SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.039 : Theory and Practice of Deep Learning

Report of Big Project
Choice 1 - HandWash Recognition

Group 5

Phang Teng Fone (1003296)
Loh De Rong (1003557)
Joey Yeo Kailing (1003846)
Yeh Swee Khim (1003885)
Koh Hui Wen (1003593)

Introduction

Due to the recent COVID-19 outbreak, handwashing with soap can be one of the defenses against the virus. By practising hand hygiene, it can be used to protect us from these diseases, as such the practice of handwashing at regular intervals should be encouraged and promoted. With the seven-step hand washing technique (broken down into 12 actions) endorsed by the Centers for Disease Control and Prevention (CDC) and World Health Organization (WHO) [1], we would like to promote this proper hand washing technique to ensure that all hand washing steps are followed correctly. By using deep learning to classify each hand washing action, the missing actions can be identified to remind users to complete them.

Related Works

Considerable research has been done in the field of action recognition. Their approaches mainly fall into one of the two following categories: (i) temporal integration using pooling, fusion or recurrent architectures and (ii) 3D Convolutional Neural Network (CNN) architectures.

Pooling is a well-known technique to combine various temporal features. Fusion is also frequently used and sometimes more preferred due to its ability to emphasize pooling location in the architecture or to differentiate information from different modalities. Inspired by the ventral stream and dorsal stream in the human visual cortex, K. Simonyan et al. [2] proposed an architecture based on two separate recognition streams (spatial and temporal), which are then combined by late fusion. The spatial stream performs action recognition from still video frames, whilst the temporal stream is trained to recognise action from motion in the form of dense optical flow. Recurrent Neural Networks (RNN), especially LSTMs, are also commonly utilized for temporal modelling on 2D CNN features extracted from the frames of a video.

In 3D convolution, channels and temporal information are represented as different dimensions. As compared to the temporal fusion methods, this design allows the temporal information to be processed hierarchically and throughout the network. As a method, M. E. Kalfaoglu et al. [3] proposed a late temporal modeling in 3D CNN architectures with BERT, which is the current state-of-the-art model, achieving a high accuracy score of 98.69% and 85.10% on the UCF101 and HMDB-51 datasets respectively.

Action recognition tasks also extend to the healthcare domain. A. Nagaraj et al. [4] used a three-stream algorithm to tackle the classification problem of distinguishing hand washing steps. The three streams include the temporal and spatial streams presented in [2], and a third stream of object-level features using the Histogram of Oriented Gradients, which were used as descriptors to improve the accuracy of the model. Overall, they were able to achieve an accuracy score of 86.6% using the three-stream algorithm on the hand wash dataset.

In our project, we will explore other types of architectures using a combination of CNN and RNN, and then compare our results with the three-stream algorithm on the same hand wash action recognition task.

Our Approach

Understanding the Dataset

The Hand Wash Dataset [5] consists of 25 individual videos for each of the 12 hand wash actions (Figure 1). This gives a total of 300 videos for the entire dataset.

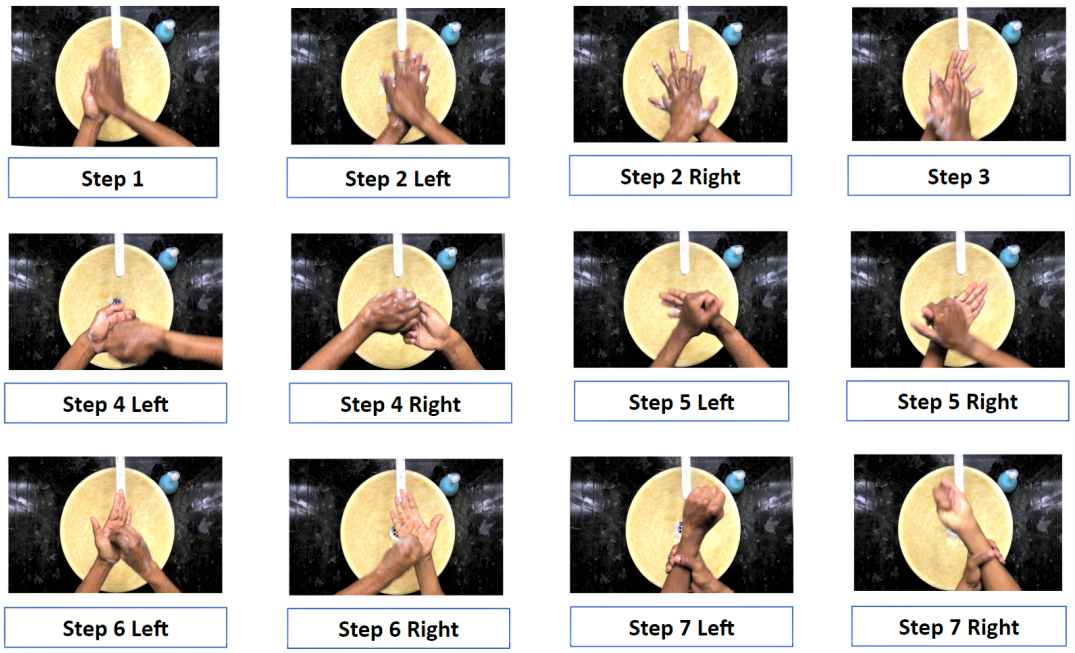

Figure 1: 12 classes of actions

Given the small sample size of the Hand Wash dataset, we recorded more hand washing videos to increase the dataset size. To increase the robustness of the model, we took videos in widely different settings and environments to provide as much variance as possible. The varied parameters include camera position, actor, background and illumination.

