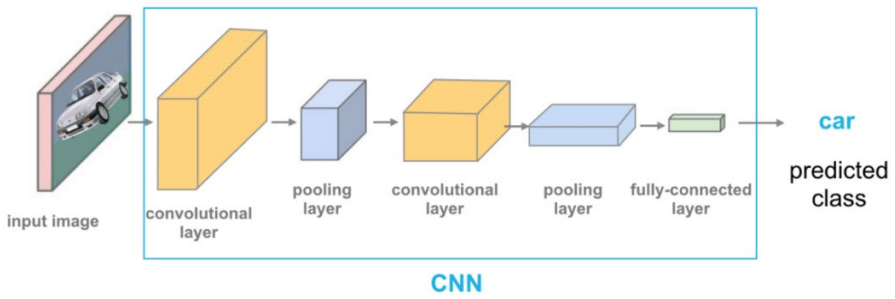


Figure 8. CNN simplified flow diagram.

The pretrained model in CNN is a common method for small dataset because the training output on the large dataset can be directly used in the related projects. If the pretrained dataset is humongous general, the features learned in the pretrained model can be effectively used in other image processing works. In the class, we learned there are two ways to use the pretrained model: feature extraction and fine-tuning. Utilizing the pretrained model ensembles standing on the shoulders of giants to solve the problems. There are some pretrained models over the years (shown in Figure 5[10]), VGG16 is what we learn and use in the class. In this HAR project, I will use different pretrained models to do the test.

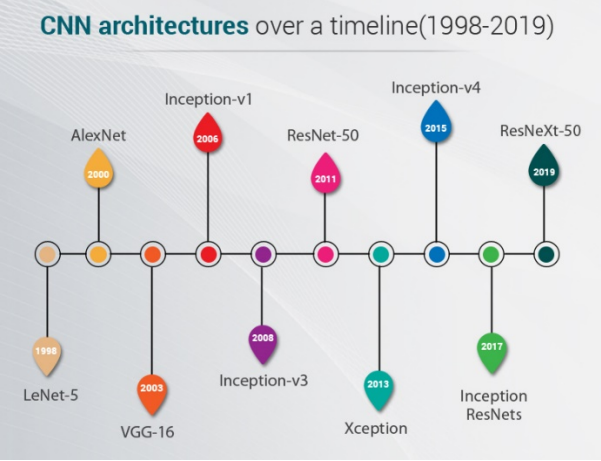


Figure 9. CNN pretrained network over the timeline

Long Short-Term Memory (LSTM)

Recurrent neural networks (RNN) need be to first introduce here. RNN is a deep learning neural network used to process sequence data. Unlike the fully connected neural network, all the inputs are independent of each other, for the inputs in RNN, they are related.

Long short-term memory (LSTM), a specific type of RNN, which is first published in 1997[11] and is designed to solve the vanishing gradient problem during the long sequence training using several gates (shown in Figure 6[12]). In other words, LSTM can have a better performance in longer sequence than traditional RNN.

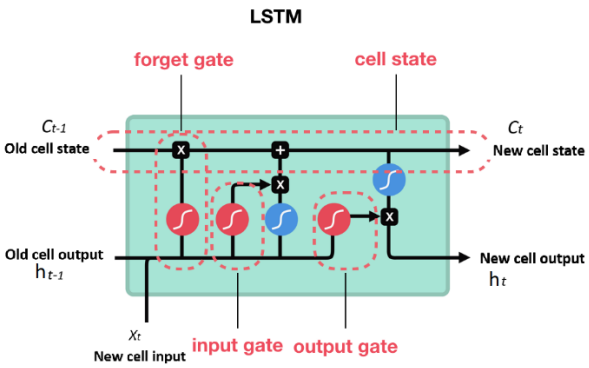


Figure 10. Structure for long short-term memory

Transformer

Transformer was first proposed in the paper 'Attention is all you need'[13] in 2017. The model was originally designed to improve the efficiency of machine translation because of the self-attention mechanism and positional encoding. For RNN and LSTM, the input sequence data is imported into the model serially, which means the time t data cannot be processed until the t-1 is completed. Using the transformer, the data can be processed parallelly. The structure of the Transformer is shown in Figure 8, it consists of encoder and decoder parts.

