Effect Prediction using Visual Attention

Yantian Zha
ID: 1209792711
yzha3@asu.edu

Xiaoyu Zhang
ID: 1207639026
xzhan320@asu.edu

Zongsi Zhang
ID: 1209360655
zzhan236@asu.edu

Jianmei Ye
ID: 1207989038
jianmeiy@asu.edu

Mengyuan Zhang
ID: 1209583385
mzhan123@asu.edu

ABSTRACT

We propose an Effect-Prediction system whose architecture is a CNN-LSTM-LSTM cascaded neural network. The system is used to predict the world state that is triggered by a human action. We adopt the "Inception" GoogLeNet as the CNN part to extract a feature map, and two attention-based LSTMs are introduced as the fundamental units of action recognition and effect prediction. We describe our architecture by visualizing how the system pays attention to video frames, and show how accurate our model is. We make our own dataset EP3A for effect prediction tasks. Please download our dataset at: https://www.dropbox.com/s/es6ipjv5u6ikysr/EP3A.zip?dl=0

1. Introduction

In industry, robots are still unable to replace human labor completely, when encountering jobs that involve high-technological production, or safety issues. Therefore, it is crucial to make robots collaborate with human workers in a smarter, and safer way. In household situations, assisting human housekeeping are the core tasks of robots. Robots that are made for those human collaboration tasks are called co-bots. In order to design a robust software brain for co-bots, dozens of AI directions are needed to explore. It is remarkable that the emerging robotics industry has marched into an era where vision-enhanced robot systems become far more intelligent and reliable than decades ago. In this project we try to find out how to use the matured computer vision technologies to make co-bot a better planner.

In the human-robot collaboration field, it is important that robots can predict the state triggered by an action (we call it *effect*), when observing a human motion. In Artificial Intelligence, the robot needs to accomplish a task by executing a plan. In some cases, the task a robot needs to do relies on prerequisite states triggered by another human agent. While such states are possibly invisible, the robot should then be able to predict them when observing the action motion. An example that helps illustrate this scenario, the robot plans to carry a box with a cup inside to another location (as shown in Figure 1.1). A human helps the robot by placing the cup into the box. From the robot's visual angle, it cannot directly observe whether the human placed the cup into the box, or behind the box. Fortunately, by observing the features of human's putting-down motion, the robot can guess the answer. The robot predicts a state, upon which the following plan execution is based.

Figure 1.1 Robot taking next-step action based on prediction of current state

In other cases, human and robot collaborate with each other to reach to a mutual goal, which implies that, either robot or human are not competent enough to accomplish the task alone. This further implies that, when observing a human taking action, the robot needs to model the intention of human, and generate a belief of future states. Once obtaining the belief, the robot needs to make up a plan to assist human. For example, a human is trying to place a cube into the box which is out of reach of him (as shown in Figure 1.2). In order to better assist the human accomplish this goal, the robot then generates a plan. By predicting the intention that the human wants to put down an object into the box, and by computing the future state where the human is able to do that, the robot moves forward and pushes the box closer to human.

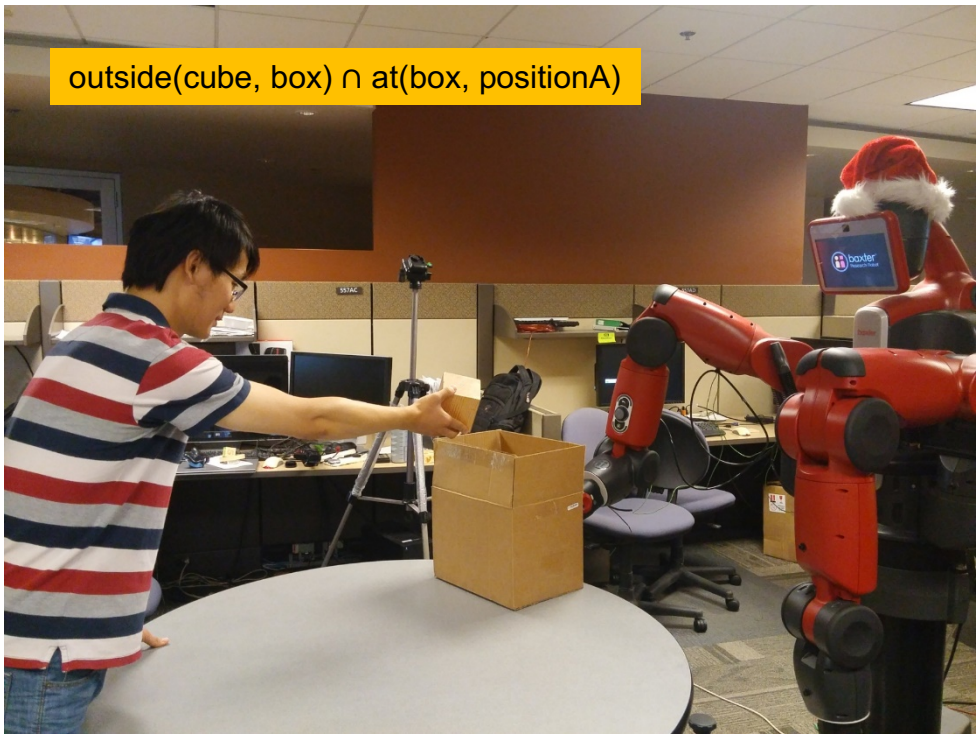


Figure 1.2 Human-robot collaborate to reach to the same "future" goal

Since there has been very little work on effects prediction, in this paper, we propose our approaches to solve this problem. The main contribution of this work are the following:

- We introduce a three-cascaded neural networks framework as the effect prediction system (Section 3.1).
- We create an effect prediction dataset which consists of RGB-D videos recorded by the MS-Kinect V1.

2. Background

In this section we provide background knowledge from three aspects on which our effect prediction system based: *computer vision*, *machine learning*, and *planning*.

2.1 Convolutional neural networks

In most recent years, deep neural networks have significantly scaled up the capacity of computer vision systems. The emergence of Convolutional Neural Networks (CNN) improves image recognition performance [1], which is a classification problem. In addition, CNN are proven to perform well in solving regression problems like pose recovery [2] or human pose estimation [3]. The reason for the widely usage of CNN might be in large part that, CNN are good at modelling spatial relationship among elements in a vector (e.g. a sentence) or a matrix (e.g. an image).

Here comes the structure of convolutional neural networks which use a parameter sharing technique. This suggests that, the size of the hidden layer is limited while we expect it to be unlimitedly large which increase the power of model. However, in the hidden layer, if some group of neurons can self-replicate themselves, this problem would be solved. Since by self-replicating, a neuron can be applied in many locations of the feature map from the previous layer. Specifically,

each upper layer neurons (e.g. **A** in Figure 2.1.1), takes care of a group of neurons from the lower layer that adjacent to each other.

