When to Interrupt

Akash Sindhu
asa325@sfu.ca

Wen Han Tang
wht6@sfu.ca

Jie Wang
jwa307@sfu.ca

1 Introduction

Shopping has always been important in modern people's daily life. However, it is very easy for people to feel confused when they are purchasing things, especially in big supercenters or markets. This would seriously reduce customers' shopping experience, affect the reputation and economic conditions of merchants, and overall cash flow within a country to run the economy.

The operators and managers in the industry of merchandise have tried to increase their capability for handling this issue, but sometimes this may lead to backfire. Considering a situation where a customer completely knows what needs to be done and doesn't want to talk to others or want to have their privacy in decision making, if in this case, an employee tries to offer help, then the customer might be unhappy. So, we tried to solve this problem using machine learning and predict when to interrupt the customer for help.

With the rapid development of artificial intelligence and neural networks, a lot of new ways have emerged to help solve difficult problems. For example, researchers have constructed models like Convolutional Neural Network for classifying images and also other architectures such as Recurrent Neural Network for natural language processing and other tasks. By utilizing the benefits of past works, we have constructed neural network models that can help identify confused customers with reasonable data we labeled ourselves.

2 Datasets and Methodology

The original data we had was the videos showing different people picking smartphones in shopping scenarios, all the videos are provided by our group's supervisor. Our objective is to, given a small video (which is a sequence of frames), the model should predict whether the video is displaying a customer being confused or not confused. Since we are having temporal information in the sequence of input data, we considered that using a Recurrent Neural Network (RNN) is the appropriate choice. The reason is that to classify a sequence of images to which class it belongs to we need the temporal knowledge stored from previous time steps to produce a more accurate decision. However, Convolutional Neural Network (CNN) has certain advantages on classification tasks and we want to leverage that to enhance our prediction accuracy, our models would be combinations of Convolutional Neural Network and Recurrent Neural Network.

There are different types of Recurrent Neural Network, and we chose to use Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Compared to simple RNN, both of them have controlled gates to determine what should be remembered from the previous time step and what should be ignored. In General, LSTM and GRU have similar performance in many scenarios and outperform simple RNN, GRU also holds a simpler structure than LSTM but LSTM can still outperform GRU in a few circumstances. Also, besides simple CNN, some pre-trained CNN based models like ResNet50 that are designed and trained specifically for image classification jobs are available. In order to pick a relatively well-performed model, we compared the performances among models that have different combinations of layers, our chosen basic combinations are:

a. CNN + LSTM

b. CNN + GRU

c. CNN + LSTM + GRU

d. ResNet50 (Pretrained Model) + LSTM + GRU

3 Data Preprocessing

In the beginning, we were required to convert each video from the original data to frames and assign each frame manually with four labels (classes) ('confused', '`not_confused`', 'uncertain', and 'none') by utilizing an annotation program which was also given by the supervisor. The output of the annotation program is directories with images from each video, and label file with respect to that video containing the dictionary of image indices and each image label.

We wrote the preprocessing function to separate the images into four folders with names confused, not confused, uncertain, and none as it is the required format for the Keras video generator. Since the name of the folders and the name inside the labels .json files were different as different people performed the annotation part and they used different tools or different naming structure, we had to spend quite a lot of time finding which all possibilities are there in the naming sequence. We remove the folder named 'none' which was not needed, and the remaining each folder's images are then converted to videos.

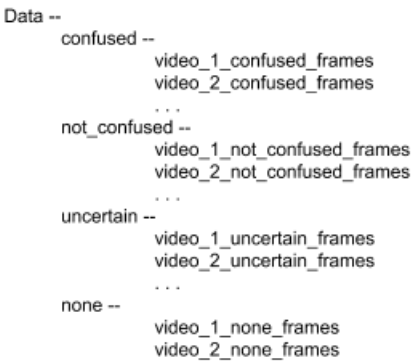

Figure 1: Directory Structure of The Data

In the problem we are trying to solve, we have temporal information stored in our video data. This temporal data is important for us as we are trying to predict the classes based on many frames which include past frames too. Models like CNN and RNN are useful in our case to first extract the features from the images and then understand the temporal information from them. Sequential models require timestamps to learn previous information and perform future calculations. In our case, the timestamps we have are the number of frames we want to send to the RNN cells sequentially. For this reason, we need a 5D tensor with (`batch_size`, timestamps, width, height, channels). Here, every batch sample has certain timestamps which means every batch sample has a few frames that are sent to the RNN. To perform this calculation, we used a custom library named Keras-video-generators which needs videos as its inputs (reference at [1]:).