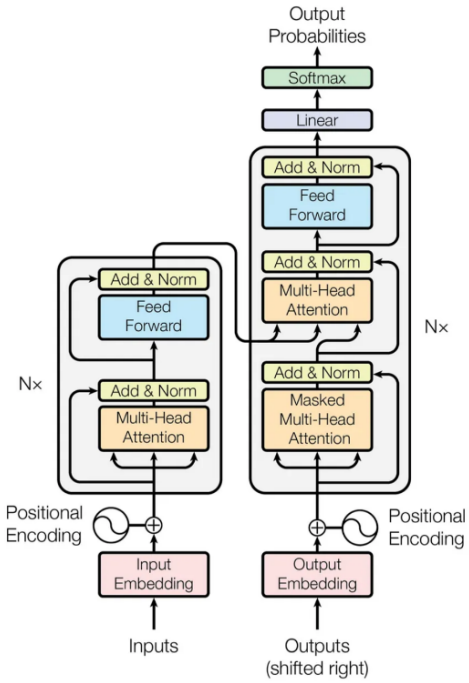


Figure 11. Structure for Transformer

Combined/Modified Architecture Used in Project

In this project, our aim is to do the video classification, since the video is be considered as series of images, the three single and common architectures could not be powerful enough to solve this problem. I use four different methods here to implement the project. Although various neural networks have been proposed in recent years, convolutional neural network is still the best approach for image feature extraction. According to this, the four different architectures used in this project are all related to convolutional neural network, 3D-CNN can be taken as adding extra one dimension to the traditional 2D-CNN, while the other three are the combination of CNN and other method. And I will illustrate the four different architectures as follows.

Long-term Recurrent Convolutional Networks (LRCN)

The LCRN or CNN+LSTM architecture is first proposed by Donahue and coworkers[14] in 2016. This architecture shown in Figure 8[14] combines the convolutional layers and recurrent layers to learn spatial and temporal information, respectively. LRCN model used to do the human activity recognition is shown in Figure 9[14].

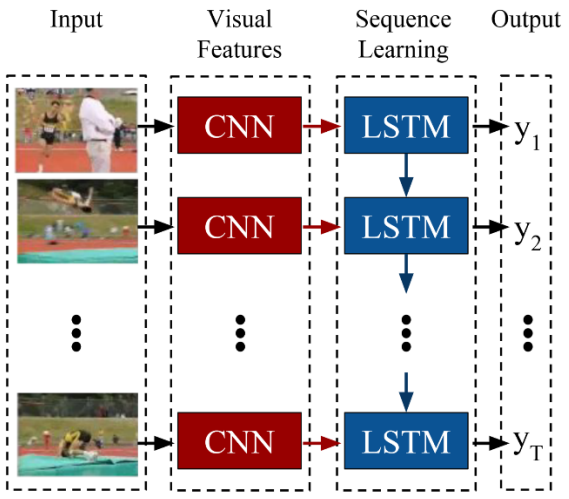


Figure 8. Long-term Recurrent Convolutional Networks (LRCN) architecture

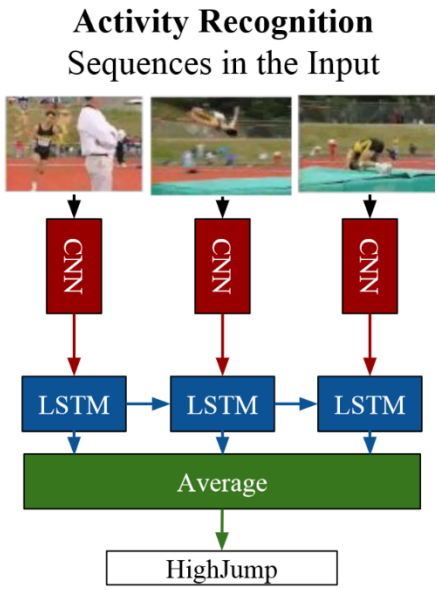


Figure 12. LRCN model for human activity recognition

3D-Convolutional Neural Network

3D Convolutional Neural Network (3D-CNN), which the spatial and the temporal information are merged in the network. It can be taken as a logical extension of 2D-CNN, the difference is that 3D CNN uses 3D filters to do the convolutions and passes the extracted information to pooling layer before the fully connected layer. The basic structure for 3D-CNN is in Figure 10.

Vrskova[15] illustrates more details related to utilize this method to do HAR.