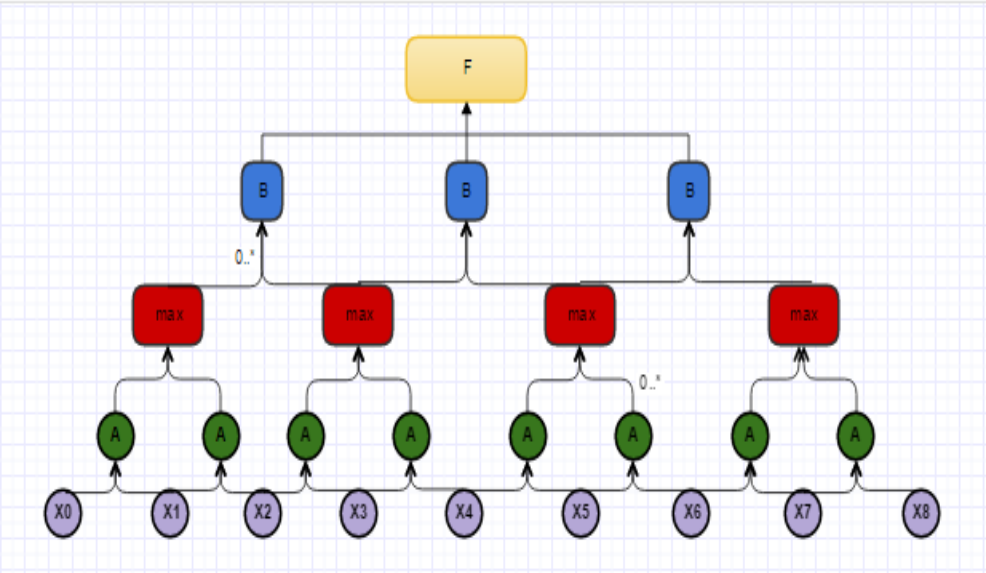


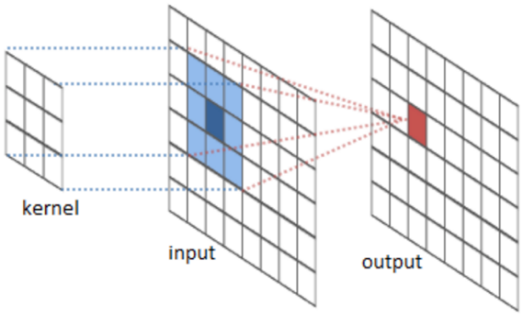
Figure 2.1.1 Convolutional Neural Network structure

In Figure 2.1.1, **F** is called a fully connected layer. The layers below **F** are called convolutional layers. Taking **A** as an example, since **A** takes care of two input units from the previous layer, the window size of **A** is 2. The output of hidden unit A comes from a convolution computation, with X0 and X1. The convolution is denoted as the Equation 2.1.2:

$$[f * g](t) = \int_0^t f(\tau)g(t-\tau)d\tau, \tau \in [0,\ t]$$

Equation 2.1.2 General convolution

As for convolution equation in CNN domain, f is typically the input, g is the kernel, a multidimensional array of parameters, which slides from one location to another location, to calculate the output (feature map). The training of CNN involves adapting the parameters in the kernel.



From the River Trail documentation

Figure 2.1.3 Feature map in CNN

The “max” part in Figure 2.1.3 is a max pooling layer. Pooling layer is used to “zoom out” the output from its previous layer. Instead of using the original points from “output”, pooling layer computes statistics between a certain point and its surrounding points, and then finds a representation of all of its neighborhood points.

2.2 Long short term memory neural networks

In order to model the relationship between each time-step, that is, temporal relationship, deep learning researchers added a feedback structure (Figure 2.2.1) to the traditional artificial neurons. This is the so-called Recurrent Neural Networks [4]. Figure 2.2.1 shows the feedback from the output neuron in hidden layer to the input part:

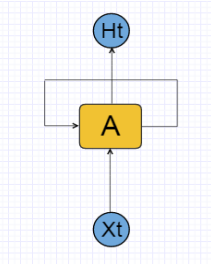


Figure 2.2.1 Recurrent Neural Network loops

If the inputs are sequential, like a sequence of words in a sentence, the relationship between inputs at each time-step, with the RNN layer, can be modelled in Figure 2.2.2. This explains why RNN obtains the memory ability.

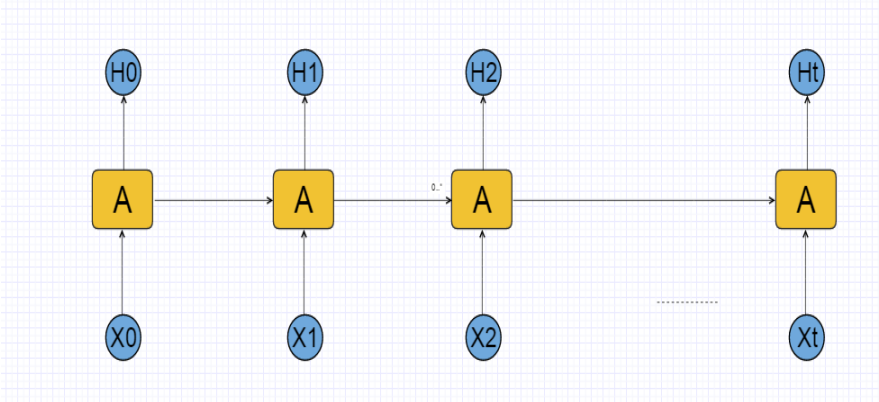


Figure 2.2.2 An unrolled recurrent neural network

However, RNN is not powerful enough to carry information at original resolution over long temporal distances [5], due to how RNN handles with long-term dependencies (RNN tries to remember everything instead of learning from them). To solve this problem, RNN needs to be able to actively forget things at some stage. This leads to the emergence of Long Short Term Memory (LSTM) neural network. Figure 2.2.3 depicts the architecture of a LSTM:

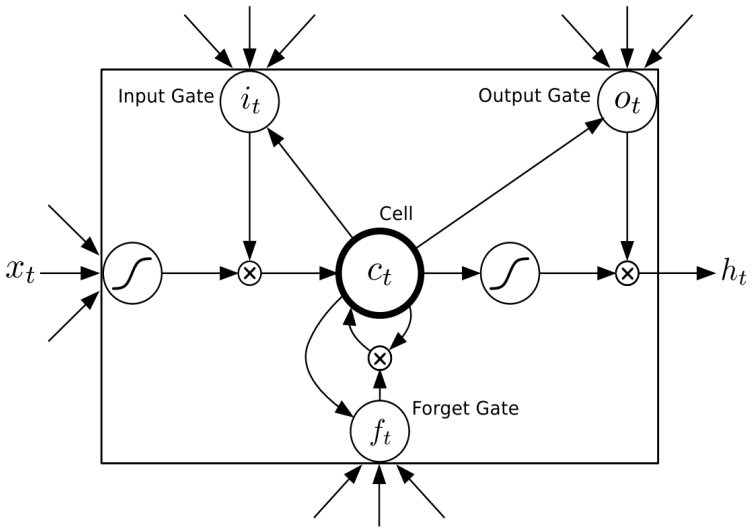


Figure 2.2.3 The LSTM architecture

In this work we use the LSTM implementation from [6] and [7]. For an input x_t at time step t, the LSTM unit computes an internal memory cell state c_t in addition to the hidden state h_t of an RNN. The hidden state h_t is computed by Equation 2.2.4:

$$h_t = o_t \odot \tanh(c_t),$$

Equation 2.2.4 The hidden state of LSTM at time t

where $\odot$ is the element-wise multiplication of the memory state (taking the hyperbolic tangent non-linearity) with the gate value. The cell state c_t is computed as a weighted summation of the previous memory cell state c_{t-1} and the new memory content update g_t by Equation 2.2.5:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t,$$

Equation 2.2.5 The internal memory cell state of LSTM at time t

where the two parameters are called forget gate f_t and input gate i_t.