After processing, the dataset is split into 392 files for train and 130 files for validation. The details are: for class confused, 57 files for validation, 171 files for training; for class `not_confused`, 73 files for validation, 221 files for training. (We will discuss why we were not using the class 'uncertain' in section 5 'Adjustment and Improvement')

4 Models

We decided to compare different basic models and tried to understand the differences in their behavior, and what factors are leading those differences.

4.1 CNN + LONG SHORT-TERM MEMORY (LSTM)

In the model of CNN+LSTM, the first layer should be a Conv2D layer that will deal with the input images (20, 224, 224, 3). After CNN processes the sequential image data, it will pass them to the LSTM layer, in which 10 units processed the data at the same time. Before using the softmax function to produce the final result, there is a dropout added to avoid overfitting problems. The table below shows how we tune the parameters:

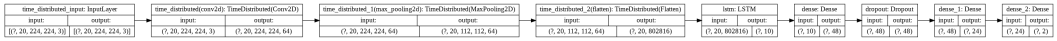

Figure 2: Plot of CNN + LSTM Model

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 20, 224, 224, 64)	1792
time_distributed_1 (TimeDist	(None, 20, 112, 112, 64)	0
time_distributed_2 (TimeDist	(None, 20, 802816)	0
lstm (LSTM)	(None, 10)	32113080
dense (Dense)	(None, 48)	528
dropout (Dropout)	(None, 48)	0
dense_1 (Dense)	(None, 24)	1176
dense_2 (Dense)	(None, 2)	50

Total params: 32,116,626
Trainable params: 32,116,626
Non-trainable params: 0

Figure 3: Summary of CNN + LSTM Model

4.2 CNN + GATED RECURRENT UNIT (GRU)

In the model of CNN+GRU, the first layer should be a Conv2D layer that will deal with the input images (20, 224, 224, 3). The only difference between CNN+LSTM and CNN+GRU is the middle layer. In GRU, we choose 10 as the hidden state which will control the GRU output units. From the table below, we can see that the GRU layer has fewer parameters than LSTM as expected.

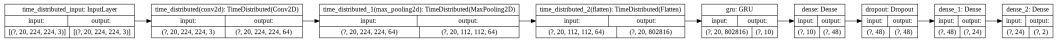

Figure 4: Plot of CNN + GRU Model

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 20, 224, 224, 64)	1792
time_distributed_1 (TimeDist	(None, 20, 112, 112, 64)	0
time_distributed_2 (TimeDist	(None, 20, 802816)	0
gru (GRU)	(None, 10)	24084840
dense (Dense)	(None, 48)	528
dropout (Dropout)	(None, 48)	0
dense_1 (Dense)	(None, 24)	1176
dense_2 (Dense)	(None, 2)	50

Total params: 24,088,386
Trainable params: 24,088,386
Non-trainable params: 0

Figure 5: Summary of CNN + GRU Model

4.3 CNN + LSTM + GRU

In the model, we try to combine LSTM and GRU to see if the performance could improve. The CNN layer is the same as the former two models. Then, it comes LSTM with 10 units first followed by GRU. The output layer is still the softmax function. The total parameters are much larger:

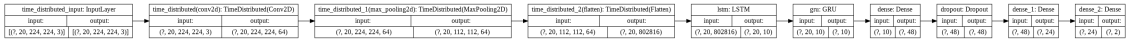

Figure 6: Plot of CNN + LSTM + GRU Model

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 20, 224, 224, 64)	1792
time_distributed_1 (TimeDist	(None, 20, 112, 112, 64)	0
time_distributed_2 (TimeDist	(None, 20, 802816)	0
lstm (LSTM)	(None, 20, 10)	32113080
gru (GRU)	(None, 10)	660
dense (Dense)	(None, 48)	528
dropout (Dropout)	(None, 48)	0
dense_1 (Dense)	(None, 24)	1176
dense_2 (Dense)	(None, 2)	50

Total params: 32,117,286
Trainable params: 32,117,286
Non-trainable params: 0

Figure 7: Summary of CNN + LSTM + GRU Model

4.4 ResNet50 + LSTM + GRU

In this model, we are using a pre-trained model ResNet50 just to see if it helps in any way. The middle layer is 10 units LSTM plus 10 units GRU, which keeps the same as the former models. The parameters table shows below: (We are not showing the plot figure because this figure would contain all the layers in ResNet50 and we can not fit this diagram in our report)

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 20, 2048)	23564800
time_distributed_1 (TimeDist	(None, 20, 2048)	0
lstm (LSTM)	(None, 20, 10)	82360
gru (GRU)	(None, 10)	660
dense (Dense)	(None, 48)	528
dropout (Dropout)	(None, 48)	0
dense_1 (Dense)	(None, 24)	1176
dense_2 (Dense)	(None, 2)	50

Total params: 23,649,574
Trainable params: 23,604,134
Non-trainable params: 45,440

Figure 8: Summary of ResNet50 + LSTM + GRU Model

5 Adjustment and Improvement

Since we spent a lot of time trying to pre-process the data that helped us to understand the data in depth. We made a few adjustments like the number of classes, hyper-parameters, and layer units with respect to the number of frames. We believe that there are more areas for improvement.

- **Hyper-parameter Tuning**: The split rate is 75% for training and 25% for validation data. We used the Keras-tuner package to perform Random Search. We are able to find the best hyper-parameters with search space as shown in fig 4.0. We trained 66 times with different sets of parameter combinations, and each iteration with 10 epochs. We only search for activation, dropout rate, learning rate, and decay rate. This helped us in finding the best hyperparameters for each model structure. We could have included more parameters in our search space like the number of layers, no. of neurons or cells in each layer, but due to time limits, we are not able to perform that as it significantly increases the time of the training.

```
Search space summary
Default search space size: 4
activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
dropout_rate (Float)
{'default': 0.1, 'conditions': [], 'min_value': 0.1, 'max_value': 0.9, 'step': 0.2, 'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
decay_rate (Choice)
{'default': 0.9, 'conditions': [], 'values': [0.9, 0.6, 0.3, 0.1], 'ordered': True}
```

Figure 9: Search Space Summary of Hyper-Parameter

- **Bias in the Annotations**: Different people performed annotations and each person has their standards when deciding whether the person is confused or not. This leads to the problem of labeling bias and variance. High bias and high variance are not good for the model as they make the data points scattered and increase the distance between predicted data points and the target data points. We believe that we have a high bias and variance in our model. One way we tried to solve this approach is to decrease classes from 3 to 2. With only two classes confused and non-confused, we are decreasing the bias and variance by reducing uncertain class which is just making the model more confused. The results we got are better than what we got in 3 classes. From 38% with 3 classes, we got to 72% with 2 classes.
- **Pose Estimation**: Since we are using CNN to extract spatial information first and then pass it to the sequential models to extract temporal information. We believe that using the post estimation model first to extract the body coordinates in 3D using this paper and localization of human joints can help a lot in our problem. First, we will use a pose estimation model and make another model of CNN and merge/concatenate these two models together to a sequential model to extract the temporal information. This technique can help a lot as we

are passing images with body coordinates and localization of human joints which means only the required information.

6 Results

We tested twice to compare the results with and output the hyper-parameter tuning. The first test was with just one trial which means 1 training without best hyperparameters. For the second test we performed 66 trials and saved the model with the best hyperparameters. We then performed the training again with those best parameters. The graphs are smoothed by the factor of .6.