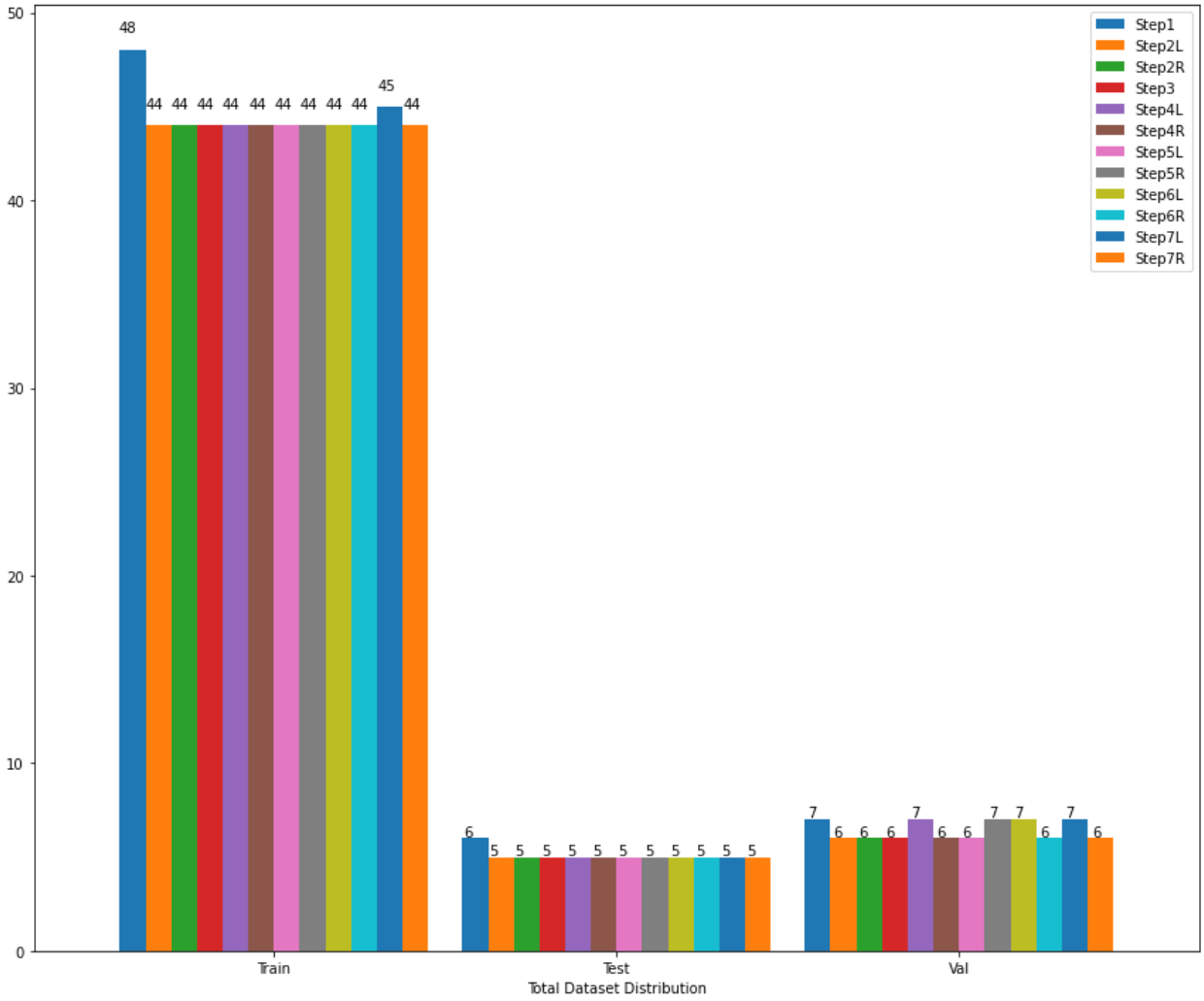

Figure 2: Distribution of the hand washing videos

After combining our self-collected and the online dataset, we did a stratified train-validation-test split, which gives rise to the distribution as shown in Figure 2. We ensured that the final distribution is balanced, which prevents the model from being overly biased toward any majority class during training and inference.

Preprocessing videos into numpy arrays

Processing of each video, including open, sampling and closing, takes approximately 1 second. Without first converting the videos into 3D numpy arrays, the model would have to redo this processing operation for every iteration, resulting in extremely long training times. Therefore, it was imperative to perform a preprocessing step once at the start so that subsequent iterations are able to access the already processed videos to reduce unnecessary computational cost.