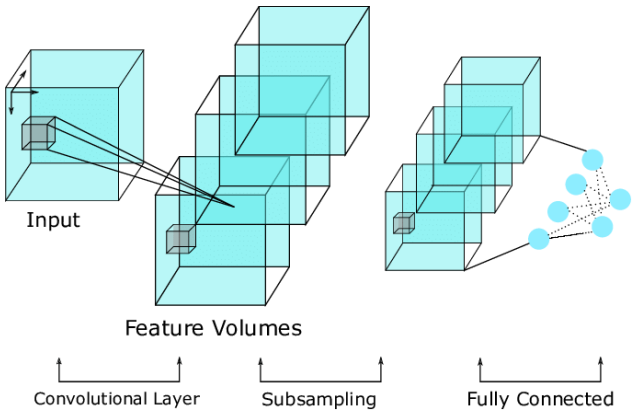


Figure 13. Basic structure for 3D-CNN.

Convolutional LSTM (ConvLSTM)

Convolutional LSTM was proposed by Shi and the coworkers in 2015[16]. It uses the ConvLSTM cell (Figure 11) which is a variant of LSTM containing the convolutional operations in the structure. We can take the ConvLSTM cell as the convolution layer embedded in the LSTM. This architecture can take both the spatial features and keep into account the temporal information.

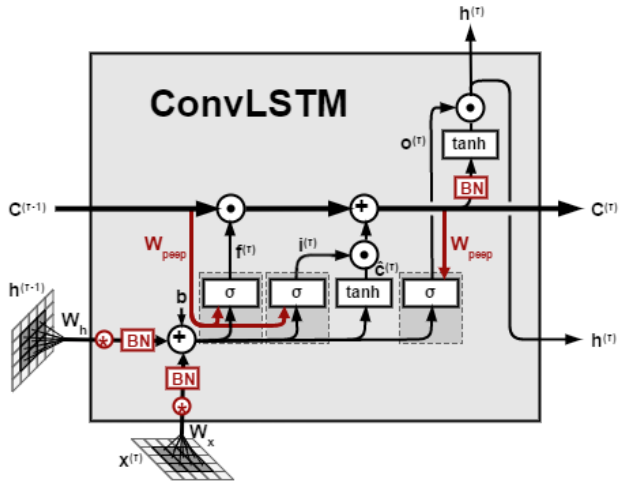


Figure 14. A ConvLSTM cell

CNN-Transformer

According to the above Transformer introduction, we know that Transformer is a pervasive model in deep learning. Since the CNN and be combine with LSTM to do human activity recognition, I am thinking of combining CNN with Transformer. This structure has been proposed in a paper[17], where it uses CNN to do the feature extraction and feed the output of CNN to Transformer Encoder layer to complete the classification (flow diagram shown in Figure 12).

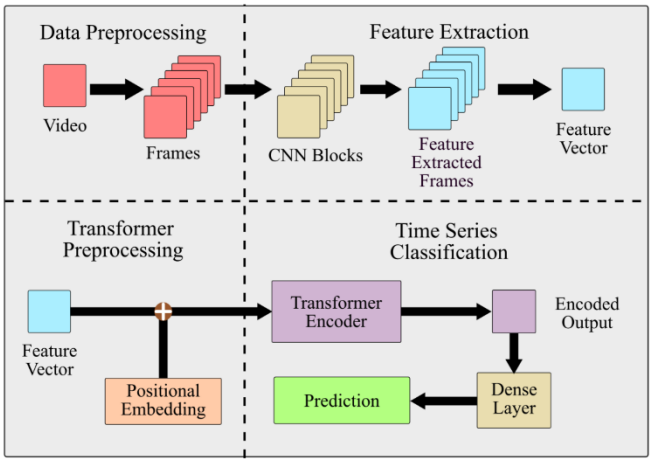


Figure 15. CNN-Transformer structure flow

UCF101 Dataset

UCF-101[1] is a dataset with 101 action categories from YouTube videos, which is an extension for UCF-50, which has 50 action categories. The total 13320 videos in UCF-101 provides diversity in actions and the dataset is created to speed up behavior recognition by learning and exploring new display actions.

The videos are divided into 25 groups, and each group consists of 4-7 different actions. Videos in the same group will have similar scenes. There are mainly five types of action categories: Human-object interaction, Human body movements, Human interaction, Musical instrument, and Sports. In this project, I selected 7 actions from the 101-dataset, and some of them are in the same group which may increase the difficulty of recognizing the activity.