6.1 CNN + LONG SHORT-TERM MEMORY (LSTM)

The performance of the first test and the second test is similar, so we only show the evaluation of the second test. From the diagram below, we can see the training (orange) accuracy grow from 0.45 to 0.55 basically as the epoch increases, and the validation (blue) accuracy rises with the same trend as the train but performs better. The loss of the training and the validation both decreases but validation looks more stable.

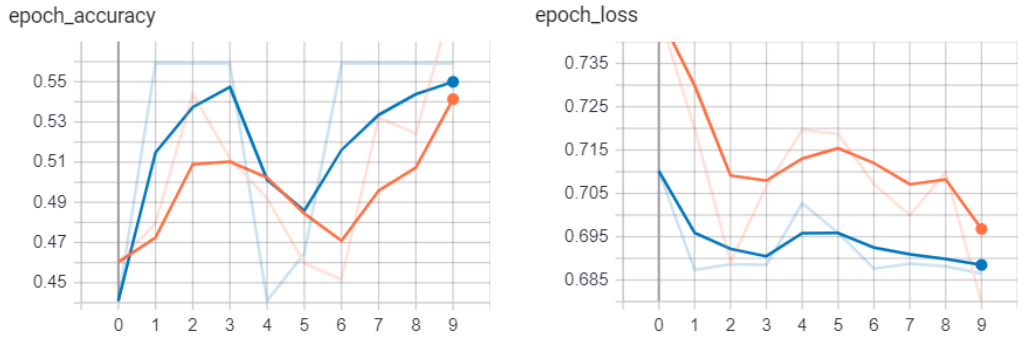

Figure 10: Accuracy and Loss vs No. of Epochs

6.2 CNN + GATED RECURRENT UNIT (GRU)

First test: For the first test with 10 epochs in 1 trial, we can see the performance of CNN+GRU is better than LSTM, both the training and validation accuracy increases with the epoch rising. The training loss decreases obviously while the validation loss decreases more stably.

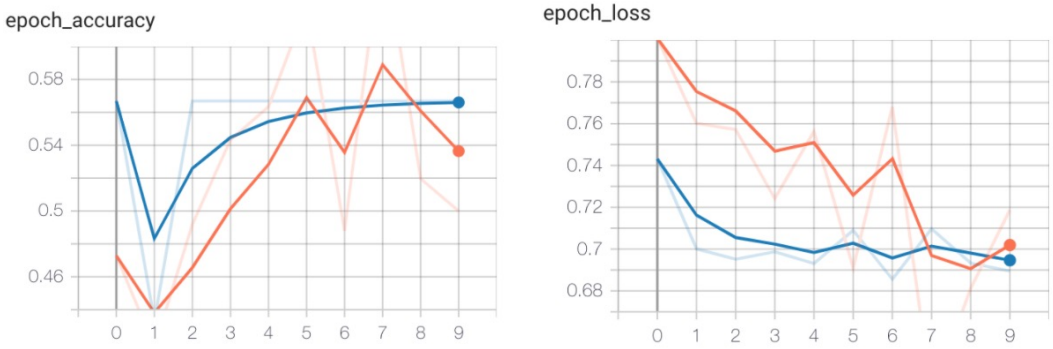

Figure 11: Accuracy and Loss vs No. of Epochs

Second test: For best hyper-parameters in 66 trials (10 epochs per trail), the training accuracy and loss is decreasing but the validation accuracy is not decreasing, it is the same for the entire time.

Validation loss is decreasing but not much. Also, after epoch 4, the training loss is starting to increase which shows that model is overfitted. But again at the same time we can say that validation accuracy is decreasing, so we leave this for discussion.