Specifically, every fourth frame of each video was sampled and resized to 128 x 128 until a total of 32 frames was reached. For the purposes of more efficient model training, we would be working with this processed numpy dataset.

Preprocessing of numpy arrays

The numpy arrays had to be further processed before being fed into the neural network. First, a series of consecutive frames was sampled from the numpy dataset using the time cropping function. Here, the number of selected frames refers to the number of timesteps for the RNN layer. Second, normalization was done by dividing each pixel by 255. This allows faster convergence when training our model, ensuring computational efficiency as larger values can slow down the learning process significantly. Third, standard data augmentation was applied to the train dataset to reduce overfitting. In particular, we explored translation in the horizontal direction and contrast as augmentation techniques.

Evaluation Metrics

Since this is a classification problem, we will be using accuracy as the main metric to evaluate the model performance. The accuracy is calculated as follows:

$$Accuracy = \frac{No.\ of\ correct\ predictions}{Total\ no.\ of\ samples} * 100\%$$

In addition, we constructed the confusion matrix in order to better understand which actions that tend to be misclassified leading to the poor model performance. A confusion matrix effectively reports the number of predicted and true values for each class.

Experiments

Different Architectures

In this work, we experimented with two different approaches: ConvLSTM and CNN-LSTM. We then shortlisted the best model based on their validation accuracy. Further experiments using hyperparameter tuning and data augmentation are described in the subsequent subsections.

ConvLSTM

ConvLSTM is a type of RNN with gates similar to LSTM, except that the internal matrix multiplications are exchanged with convolution operations. This implementation allows the layers to extract spatio-temporal features from the video frames.

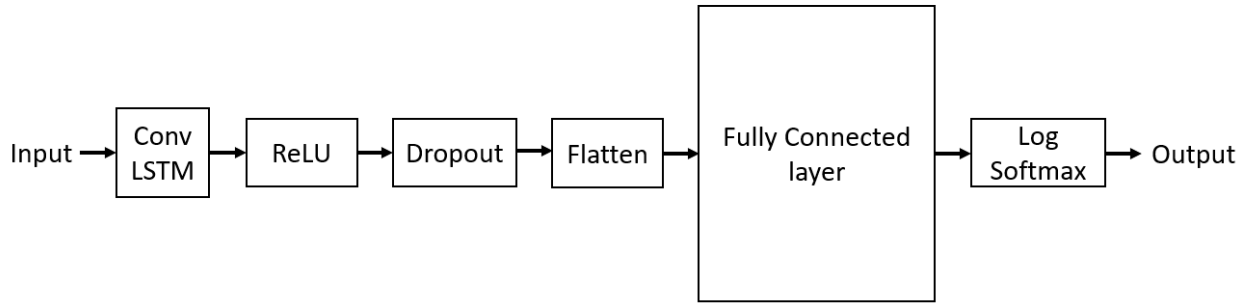

Figure 3: ConvLSTM model architecture.

We modified the ConvLSTM model [6] and came up with the final model architecture consisting of one ConvLSTM layer with ReLu activation and one Dropout layer, followed by a fully connected layer with softmax for classification (Figure 3). Initial experiments with more layers result in model overfitting, which is likely due to our small dataset size.

CNN-LSTM

CNN-LSTM model is a typical architecture for many video action recognition tasks (Figure 4). The CNN model can first extract high-level spatial features from the frames of the input video. Initially, we used a simple CNN architecture for the CNN backbone, with only one 2D convolutional layer, one fully-connected layer with ReLU activation and one Dropout layer. We also experimented with transfer learning using pre-trained backbones such as AlexNet and ResNet-50.

The extracted spatial features are then fed into the LSTM layer to extract temporal correlation between the frames by remembering past ones. Finally, the last output from the LSTM layer can be put into a final fully connected layer with softmax for classification.