Model building

Data preparation and preprocessing

After downloading and inputting the data file, the first step is to preview the whole file. And I randomly select one file to preview the first image in the first video file. The image is shown in Figure 13.

```
[ ] cv2.putText(frame_rgb, selected_activity, (20,50), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (66, 135, 245), 2)
    plt.imshow(frame_rgb)
```

```
<matplotlib.image.AxesImage at 0x7b92eba9ece0>
```

Figure 16. Preview the first image in a randomly selected video file

I selected 7 classifications to save the computational cost which are ‘JumpRope’, ‘Kayaking’, ‘Lunges’, ‘Diving’, ‘PlayingGuitar’, ‘PlayingPiano’ and ‘Playing Violin’. After that, I start to read the video frame by frame and select 20 frames of each video file. These 20 frames are evenly distributed. The final step for data preparation is to create the dataset for this project.

Different Architecture

Long-term Recurrent Convolutional Networks (LRCN)

For image feature extraction, the pretrained neural network I selected is InceptionResNet which was proposed in 2017. I froze the other layers and only allowed the top fourth layer to be trained (Figure14).

```
[10]: inceptionresnet = InceptionResNetV2(
          include_top=False,
          weights="imagenet",
          input_shape=(100,100,3)
      )

[11]: features_train.shape

 [1]: (771, 20, 100, 100, 3)

[12]: # only train the last layer
      for layer in inceptionresnet.layers[:-4]:
          layer.trainable = False
```

Figure 17. Pretrained Inceptionresnet neural network

The model building for LRCN is shown in Figure 15. Here what needs to be mentioned is the TimeDistributed() layer. When we learned the convolutional layer in the class, we knew the model only needed to take one image to do the classification. But in this case, what we need to process is a sequence of images as input, and to predict what a specific sequence is showing. Luckily, Keras does provide the layer to deal with that kind of data which is Time Distributed layer! TimeDistributed layer[18] apply the same function to several input images, and it produce one output with every input to get the result. TimeDistributed layer is a powerful way to work with the video frames. Instead of having several input models, we just need to have one model for each input and then the LSTM layer can deal with the data in time.

```
[13]: model = Sequential()

model.add(TimeDistributed(inceptionresnet, input_shape=(20, 100, 100, 3)))

model.add(TimeDistributed(Flatten()))

model.add(LSTM(32))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(len(data_classes), activation='softmax'))
```

Figure 18. Model building for LCRN.

We can also visualize the connectivity of the model I just built in Figure 16 using the plot_model function at Chapter 7 in our reference book.