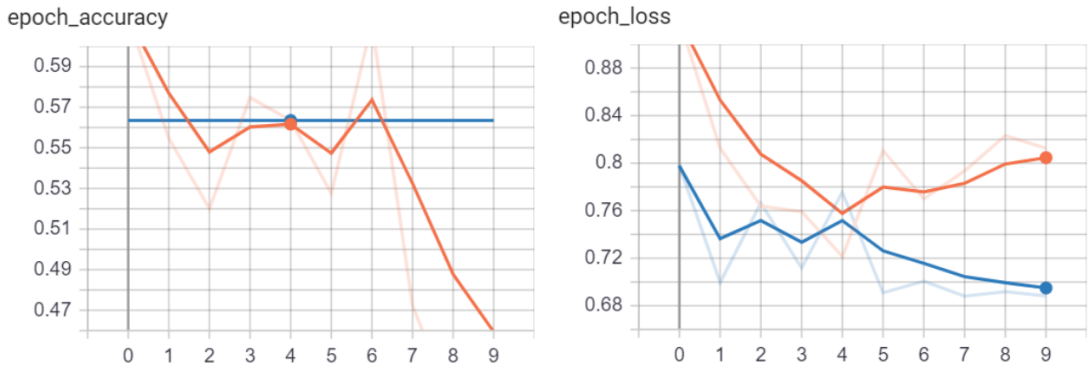

Figure 12: Accuracy and Loss vs No. of Epochs

6.3 CNN + LSTM + GRU

We only show the second test because the result of the first and the second is very close. We can see there is overfit problem from the diagram because the accuracy of train rises but the validation does not. The same trend can be seen in loss diagram. The loss of validation has no obvious reduce.

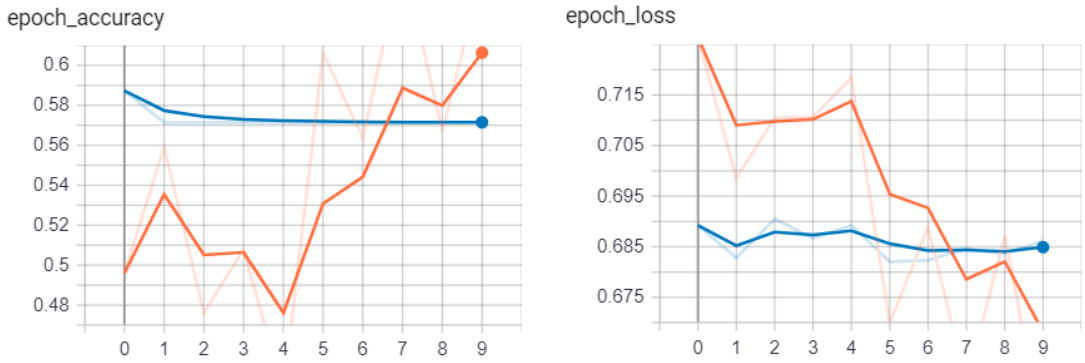

Figure 13: Accuracy and Loss vs No. of Epochs

6.4 ResNet + LSTM + GRU

First test: For the first test with 10 epochs for 1 trial, the training accuracy increases up to 0.72 from 0.48, which is a big improvement. While the validation accuracy does not increase as expected. We can and cannot say there is overfit, this depends on many things or maybe we have to see more epochs. However, the loss of both training and validation decreases almost in sync.