The input gate i_t, forget gate f_t, output gate o_t, and updated memory content g_t at time-step t are given by Equation 2.2.6:

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} M\begin{pmatrix} h_{t-1}, \\ X_t \end{pmatrix},$$

Equation 2.2.4 Everything the cell has observed until time t

where σ is the logistical sigmoidal non-linearity.

2.3 Planning in Artificial Intelligence

A planning system consists of a domain model and a problem model. A state is a subset of facts that have are "True". The domain model stores the dynamics of the world. For example, the facts, the possible actions.

Definition1: an action is a tuple of (pre(a), add(a), del(a)). pre(a) is a set of preconditions, which must be satisfied firstly. The execution of action a triggers the word to move into a new state, by "adding some states to the pre(a)", and "deleting some states from pre(a)". This new state triggered by an action is called "effect", which is the output of our system.

3. Effect-Prediction System and approach

3.1 System overview

The Effect-Prediction system is a CNN-LSTM-LSTM cascaded neural network architecture, shown as Figure 3.1.1.

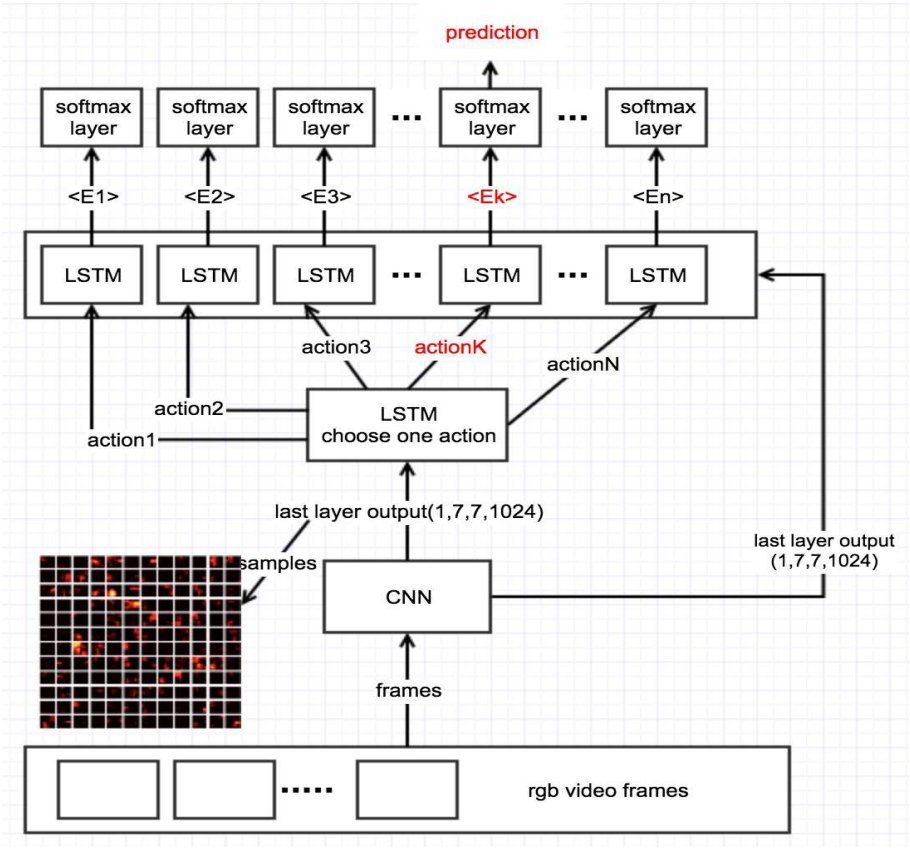


Figure 3.1.1 Effect-Prediction System

Figure 3.1.1 shows our Effect-Prediction system is a CNN-LSTM-LSTM cascaded network.

3.2 Dataset creation

We use MS-Kinect V1, the RGB-D sensor to create our own dataset.

To better catch the actions, we set FPS=60 for the sensor. The normalized length of frame sequence is 4. To fit our model, we remove the last effect frames while generate a texture label for each action instead. The key point is to train a model to predict the effect like a human.

We create three daily actions of *throw*, *pour*, *put down* with seven effect predicts of *inside*, *outside*, *on*, *left*, *right*, *front* and *back*. Figure 3.2.1 is an example of frames for the three actions.