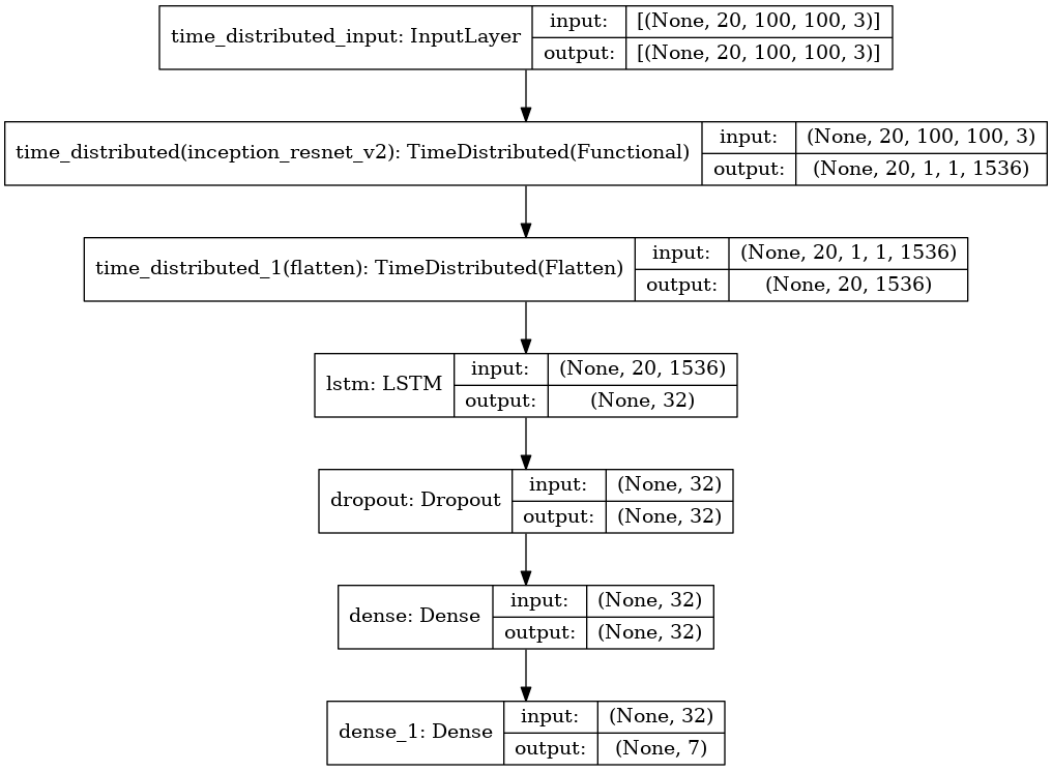


Figure 19. The connectivity of LRCN model

3D-Convolutional Neural Network

For this model building, I am struggling how to use the pretrained neural network, I decided to build the model from scratch. The model building code is shown in Figure 17. And the connectivity visualization for the model is in Figure 18.

```
[11]: model = Sequential()
      model.add(Conv3D(8,(3,3,3), activation='relu', input_shape=(20,100,100,3)))
      model.add(Conv3D(16,(3,3,3), activation='relu'))
      model.add(MaxPooling3D((2,2,2)))

      model.add(Conv3D(32,(3,3,3), activation='relu'))
      model.add(Conv3D(64,(2,2,2), activation='relu'))
      model.add(MaxPooling3D((2,2,2)))
      model.add(Dropout(0.3))
      model.add(Flatten())

      model.add(Dense(128,'relu'))
      model.add(Dropout(0.3))
      model.add(Dense(64,'relu'))
      model.add(Dropout(0.3))
      model.add(Dense(len(data_classes),'softmax'))
      model.summary()
```

Figure 20. Model building from scratch for 3D-CNN

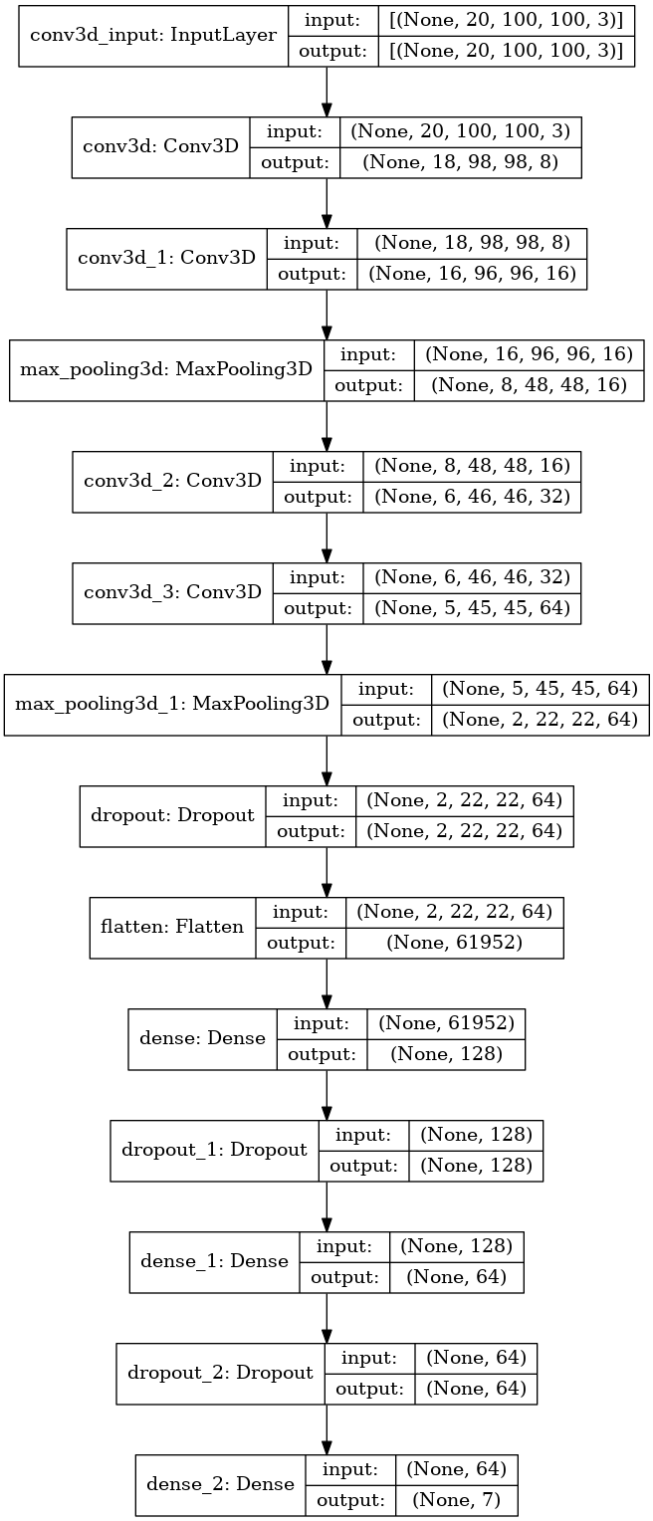


Figure 21. The connectivity of the 3D-CNN

Convolutional LSTM (ConvLSTM)