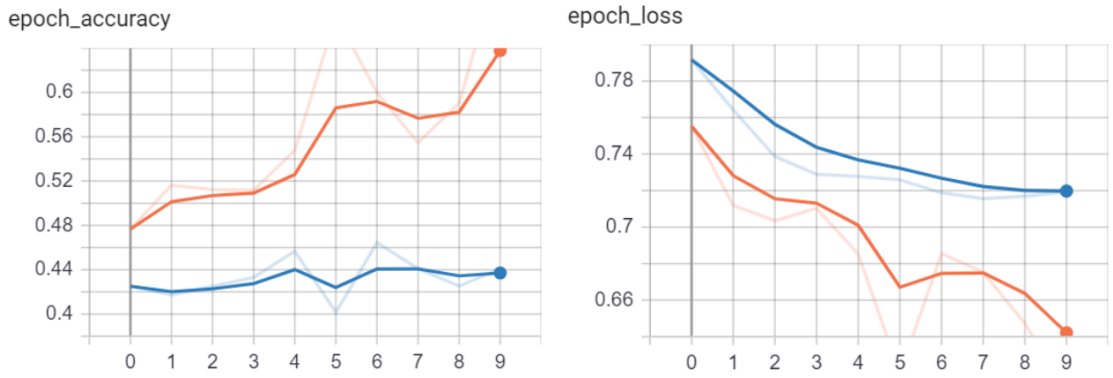

Figure 14: Accuracy and Loss vs No. of Epochs

Second test: For best hyper-parameters in 66 trials (5 epochs per trail instead of 10, because ResNet50v2 took a lot of time to train). The training accuracy is going from .44 to .58 which is good because there is no big fall in 5 epochs. We can say that using the pre-trained weights are useful for us but again we are a little confused as we got .72 accuracy in the first test with just 1 trial. Overall in general, pre-trained weights are helpful.

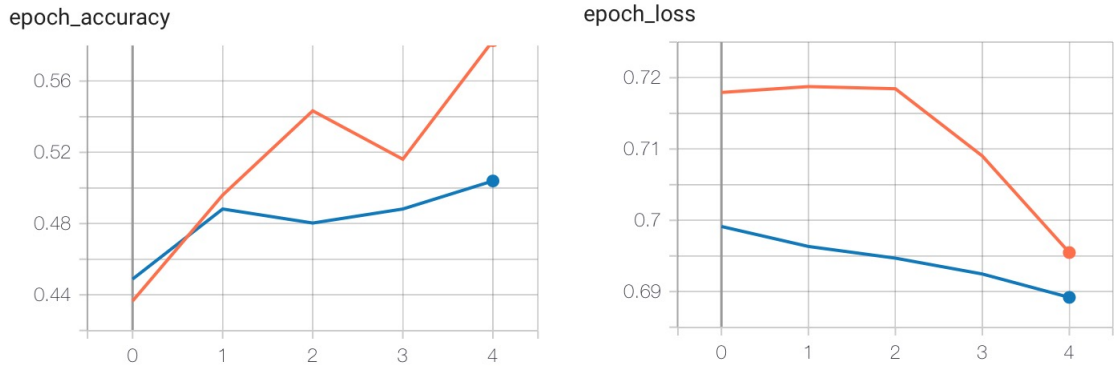

Figure 15: Accuracy and Loss vs No. of Epochs

7 Conclusion

The best model we have seen so far is the ResNet50 + LSTM + GRU which has the best results with the possibility of going up to 72 percent with two classes. Also, it does not have sudden big falls and jumps. This shows that pre-trained models are helpful in our case and pose estimation techniques would be useful to our problem if we can apply it. We believe that the models we are using with specific structure are well researched by us given the factors within our knowledge and time.

8 Contributions and Link to the Github Repository

Research: Akash Sindhu, Wen Han Tang, Jie Wang
Data Preprocessing: Akash Sindhu, Wen Han Tang, Jie Wang
Models Selection and Implementation: Akash Sindhu, Wen Han Tang, Jie Wang
Optimization: Akash Sindhu, Wen Han Tang, Jie Wang
Report Writing: Akash Sindhu, Wen Han Tang, Jie Wang

Link to the Github Repository:

`https://github.com/Akashsindhu/when_to_interupt`

9 Reference

[1]Ferlet, P. (2019, December 05). Training neural network with image sequence, an example with video as input. Retrieved December 17, 2020, from https://medium.com/smileinnovation/training-neural-network-with-image-sequence-an-example-with-video-as-input-c3407f7a0b0f

[2]Olah, C. (n.d.). Understanding LSTM Networks. Retrieved December 17, 2020, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3]Pavllo, D., Feichtenhofer, C., Grangier, D., Auli, M. (2019, March 29). 3D human pose estimation in video with temporal convolutions and semi-supervised training. Retrieved December 17, 2020, from https://arxiv.org/abs/1811.11742

[4]Sun, J., Wang, J., Yeh, T. (n.d.). Video Understanding: From Video Classification to Captioning. Retrieved December 17, 2020, from http://cs231n.stanford.edu/reports/2017/pdfs/709.pdf