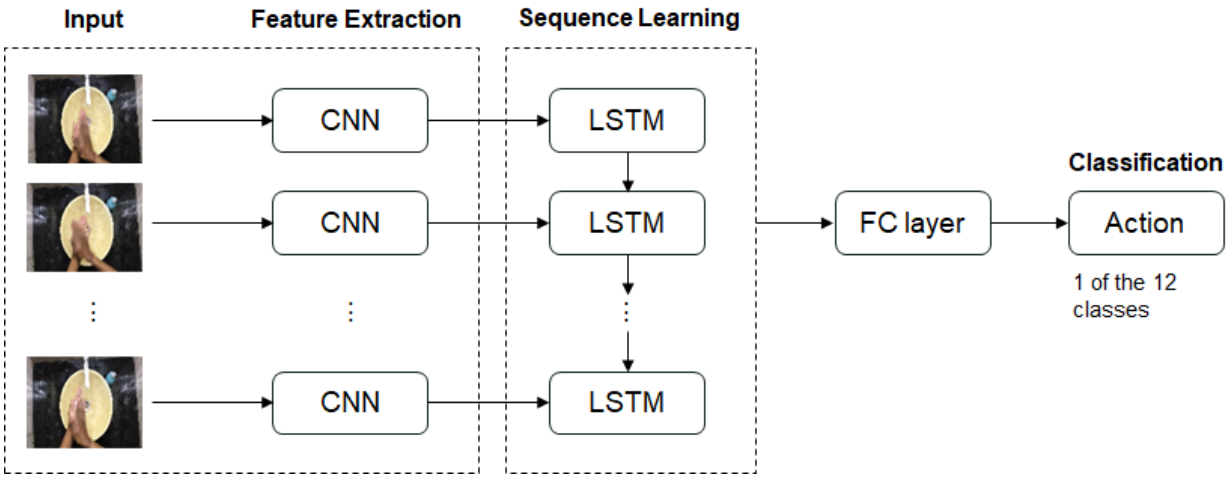

Figure 4: CNN-LSTM Model Architecture

Comparing performances of architectures

We trained for 50 epochs for each of the four proposed architectures and recorded the validation accuracies in Table 1 to determine which is the best architecture.

Model	**Validation Accuracy**
ConvLSTM	29.9%
CNN-LSTM with custom CNN layers	10.4%
CNN-LSTM with ResNet-50 backbone	32.5%
CNN-LSTM with AlexNet backbone	**58.4%**

Table 1: Comparing the performance of different model architectures

The CNN-LSTM with AlexNet backbone was able to achieve the highest validation accuracy of 58.4%. Comparing the pre-trained CNN models, ResNet50 and AlexNet, ResNet-50 performs worse than AlexNet. This could be because the ResNet-50 has much more parameters than AlexNet and given our small dataset size, a model with too many parameters will result in overfitting.

Nonetheless, using pre-trained CNN models can help to boost the accuracy and improve the model performance, as compared to training a custom model from scratch as shown in Table 1, where both the ConvLSTM model and CNN-LSTM model with custom CNN layers yielded poorer validation accuracies. In particular, the performance of the CNN-LSTM model with custom CNN layers was almost as poor as a random model, which would have an expected validation accuracy of 8.3%. This is because the pretrained models have already learnt general features that can be reused, making it easier for the models to learn and converge.

Therefore, in subsequent experiments and training of the final model, we will use the CNN-LSTM architecture with pretrained AlexNet weights in the CNN layers.

Hyperparameter Tuning

We conducted a grid search to find the best combination of hyperparameters, including batch size and learning rate. The search space for batch size was [8, 16, 32] and the search space for learning rate was [0.01, 0.001, 0.0001]. Each combination was run for 20 epochs. The experimental results are recorded in Table 2.

Batch Size	**Learning Rate**	**Validation Accuracy**
8	0.01	9.1%
8	0.001	50.6%

8	0.0001	32.5%
16	0.01	9.1%
16	0.001	48.1%
16	0.0001	20.8%
32	0.01	19.5%
32	**0.001**	**51.9%**
32	0.0001	20.8%

Table 2: Grid Search on Hyperparameters

We find that a batch size of 32 and learning rate of 0.001 achieves the highest validation accuracy of 51.9%. Hence, we will use these hyperparameters for training our final model.

We also varied the spatial dimensions of the input frames from the video, and the validation accuracy is recorded in Table 3.

Spatial Dimension	**Validation Accuracy**	**Time taken (mins)**
64 x 64	54.5%	4:56
84 x 84	57.1%	5:02
128 x 128	**83.1%**	7:50

Table 3: Varying Spatial Dimensions

From Table 3, it is evident that as spatial dimension increases, the validation accuracy also increases. This is because the larger the spatial dimension, the higher the resolution is of each input frame. The CNN layers would then be able to extract the spatial features more effectively since the finer details of the hands are present. This could help the model better distinguish between similar handwashing actions.

However, the time taken to train the model for the same number of epochs increases with the spatial dimension, since the kernels have to convolve over a larger pixel area. There is a tradeoff between the model performance and time taken during training, but comparing the spatial dimensions of 84 x 84 and 128 x 128, the increase in validation accuracy is much more significant (about 26% accuracy) than the increase in time taken (about 2:50 min). It is much better to train for longer periods of time to achieve better model performance than to train quickly but have poor performance.

Data Augmentation

Using the AlexNet as the pretrained CNN model with a batch size of 32, learning rate of 0.001 and frame size of 128 by 128, we applied data augmentation on the original dataset, using translation and contrast with the goal of reducing overfitting.

Data Augmentation	**Validation Accuracy**
None	**83.1%**
Translation	81.8%
Contrast	80.5%

Table 4: Validation Accuracy on AlexNet with Different Data Augmentation

From Table 4, data augmentation did not improve the validation accuracy even though in principle, data augmentation should increase the total number of effective training samples and thereby reduce overfitting. This may be because we froze most of the layers of the AlexNet backbone and are only updating the top few layers and the LSTM layer, thus the bottom layers are unable to generalize well on the augmented data. Introducing the new augmented data during the fine-tuning process could disturb the predictive power of the model, resulting in a decrease in overall model performance. Data augmentation may thus work better when we train the model from scratch.