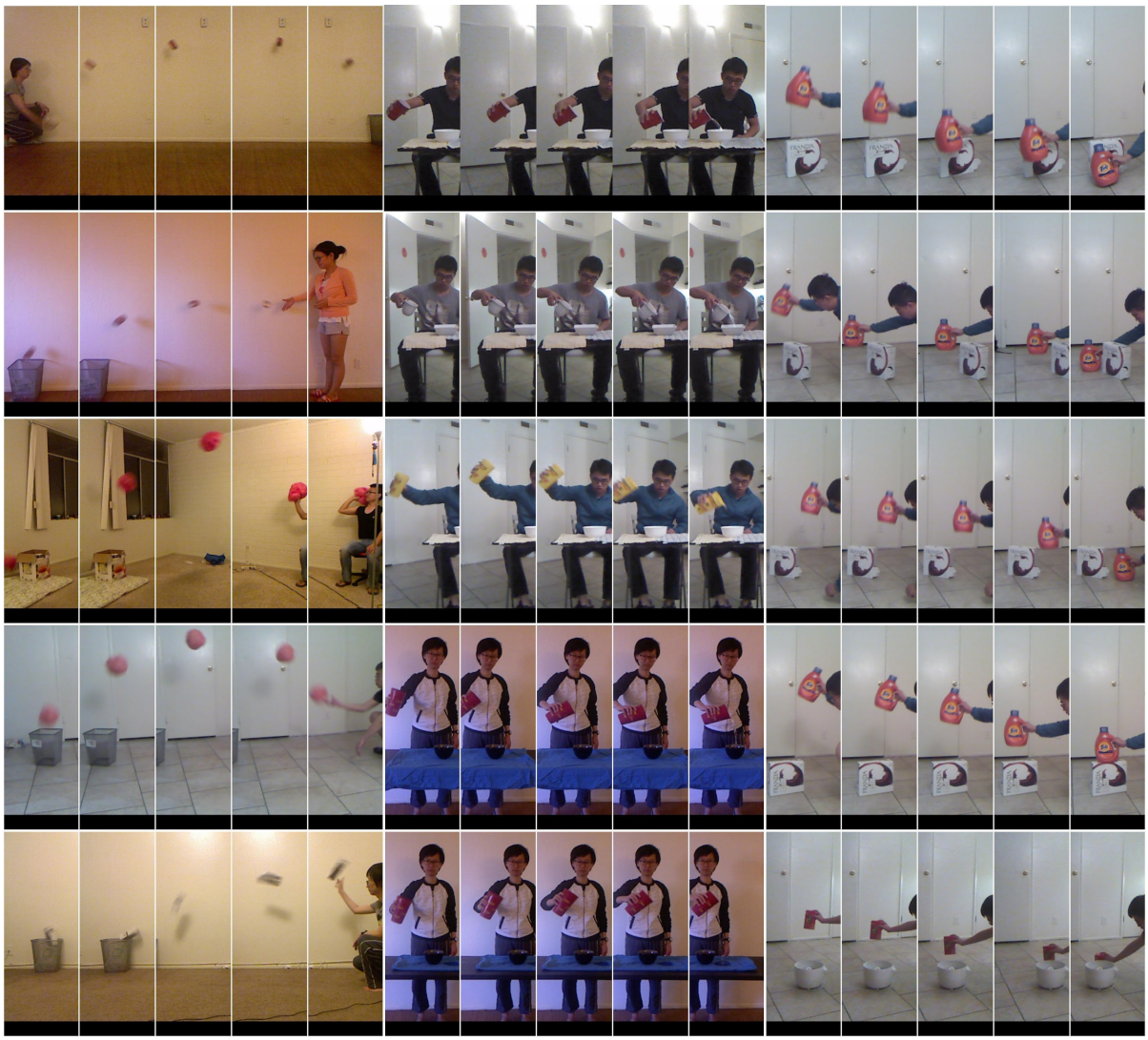


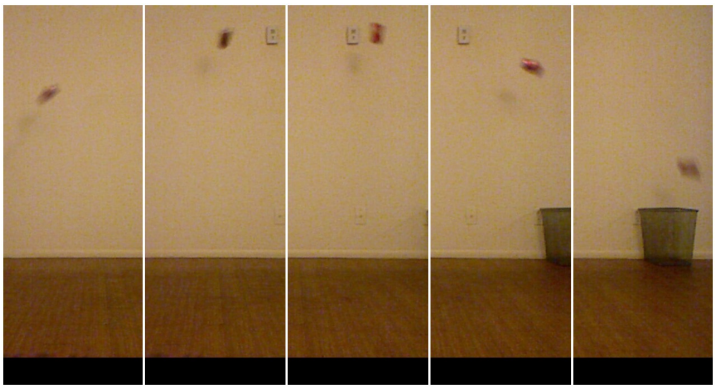
Figure 3.2.1 Example frames for the three actions.

Since what our system does is to predict the effect when observing a human action, we need to remove those frames that contains effect states in each of our video. But a question is: how many frames should we remove?

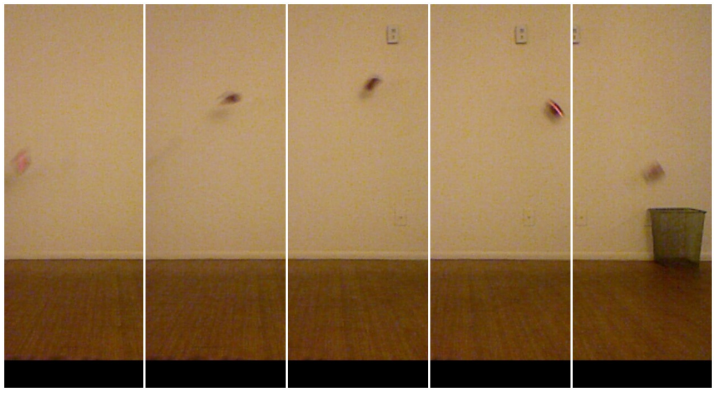
An interesting part of deep learning is that model can make decision like a human when distinguishing some corner cases.

For example, when we're trying to throw an object into a trash bin, if the input frame is recorded as the object far away from the bin or almost touch the ground, even the object is still in the air, it's very easy to tell from our human eyes that the object will land outside the trash bin, so does our model.

Figure 3.2.2 shows two trials of throw action with different track and result in different effects. Our model can tell the final effect for both of the tracking effect that **A** is outside and **B** is inside, just like our human eyes.



A-Label: outside



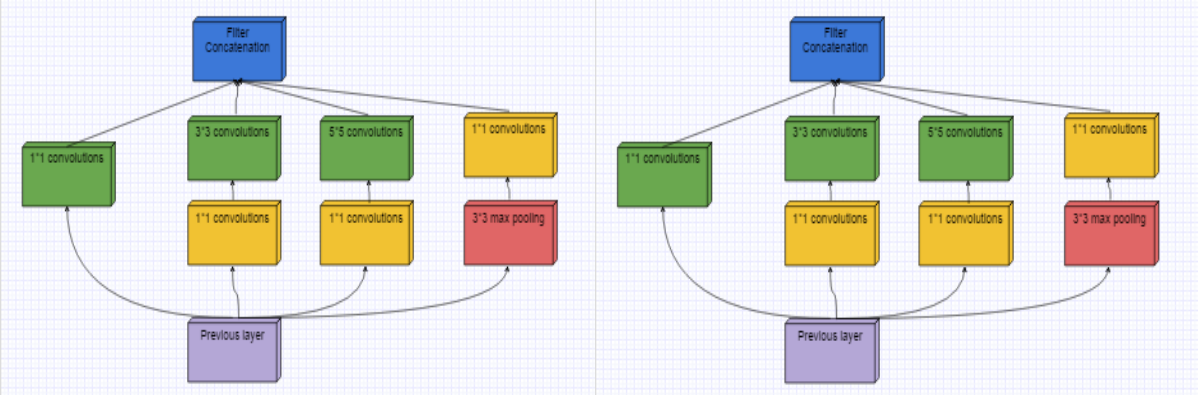
B-Label: inside

Figure 3.2.2: “throw” action for label *outside* and *inside*

3.3 Training through GoogLeNet

After building the dataset, we need to do the pre-processing work on the video frames. The goal is to complete feature detection and output corresponding feature maps as part of the input of LSTM for effect prediction. To do this we select a deep convolutional neural network architecture codenamed Inception [8] which also known as GoogLeNet.

Inception is a high efficiency CNN structure for image recognition and object detection. The core idea of Inception is to approximate and cover optimal local sparse structure in a convolutional vision network with readily available dense components. The Inception module is shown in Figure 3.3.1. In lower layer, when units are grouped into filter banks, we would end up with a lot of clusters concentrated into a single region that is covered by 1x1 convolutions in the higher layer. To solve this problem, inception uses larger patches for convolution to get a smaller number of spatially spread out clusters.



(a) Naive version (b) Module with dimension reduction

Figure 3.3.1: modules of Inception Module

The overall Inception GoogLeNet architecture can be visualized as Figure 3.3.2:

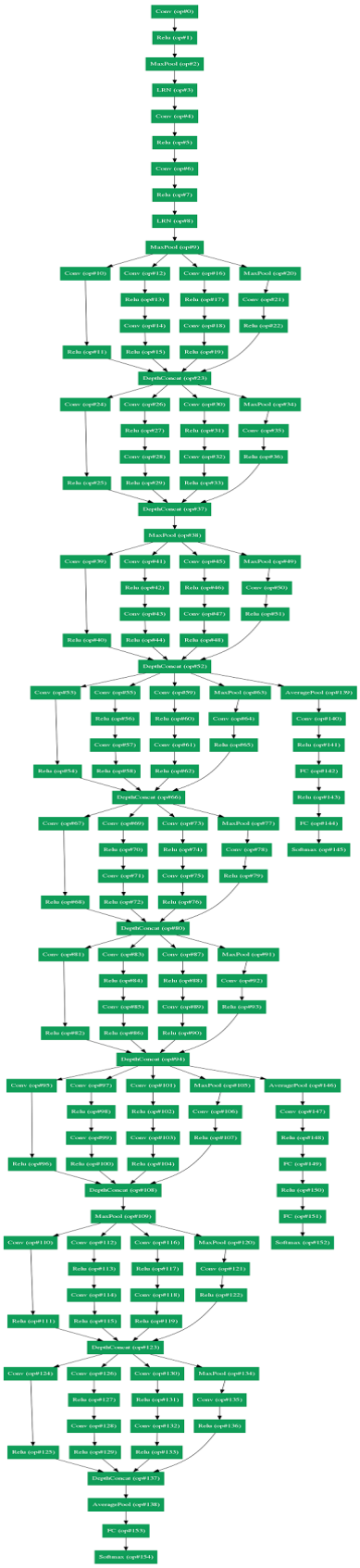
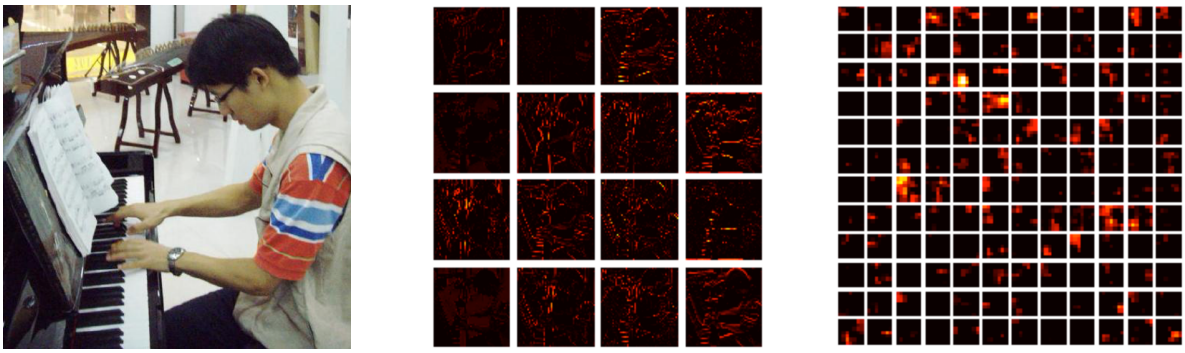


Figure 3.3.2 GoogLeNet network full architecture

We first try to train and test the entire system with RGB images only. Then the input of GoogLeNet is the 640*480 raw image. To train the model for every action we use image sets with frames of effects removed. To train the inception module, we use our scripts to translate the raw images into h5 formats and use them as the formative input for inception. Data is arranged in N*640*480*3 where N is dimension where every actions listed in order. Labels corresponded with every action is listed in another file in the same order.

Our model is trained with a dummy class at index 0, which never gets used. This is in order to be consistent with the Matlab indexing convention, which starts with 1. The total number of predictions is actually 1008 classes. we used 1008 instead of 1001 just to make numerical optimization better for historical reasons. In practice, only the indices 1 to 1000 are going to have nontrivial predictions, since class 0 and 1001-1007 never have positive examples during training.

There are generally two layers of convolution in our model. First layer uses 64 filters and yields the feature map of 112*112*64. The second layer uses 1024 filters and finally outputs the feature map of 7*7*1024, which will be used as input in LSTMs. Also, Inception can output the prediction results of object detection. After normalizing the result with softmax, the prediction results will be in the format of probability. For example, as is shown in Figure 3.3.3. The top prediction is "334 (prob. 0.1547) synset n04515003 upright, upright piano", which successfully detected the piano in the raw image. In conclusion, the output of our ConvNet performs well in object detection.



Raw image example First layer output(112*112*64) Last layer output(7*7*1024)

Figure 3.3.3 Sample Output of Inception

3.4 LSTM with Soft Attention Mechanism

We use the attention-based LSTM framework from the work of [9], as described in Figure 3.4.1. The action recognizer (the second level) and effect predictor (the third level) of our effect-prediction are constructed from the attention-based LSTM units.

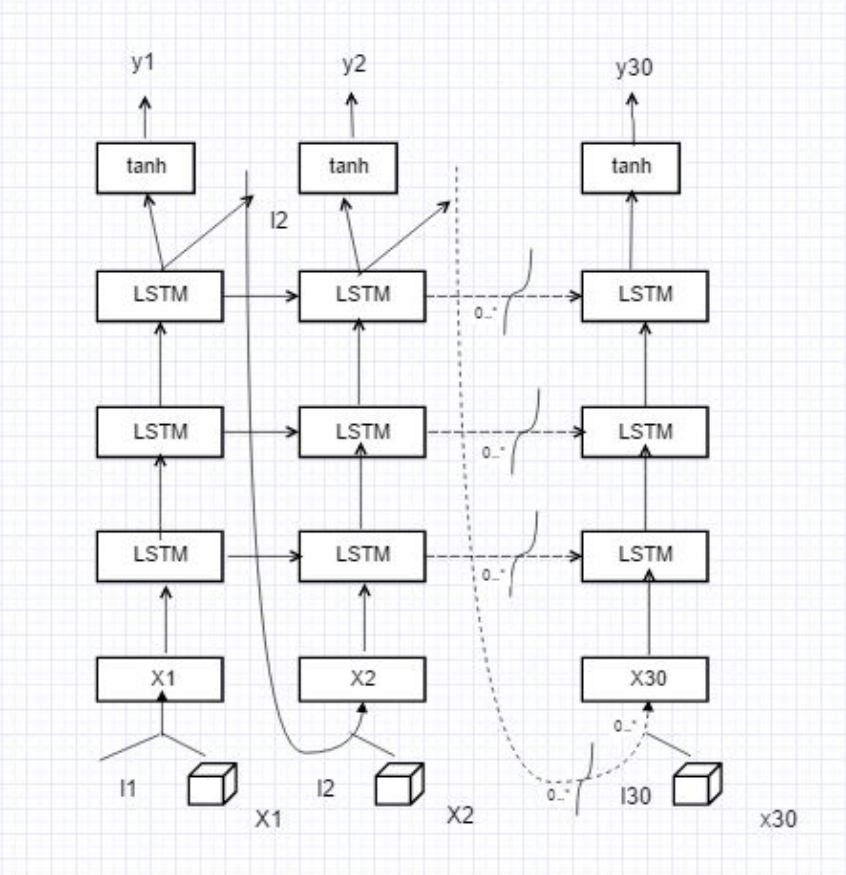


Figure 3.4.1 Attention-based LSTM framework

The attention-LSTM unit takes the input of 7*7*1024 feature map from previous CNN level, and outputs a prediction $\hat{y}$. The prediction, combined with the label y, are used by loss function (see Equation 3.3) to update the weights. In addition to that, the last layer of LSTM computes the attention softmax, which can be seen as the energy distribution in an energy model. This distribution changes the energy of different parts of input at next time-step.

$$l_{t,i} = p(L_t = i|h_{t-1}) = \frac{\exp\,(W_i^T h_{t-1})}{\sum_{j=1}^{K\times K} \exp\,(W_i^T h_{t-1})},\ i \in 1 \ldots K^2$$

Equation 3.1

The energy distribution is firstly computed by finding the confidence of each location (suppose that there are totally K*K locations) at the next-time, given the current-time state.