The same problem has also happened in the ConvLSTM architecture. Since the convolutional and lstm are in the same layer, I build the model from scratch instead of utilizing the pretrained network. The Maxpooling3D layer here is to reduce the dimensions of the frames to improve computational efficiency. Figure 19&20 show the details in the model building.

```
[11]: model = Sequential()

model.add(ConvLSTM2D(filters = 8, padding = "same",  kernel_size = (3, 3),
                     return_sequences = True, data_format = "channels_last", input_shape = (sequence_length,100,100,3)))
model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
model.add(ConvLSTM2D(filters = 10, padding = "same",  kernel_size = (3, 3),
                     return_sequences = True, data_format = "channels_last"))
model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
model.add(ConvLSTM2D(filters = 16, padding = "same",  kernel_size = (3, 3),
                     return_sequences = True, data_format = "channels_last"))
model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
model.add(Flatten())
model.add(Dense(32, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(7, activation = "softmax"))
```

Figure 22. Model building for ConvLSTM

```
[12]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 20, 100, 100, 16)  11008
_________________________________________________________________
dropout (Dropout)            (None, 20, 100, 100, 16)  0
_________________________________________________________________
conv_lst_m2d_1 (ConvLSTM2D)  (None, 20, 100, 100, 32)  55424
_________________________________________________________________
dropout_1 (Dropout)          (None, 20, 100, 100, 32)  0
_________________________________________________________________
flatten (Flatten)            (None, 6400000)           0
_________________________________________________________________
dense (Dense)                (None, 32)                204800032
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 231
=================================================================
Total params: 204,866,695
Trainable params: 204,866,695
Non-trainable params: 0
_________________________________________________________________
```

Figure 23. Model summary for ConvLSTM

CNN-Transformer

The pretrained convolutional neural network used here is DenseNet121. First, I extract the features for the sequence of images and get the features output. Next step is to connect the extracted features with the Transformer Encoder part. The Transformer Encoder code here is referenced on the Github[19].

Result and Discussion

Result for each Model

Here I learned from Chapter 7 in the reference book using the EarlyStopping in each model. In the previous assignment, I realized that every time I run the model for 50 or 70 epochs, I need to wait for a long time, and we can always tell the model has been overfitted in the early epoch. According to this, the method is wasteful. The better way used in Chapter 7 is to handle this with early stop when the validation loss has not been improved for some epochs (Figure 21 for EarlyStopping).

```
[16]: # Create callback.
callbacks = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', restore_best_weights = True)

# Compile the model
model.compile(loss = 'categorical_crossentropy', optimizer = 'rmsprop', metrics = ["accuracy"])

# Start training the model.
history = model.fit(x = features_train, y = labels_train, epochs = 60, batch_size = 6,
                                        shuffle = True, validation_split = 0.2, callbacks = callbacks)
```

Figure 24. Early stopping callback to save computational time

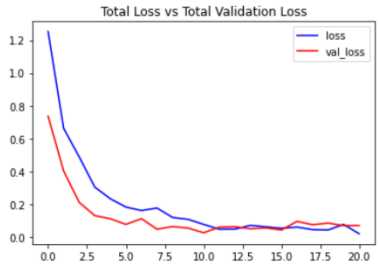
The accuracy for each model is shown in the figure below. Total(training) and validation loss & accuracy is as follows.

	LRCN	3D-CNN	ConvLSTM	CNN+Transformer
Accuracy	96.8%	86.0%	89.6%	81.2%

Figure 25. Accuracy for each model

Long-term Recurrent Convolutional Networks (LRCN)

```
[19]: # Visualize the training and validation loss metrices.
plot_metric(history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
[20]: # Visualize the training and validation accuracy metrices.
plot_metric(history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

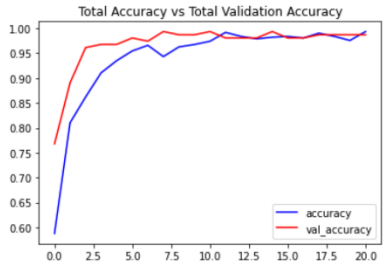
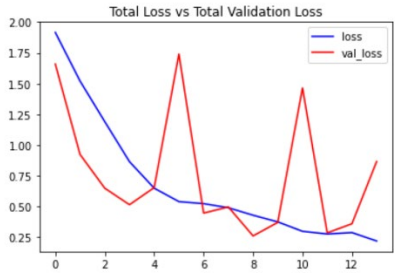


Figure 26. Result visualization for LRCN

3D-Convolutional Neural Network