Results and Discussion

Final Evaluation on Test Set

Overall, using our best model, the CNN-LSTM model with pretrained AlexNet backbone, batch size of 32, learning rate of 0.001 and frame size of 128 by 128, we are able to achieve a test accuracy of **88.5%**.

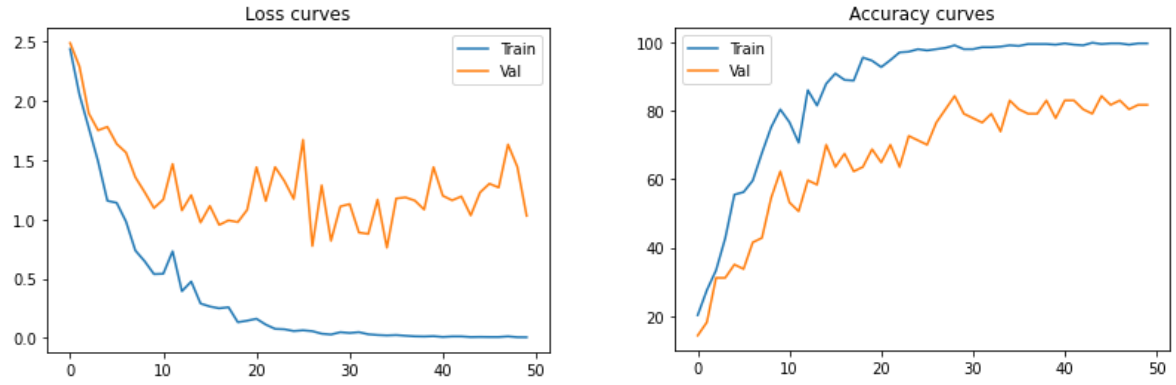

Figure 5: Loss and Accuracy Train and Validation Curves

From Figure 5, we can see that the validation loss is much higher than the train loss, and that the validation accuracy is much lower than the train accuracy. Moreover, we can see that the validation loss is quite volatile, which means that our model is overfitting. This could be due to our dataset being very small in size, resulting in the model not seeing enough samples to be able to generalize well. Nonetheless, we have reduced overfitting as much as possible by adding regularization such as Dropout layers and saving the model based on best validation accuracy.

Actual Classes

Predicted Classes

	Step 1	Step2 Left	Step 2 Right	Step 3	Step 4 Left	Step 4 Right	Step 5 Left	Step 5 Right	Step 6 Left	Step 6 Right	Step 7 Left	Step 7 Right
Step 1	**6**	0	0	0	0	0	0	0	0	0	0	0
Step2 Left	0	**5**	0	0	0	0	0	0	0	0	0	0
Step 2 Right	0	0	**5**	0	0	0	0	0	0	0	0	0
Step 3	0	0	0	**4**	0	0	0	0	0	0	0	0
Step 4 Left	0	0	0	0	**5**	0	0	0	0	0	0	0
Step 4 Right	0	0	0	0	0	**4**	0	0	0	0	0	**1**
Step 5 Left	0	0	0	0	0	0	**5**	0	0	0	0	0
Step 5 Right	0	0	0	0	0	**1**	0	**5**	0	**1**	0	0
Step 6 Left	0	0	0	0	0	0	0	0	**4**	0	0	0
Step 6 Right	0	0	0	0	0	0	0	0	**1**	**3**	0	0
Step 7 Left	0	0	0	0	0	0	0	0	0	0	**4**	0
Step 7 Right	0	0	0	**1**	0	0	0	0	0	**1**	**1**	**4**

Figure 6: Confusion Matrix of Test Set

Overall, our proposed solution is able to predict the correct classes for most of the test samples (Figure 6). However, some classes are harder to predict, such as Step 6 Right. There are some “weird” cases of model misclassification in Step 6 Right where the actual and wrongly predicted classes do not look similar. This can be seen in Figure 7a where Step 6 Right has been wrongly classified as Step 5 Right, when the actual hand washing action for Step 5 Right is as shown in Figure 7b. Another instance of a “weird” misclassification is when Step 6 Right is wrongly predicted as Step 7 Right as shown in Figure 8a and Figure 8b.

Figure 7a: Step 6 Right wrongly classified as Step 5 Right

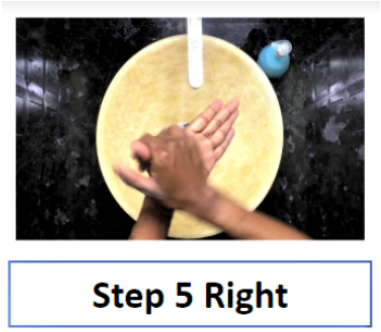

Figure 7b: Actual video frame of Step 5 Right

Figure 8a: Step 6 Right wrongly classified as Step 7 Right

Figure 8b: Actual video frame of Step 7 Right

From the confusion matrix on the test set in Figure 6, we can see that there is one case where Step 6 Left is wrongly predicted as Step 6 Right, and one case where Step 7 Left is wrongly predicted as Step 7 Right. This indicates that there is some difficulty in differentiating between the left and right actions, which is one of the possible reasons for misclassification in our model.