$$x_t = \mathbb{E}_{p(L_t|h_{t-1})}[X_t] = \sum_{j=1}^{K\times K} l_{t,i}\, X_{t,i}$$

Equation 3.2

Then the system calculates the overall distribution by finding the expectation of input at the next-time, with the confidence distribution.

$$L = -\sum_{t=1}^{T}\sum_{i=1}^{C} y_{t,i}\log \hat{y}_{t,i} + \lambda \sum_{i=1}^{K^2}(1 - \sum_{t=1}^{T} l_{t,i})^2 + \gamma \sum_{i}\sum_{j}\theta_{i,j}^2,$$

Equation 3.3

4. Experiments

We have created our own effect-prediction dataset to do the evaluation. The dataset directory structure is described as Figure 4.1:

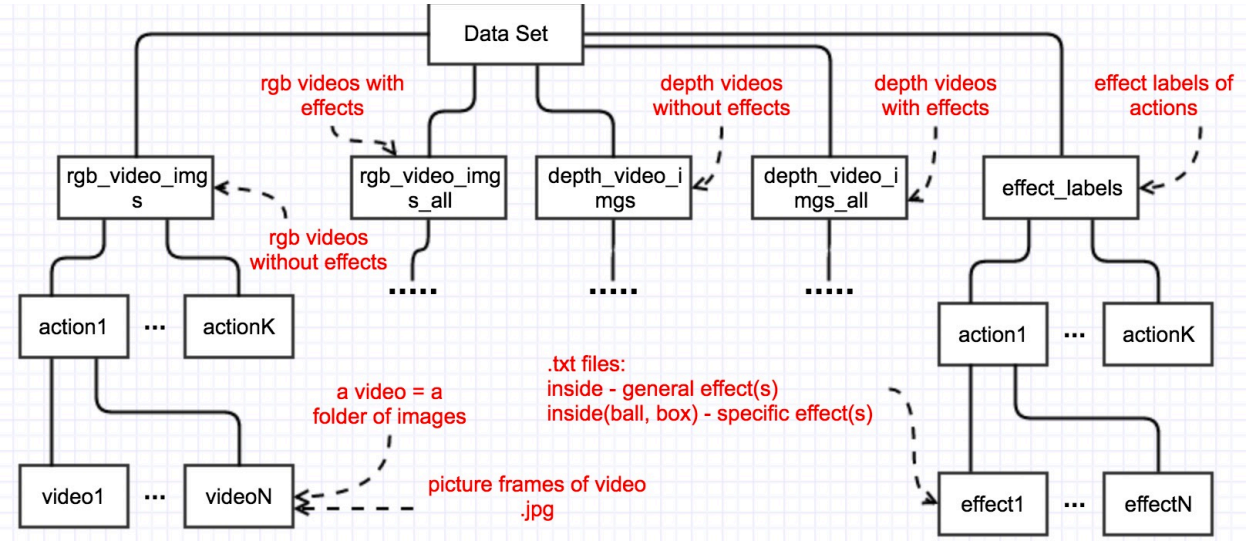


Figure 4.1 Full dataset directory structure

Our dataset contains both RGB/Depth images. But now we only use the RGB images (folder rgb_video_imgs). The dataset has three actions that have multiple effects: *throw*, *put_down*, and *pour*. *Throw* has the effects of "inside" and "outside"; *Put_down* has the effects of "on", "inside", "right", "left", "front", "back"; *Pour* has the effects of "inside" and "outside".

The Figure 4.2 is the visualization output from our action recognizer, with the action "throw". It is correctly trying to focus on the human and the trash bin.

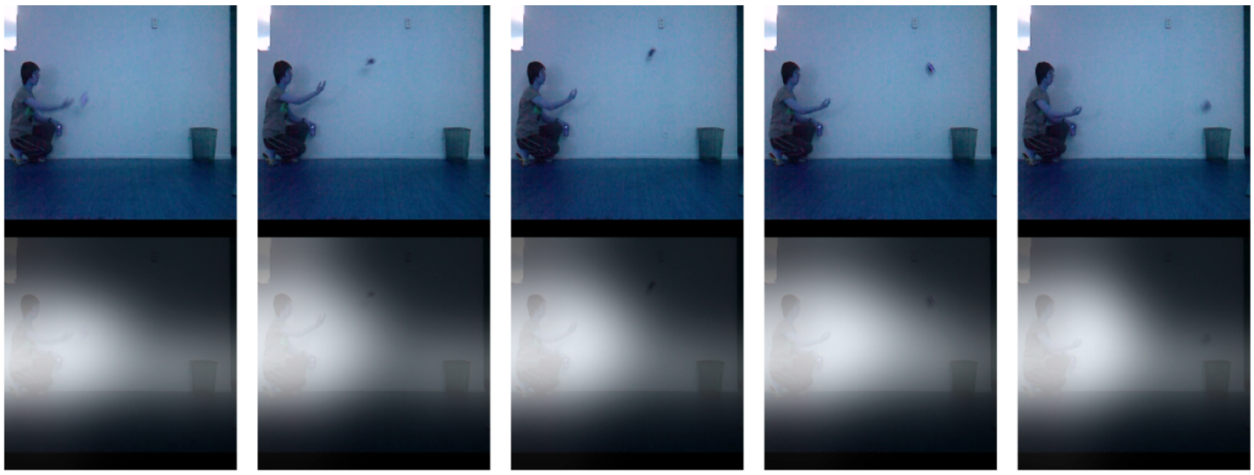
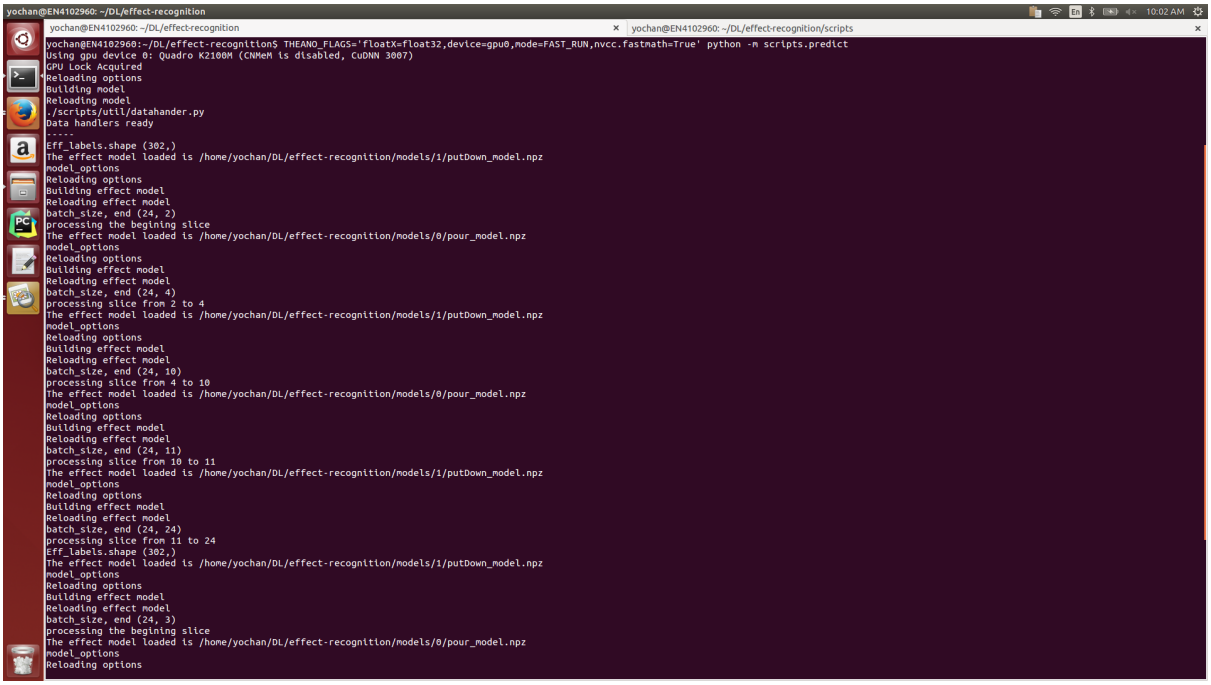