```
[19]: # Visualize the training and validation loss metrices.
plot_metric(history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
[20]: # Visualize the training and validation accuracy metrices.
plot_metric(history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

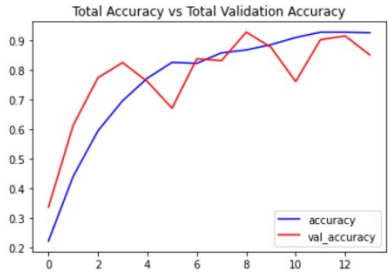
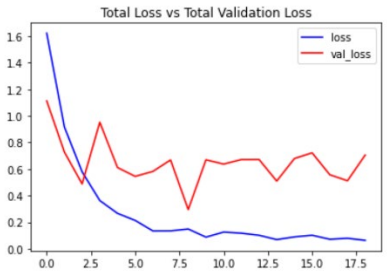


Figure 27. Result visualization for 3D-CNN

Convolutional LSTM (ConvLSTM)

```
[16]: # Visualize the training and validation loss metrices.
plot_metric(history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
[17]: # Visualize the training and validation accuracy metrices.
plot_metric(history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

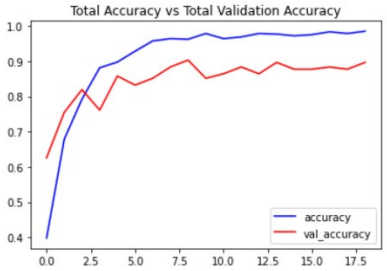
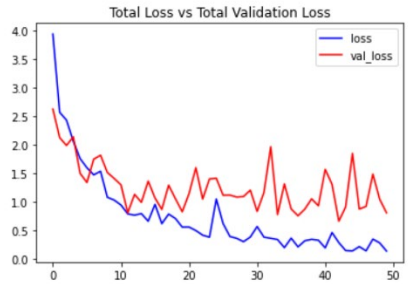


Figure 28. Result visualization for ConvLSTM

CNN-Transformer