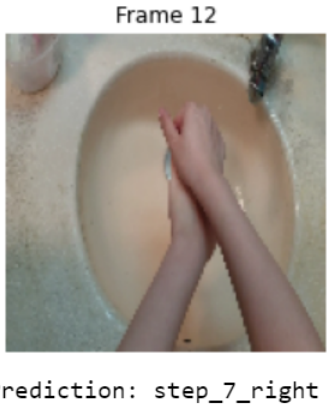

Figure 9a: Step 2 Left wrongly classified as Step 7 Right

Figure 9b: Actual video frame of Step 7 Right

Additionally, if the sink and the hands are of similar colours, the model may also malfunction. An example of misclassification is when Step 2 Left is wrongly predicted as Step 7 Right as shown in Figure 9a and Figure 9b. One way to confirm if the model is learning only the hand features instead of the background is to plot a heatmap of the most sensitive pixels using gradient sensitivity methods.

Performance Comparison with Three-Stream-Algorithm

Models	**Test Accuracy**
Three-Stream-Algorithm	86.6%
Our CNN-LSTM model with pretrained AlexNet backbone	**88.5%**

Table 5: Test accuracy for comparison between models

The three-stream-algorithm [4] consists of RGB Frames, Optical Flow and Histogram of Oriented Gradients models. These models were pre-trained on the ImageNet dataset then further trained on the Hand Wash dataset. The three-stream-algorithm was trained on the hand wash dataset with 3,504 video clips, while our proposed CNN-LSTM model only trained on 711 video clips.

Comparing the performances from Table 5, our proposed architecture was able to achieve about the same test accuracy as the three-stream-algorithm. Given our significantly smaller dataset, it is possible that our proposed solution is quite effective in achieving state-of-the-art performance.

Since both the state-of-the-art model and our model use pre-trained neural networks that have shown relatively high accuracy scores, this proves the effectiveness of using transfer learning

on hand washing datasets. The state-of-the-art model uses DenseNet-BC pretrained with ImageNet weights and our model uses pretrained AlexNet. These pretrained models might have been trained on similar datasets as the hand washing dataset, making them reusable and effective. Moreover, they do not have too many learnable parameters that will cause the models to overfit as compared to ResNet-50 that have a large number of parameters, hence resulting in a poor accuracy score during our experiment. Hence, although transfer learning tends to allow rapid progress and improves the performance of models, the right pretrained model must be used to ensure its effectiveness on the task that it is being trained on.

However, given the significant difference in our dataset sizes, their hand wash dataset might contain a larger diversity of environmental variables. In addition, since their model will be able to capture more features from their larger dataset size, it might be more generalizable in classifying hand washing videos. In comparison, our dataset size might be too small for our model to be generalizable, hence causing it to overfit easily. Therefore, if we were to test our model on a larger test set with more variance of environmental factors, our test accuracy might not be able to perform as well.

User Interface

A simple GUI where the user can upload and select a group of custom video clips with different hand washing steps and input them into our model (Figure 10). After the model outputs the hand washing steps for the respective video clips, the GUI will display whether all hand washing actions have been executed, and if not, display the missing actions (Figure 11).