Figure 4.2 Visualization of “throw” action recognition

The Figure 4.3 is the detailed implementation of our effect-prediction system. Both the second level (action recognizer) and the third level (effect predictor) of our system use the same batch size (24), and once the action label (the output of second level) is computed, the system assigns an effect predictor model, takes the feature map as the input, and outputs an effect label.



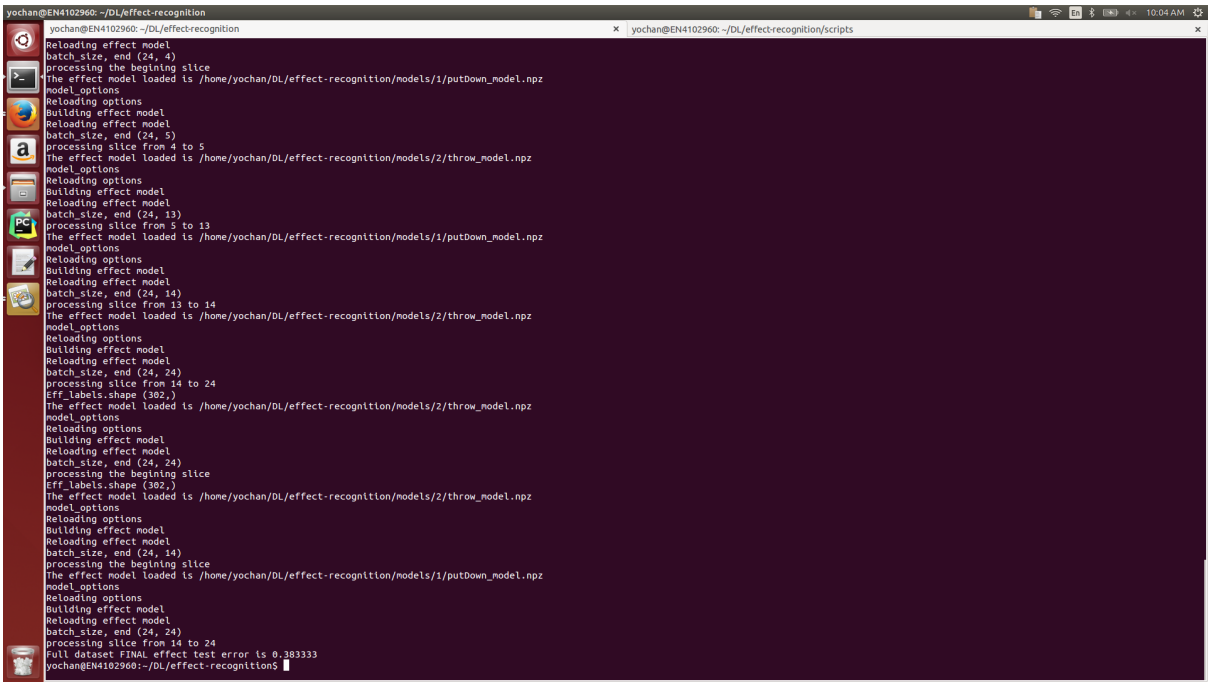


Figure 4.3 Effect-prediction system implementation

5. Results

The Table 5.1 is the overall statistics of accuracy from each level of model in our system:

Modules	Accuracy_Final_Train	Accuracy_Final_Test	Num_outputs	Max_epochs
Action	100%	41.11%	3 actions	15
Effects_Put_Down	99.19%	46.15%	7 effects	15
Effects_Pour	97.5%	66.67%	2 effects	15
Effects_Throw	66.07%	42.86%	2 effects	15
E-P system testing		31.11%	3 actions, 7 effects	

Table 5.1 Overall accuracy of the system

Test_final: 90 videos, 712 frames.

Train_action: 436 videos, 3399 frames.

Train_effect_throw: 112 videos, 690 frames.

Train_effect_put_down: 126 videos, 1102 frames.

Train_effect_pour: 200 videos, 1605 frames.

Test_effect_throw: 28 videos, 168 frames.

Test_effect_put_down: 26 videos, 216 frames.

Test_effect_pour: 30 videos, ~245 frames.

6. Conclusion

We have presented an effect based Deep Learning system for action detection and prediction. By learning from RGBD video frames this system shows ability of predicting the result of action. Although the general accuracy is not ideal and overfitting problem exists, this system is proved to be effective. Based on planning system, the application of our module has shown great potential in contributing directly to planning works in artificial intelligence. It's possible to output accurate and formative prediction of effects to help robot do planning work.

7. References

[1] Yann LeCun; Leon Bottou; Yoshua Benjio; Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition.

[2] Jonathan Tompson; Murphy Stein; Yann LeCun; Ken Perlin. 2014. Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks.

[3] Arjun Jain; Jonathan Tompson; Mykhaylo Andriluka; Graham Taylor; Christoph Bregler. 2014. Learning Human Pose Estimation Features with Convolutional Networks. ICLR.

[4] Colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] Yoshua Bengio, Patrice Simard, Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. IEEE Transactions on neural networks, VOL. 5, NO. 2, MARCH, 1994.

[6] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. CoRR, abs/1409.2329, 2014.

[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. ICML, 2015.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015.

[9] Shikhar Sharma, Ryan Kiros, Ruslan Salakhutdinov. Action recognition using visual attention. In NIPS, 2015.