```
[33]: # Visualize the training and validation loss metrices.
plot_metric(history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```




```
[34]: # Visualize the training and validation accuracy metrices.
plot_metric(history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

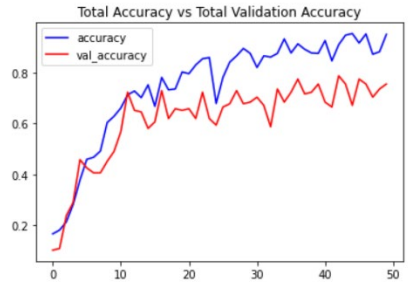


Figure 29. Result visualization for CNN+Transformer

Discussion

For the 4 architectures, LRCN shows the best performance. The number of layers, nodes, and each hyperparameter changed could affect the performance. For LRCN and CNN-Transformer, I utilized pretrained convolutional layers to extract spatial information. However, for the other two models, incorporating a pretrained model proved challenging, as the CNN is either directly combined with an LSTM unit or uses a 3D filter. ConvLSTM's accuracy results indicate overfitting due to the absence of a pretrained model.

While CNN-Transformer achieves an accuracy of 81.2%, it's important to note that this doesn't diminish its efficacy compared to LRCN, which has an accuracy of 96.8%. Despite CNN+Transformer having a faster training time than LRCN, the impact on training time is negligible for our moderately sized model. The drawback of employing the transformer lies in its coding complexity; although I referenced code from Github, I encountered difficulties with the remaining code. In contrast, LRCN stands out for its simplicity in understanding and straightforward coding process.

As highlighted in the abstract, I explored five methods and implemented four of them, leaving the fifth as an average of each single frame's probabilities. This method has limitations, particularly in activities like sitting and standing, where it may yield incorrect results due to its disregard for temporal data.

While computer vision-based methods in Human Activity Recognition offer promising results, they face challenges related to privacy and light dependency. Cameras may struggle to capture videos at night, highlighting the limitations of this approach. Despite these issues, computer vision remains an initial and foundational approach in HAR, aligning with the focus on deep

learning in this course, prompting an exploration of various neural network architectures for the completion of this project.

Conclusion

In this project, I employed four distinct architectures to conduct Human Activity Recognition: Long-term Recurrent Convolutional Networks (LRCN), 3D-Convolutional Neural Network, Convolutional LSTM (ConvLSTM), and CNN-Transformer. The primary emphasis of this project is to experiment with various models for the same problem, providing valuable practice for deep learning studies.

Considering factors such as ease of understanding, coding complexity, and performance results, I would suggest utilizing Long-term Recurrent Convolutional Networks (LRCN) as the preferred computer vision-based method for Human Activity Recognition.

References

(1) Soomro, K.; Zamir, A. R.; Shah, M. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. arXiv December 3, 2012. https://doi.org/10.48550/arXiv.1212.0402.

(2) Vrigkas, M.; Nikou, C.; Kakadiaris, I. A. A Review of Human Activity Recognition Methods. *Front. Robot. AI* **2015**, *2*.

(3) Lara, O. D.; Labrador, M. A. A Survey on Human Activity Recognition Using Wearable Sensors. *IEEE Commun. Surv. Tutor.* **2013**, *15* (3), 1192–1209. https://doi.org/10.1109/SURV.2012.110112.00192.

(4) Gani, O. A Novel Approach to Complex Human Activity Recognition.

(5) Hussain, Z.; Sheng, M.; Zhang, W. E. Different Approaches for Human Activity Recognition: A Survey. *J. Netw. Comput. Appl.* **2020**, *167*, 102738. https://doi.org/10.1016/j.jnca.2020.102738.

(6) Kang, C. *RNN - Many-to-one*. Chan`s Jupyter. https://goodboychan.github.io/python/deep_learning/tensorflow-keras/2020/12/06/01-RNN-Many-to-one.html (accessed 2023-11-16).

(7) Vandans, O.; Yang, K.; Wu, Z.; Dai, L. Identifying Knot Types of Polymer Conformations by Machine Learning. *Phys. Rev. E* **2020**, *101* (2), 022502. https://doi.org/10.1103/PhysRevE.101.022502.

(8) Wei, Q.; Melko, R. G.; Chen, J. Z. Y. Identifying Polymer States by Machine Learning. *Phys. Rev. E* **2017**, *95* (3), 032504. https://doi.org/10.1103/PhysRevE.95.032504.

(9) Team, T. A. *Convolutional Neural Networks for Dummies – Towards AI*. https://towardsai.net/p/deep-learning/convolutional-neural-networks-for-dummies, https://towardsai.net/p/deep-learning/convolutional-neural-networks-for-dummies (accessed 2023-11-16).

(10) Ratan, P. *What is the Convolutional Neural Network Architecture?*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/ (accessed 2023-11-16).

(11) Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9* (8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

(12) Dinesh. *Beginner's Guide to RNN & LSTMs*. Medium. https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf (accessed 2023-11-16).

(13) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. arXiv August 1, 2023. https://doi.org/10.48550/arXiv.1706.03762.

(14) Donahue, J.; Hendricks, L. A.; Rohrbach, M.; Venugopalan, S.; Guadarrama, S.; Saenko, K.; Darrell, T. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. arXiv May 31, 2016. https://doi.org/10.48550/arXiv.1411.4389.

(15) Hlavatá, R.; Hudec, R.; Kamencay, P.; Sykora, P. Human Activity Classification Using the 3DCNN Architecture. *Appl. Sci.* **2022**, *12*, 931. https://doi.org/10.3390/app12020931.

(16) Shi, X.; Chen, Z.; Wang, H.; Yeung, D.-Y.; Wong, W.; Woo, W. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. arXiv September 19, 2015. https://doi.org/10.48550/arXiv.1506.04214.

(17) Wensel, J.; Ullah, H.; Munir, A. ViT-ReT: Vision and Recurrent Transformer Neural Networks for Human Activity Recognition in Videos. *IEEE Access* **2023**, *11*, 72227–72249. https://doi.org/10.1109/ACCESS.2023.3293813.

(18) Ferlet, P. *How to work with Time Distributed data in a neural network*. Smile Innovation. https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00 (accessed 2023-11-18).

(19) *Video-Classification-Using-Transformer/trained_vit(no_tubelet).ipynb at main · akashghimireOfficial/Video-Classification-Using-Transformer*. GitHub. https://github.com/akashghimireOfficial/Video-Classification-Using-Transformer/blob/main/trained_vit(no_tubelet).ipynb (accessed 2023-11-18).