Figure 10: Video file upload for classification

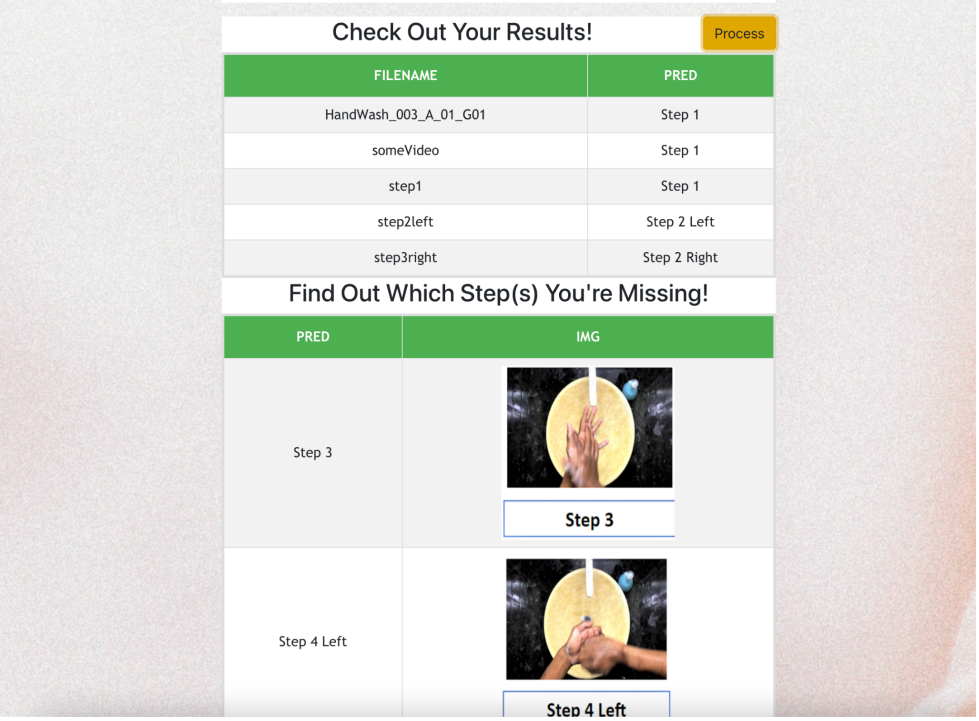

Figure 11: Predicted outputs of each video

By loading our pretrained AlexNet model, we can easily reuse the weights for prediction at the application layer. Furthermore, the model is able to run only on CPU, making deployment using an average EC2 instance possible.

A video link for the demo is uploaded here: https://www.youtube.com/watch?v=DLfKYGBf7oE.

A web demo link running the pretrained AlexNet can be found at http://3.236.221.43:8081/. Please note that this link will only be available for the course duration. If the link expires, kindly refer to our GitHub repository for instructions on how to run locally.

Conclusion

In conclusion, our proposed solution consists of a CNN-LSTM architecture with Alexnet backbone, and was able to achieve a high test accuracy of 88.5%, which is a comparable performance to the state-of-the-art models. While the model is unable to predict the correct classes all the time, it gives a good gauge about which of the 12 handwashing actions were done or left out. As such, the implementation of our solution would promote hygiene and healthy habits in order to protect oneself against the pandemic.

Future Work

An area of improvement is the expansion of the handwash dataset. Increasing the dataset with diverse environmental conditions such as different brightness or different background sinks would help make the model more robust and be able to generalize well on unseen data. With a larger dataset, it also becomes more possible to train the architecture from scratch and employ data augmentation techniques to further improve model robustness.

As our model misclassified some of the steps, one method to tackle this is to have model interpretability methods such as using the LIME algorithm which attempts to identify the model features to understand how the predictions change. Once the features are identified, we can then tune the neural network to target those misclassified features to improve model accuracy.

Moreover, as there are limitations to classifying coloured images, given different coloured sinks or different colour of the skin, 3D skeleton-based action recognition can be useful to analyse spatial joint relations and temporal posture dynamics. With the skeleton data, it might allow our model to learn more features of each action to improve the model accuracy [7].

It may be good to also explore larger spatial dimensions, since from our experiments in Table 3, the larger the spatial dimensions, the higher the validation accuracy. By further investigating the tradeoffs, the extent to which the accuracy stops increasing can be determined and the most optimal spatial dimension can be used to further improve the model.

State-of-the-art action recognition models such as 3D-ConvNet can also be explored for better accuracy of the model as it is able to learn spatiotemporal features by incorporating 3D filters and pooling kernels from a 2D architecture (DenseNet) which enables the model to be able to learn motion characteristics of the hand washing steps.

Lastly, in order to correctly classify input videos that do not belong in any of the 12 classes, a binary classifier can be separately trained to distinguish these videos correctly. During prediction, the sample can be fed into the binary classifier, and only be inputted in the CNN-LSTM architecture if the binary classifier had predicted that one of the 12 classes were present.

Contributions

The contributions of each group member are listed in Table 6.

Name	**Contribution**
Teng Fone	● Literature review on related works ● Dataset visualization ● Implementation of UI ● Report

De Rong	● Literature review on related works ● Data preprocessing into numpy arrays ● Implementation of video processing function ● Implementation of predict function ● Implementation of transfer learning in CNN-LSTM architecture ● Experimental design ● Report
Joey	● Implementation of ConvLSTM architecture ● Implementation of evaluate function ● Implementation of data augmentation ● Experiments for different architectures ● Experiments for hyperparameter tuning ● Report
Swee Khim	● Implementation of ConvLSTM architecture ● Implementation of evaluate function ● Implementation of data augmentation ● Experiments for different architectures ● Experiments for hyperparameter tuning ● Report
Hui Wen	● Implementation of Dataset class ● Implementation of basic CNN-LSTM architecture ● Implementation of train function ● Implementation of evaluate function ● Implementation of function to plot learning curves ● Implementation of data augmentation ● Report

Table 6: Contributions of group members

How To Run The Codes

Github: https://github.com/huiwen99/HandWash

Model Architecture

Python Dependencies

- `torch`
- `torchvision`
- `matplotlib`
- `numpy`
- `pandas`
- `cv2`
- `os`
- `argparse`
- `random`
- `datetime`

Dataset

Download the pre-processed numpy dataset to the root directory:
`wget https://storage.googleapis.com/dl-big-project/dataset.zip`

Unzip the file: `unzip ./dataset.zip`

Reproducibility

Our model weights are too large to fit in this repository. Download the weights of our best model into the `save_weights` folder by running the command:
`wget https://storage.googleapis.com/dl-big-project/alexnet_128.pt`

Instructions to run the python files in the notebook

main branch

1. `train.py` : Trains the chosen architecture on the numpy dataset. The default model is CNN-LSTM with custom CNN layers.