Appendix

1. Software Usage

cd $DATASET
find . -name '*.DS_Store' -type f -delete
find . -name '*Icon*' -type f -delete

Get basic information of dataset for action recognizer training:
>> cd data_preprocessing
>> Run train_filename.m to get

```
>> train_filename
Have processed 1th video, which has 4 frames =^_^=
Have processed 2th video, which has 4 frames =^_^=
Have processed 3th video, which has 5 frames =^_^=
Have processed 4th video, which has 4 frames =^_^=
Have processed 5th video, which has 7 frames =^_^=
Have processed 6th video, which has 7 frames =^_^=
Have processed 7th video, which has 7 frames =^_^=
Have processed 8th video, which has 6 frames =^_^=
```

Figure Appendix.1

Get basic information of dataset for effect predictor training:
>> Run train_filename_EF.m

Get labels.txt of effect-predictor:
>> cd effect-recognition
>> Run Labling.py to get effect label txt file.
Get .h5 dataset of 7*7*1024 feature-map from GoogLeNet:
>> cd inception-master
>> ipython 771024.py

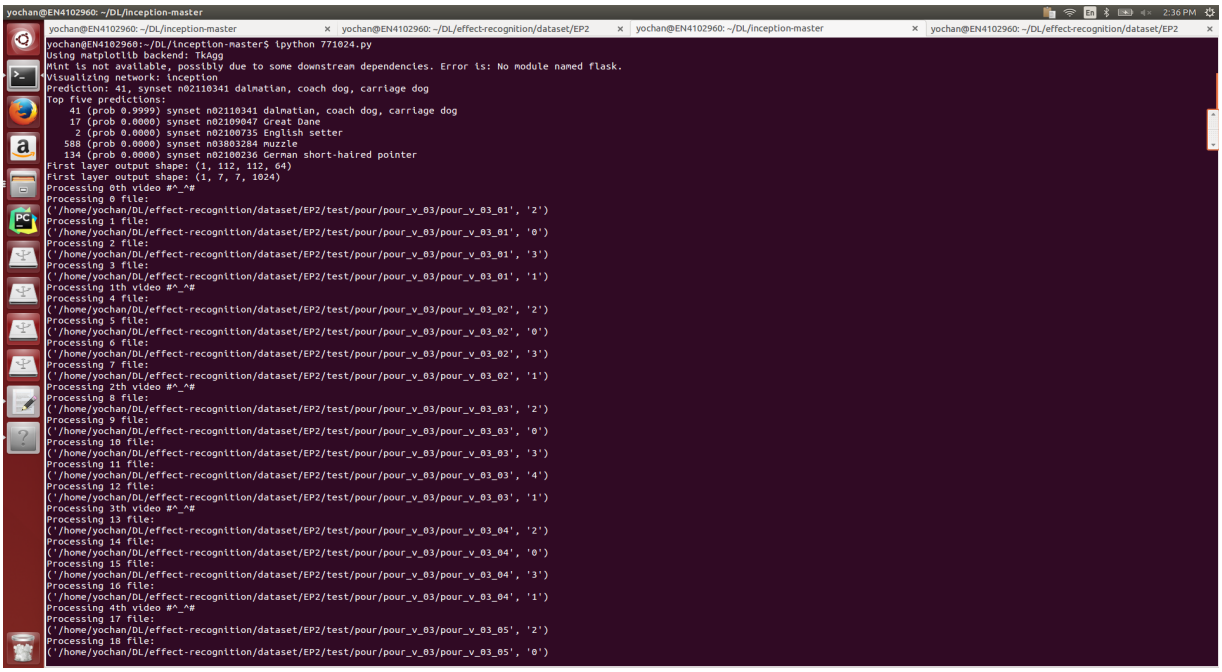


Figure Appendix.2

If you are using an UNIX-based OS:
Please use command “grep –r --color ‘Yantian’ *” to find out how Yantian Zha modified the code;
Please check the title line of the code to see the authorization.

Training of action recognizer and effect predictor:
>> cd effect-recognition
>> THEANO_FLAGS='floatX=float32,device=gpu0,mode=FAST_RUN,nvcc.fastmath=True' python -m scripts.evaluate_ucf11

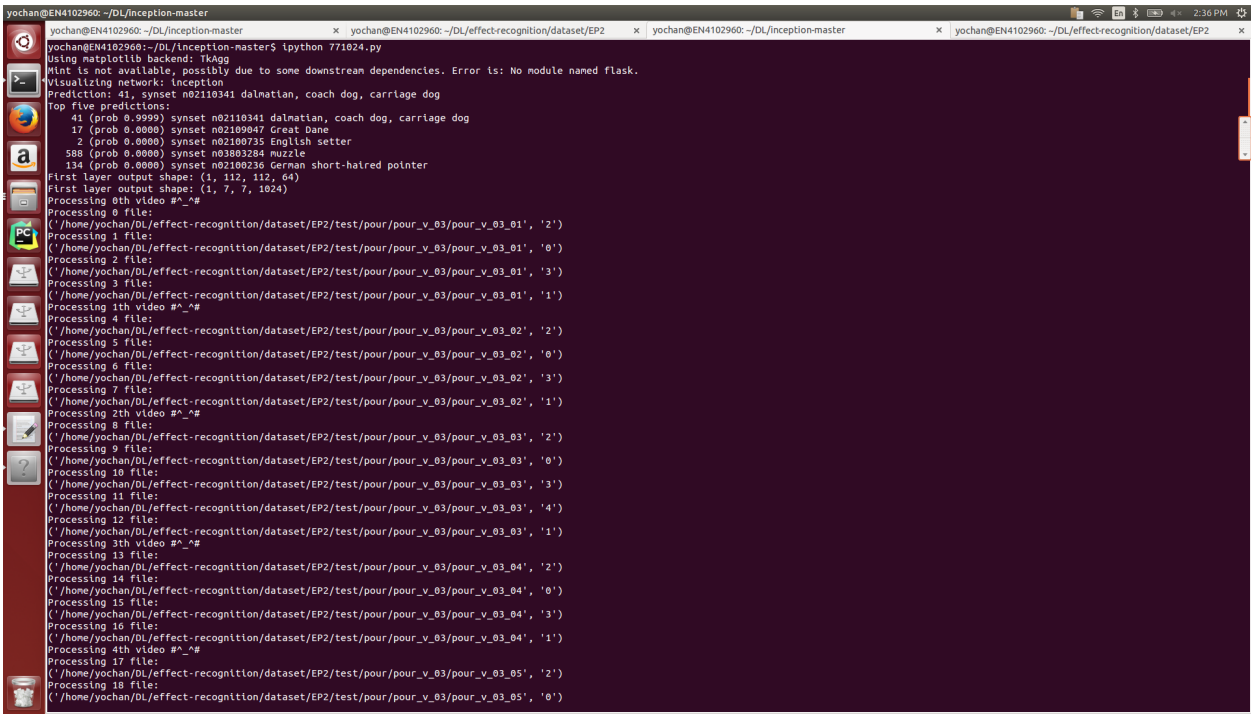


Figure Appendix.3

Final testing of the Effect-Prediction system:
>> cd effect-recognition
>> THEANO_FLAGS='floatX=float32,device=gpu0,mode=FAST_RUN,nvcc.fastmath=True' python -m scripts.predict

2. Contribution List

Yantian Zha: [design of effect prediction system, coding of GoogLeNet dataset preprocessing, coding of attention-LSTM dataset preprocessing, coding of effect- prediction system, author of project presentation, organizing and guiding the whole group, Report writing (Abstract, 1.Introduction, 2.Background, 3.4.LSTM with Soft-Attention Mechanism, 4.Experiment, 5. Results, Appendix)], 34%;

Xiaoyu Zhang: [cascade LSTM-LSTM setup, GoogLeNet Environment setup, Report writing (3.4.LSTM with Soft-Attention Mechanism, 2.Background), Report integration], 22%;

Jianmei Ye: [creation of effect- prediction dataset, coding of its data preprocessing, Report writing (3.2.Dataset Creation)], 18%;

Zongsi Zhang: [creation of effect- prediction dataset, coding of its data preprocessing, Report writing (3.3 Training through GoogLeNet, 6.Conclusion)], 18%;

Mengyuan Zhang: [DL environment setup, dataset creation, Report writing (Figures drawing)], 8%.