   To train the default model, run the following command: `!python train.py`

   Optional parameters:

   `--arch`: set architecture (either `convlstm` or `alexnet` or `resnet50` or `custom`)

   `--epochs`: set number of training epochs

   `--batch`: set batch size

   `--num_frames`: set number of frames per video

   `--lr`: set learning rate

   `--beta1`: set first momentum term for Adam optimizer

`--beta2`: set second momentum term for Adam optimizer

`--weight_decay`: set weight decay for regularization on loss function

`--gamma`: set gamma for learning rate scheduler

`--step_size`: set step size for learning rate scheduler

`--cuda`: enable cuda training

`--checkpoint`: filepath to a checkpoint to load model

`--save_dir`: filepath to save the model

`--data_aug`: set data augmentation type: None or `constrast` or `translate`

`--aug_prob`: decide on the probability of the dataset to perform data augmentation

2. `evaluate.py` : Evaluate the trained model on the test set.

   After training the model and saving it, we can evaluate the model on the test set.

   For example, the model is saved as `"./alexnet_128.pt"`.

   Run the following command:

   ```
   !python evaluate.py --checkpoint "./alexnet_128.pt" --arch alexnet
   ```

   where the `arch` parameter has to match the architecture of the saved model.

   Optional parameters:

   `--dataset`: choose dataset to evaluate on -- validation or test

   `--batch`: set batch size

   `--cuda`: enable cuda

3. `predict.py` : Predicts the class of a video using the trained model.

   After training the model and saving it, we can use the model to predict the class of handwash videos.

   Run the following command:

   ```
   %run predict.py --checkpoint "./alexnet_128.pt" --video_path video_file_path
   ```

   where `video_file_path` is the file path to the video.

Optional parameters:

`--arch`: set architecture (either `convlstm` or `alexnet` or `resnet50` or `custom`)

`--cuda`: enable cuda

Experiments -- Description of notebooks

1. `Model Experiments.ipynb` : Training and experiments are done here.
    a. Tested out 4 different architectures ConvLSTM and CNN-LSTM with AlexNet, ResNet-50 and custom.
    b. After finding the best model amongst the 4 architectures, apply hyperparameter tuning by varying the batch size, learning rate and spatial dimensions.
    c. Once we have derived the best parameters to use with the best model, apply data augmentation such as contrast and translation.
2. `Model Experiments Testing.ipynb` : Testing of experimental models on validation and test sets.

- Follow the instructions in the `Model Experiments Testing.ipynb` notebook if you would like to download the experimental models and retest them on the validation set.

UI Implementation

Dependencies

- `React`
- `PyTorch`
- `Numpy`
- `NodeJS`
- `OpenCV`

Pre-requisites

Download pre-trained AlexNet model to `./HandWash/handwashUI/machine_learning/model`:
`wget https://storage.googleapis.com/dl-big-project/alexnet_128.pt`

Installation

Local:

Change directory to handwashUI: `cd HandWash/handwashUI/` then install required npm packages: `npm install`.

Run the server and front end concurrently by: `npm run dev`

Cloud Server:

Tested on AWS EC2 Instance (t2.medium with 16GB Disk Storage on Ubuntu LTS 20.X). Note that the API URL axios address calls found in `App.js` and `fileupload.js` must be changed.

References

[1] Boshell, P. (2016). What Is The Correct Hand Washing Technique? Debgroup. Retrieved April 20, 2021, from https://info.debgroup.com/blog/what-is-the-correct-hand-washing-technique

[2] Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in neural information processing systems. pp. 568–576 (2014)

[3] M. E. Kalfaoglu, S. Kalkan, and A. A. Alatan, “Late temporal modeling in 3D CNN architectures with BERT for action recognition,” arXiv [cs.CV], 2020.

[4] A. Nagaraj, M. Sood, C. Sureka, and G. Srinivasa, “Real-time Action Recognition for Fine-Grained Actions and The Hand Wash Dataset,” 2019.

[5] real-timeAR, “Sample: Hand Wash Dataset,” *Kaggle*, 19-Apr-2020. [Online]. Available: https://www.kaggle.com/realtimear/hand-wash-dataset.

[6] A. Palazzi, H. Yu, and S. Pini, “ndrplz/ConvLSTM_pytorch,” *GitHub*, 2017. [Online]. Available: https://github.com/ndrplz/ConvLSTM_pytorch.

[7] T. Huynh-The and D. Kim, "Data Augmentation For CNN-Based 3D Action Recognition on Small-Scale Datasets," 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), 2019, pp. 239-244, doi: 10.1109/INDIN41052.2019.8972313.