DLCV HW4 Report

B05901027 詹書愷

Collaborators: B05602042 林奕廷、B05901074 陳泓均

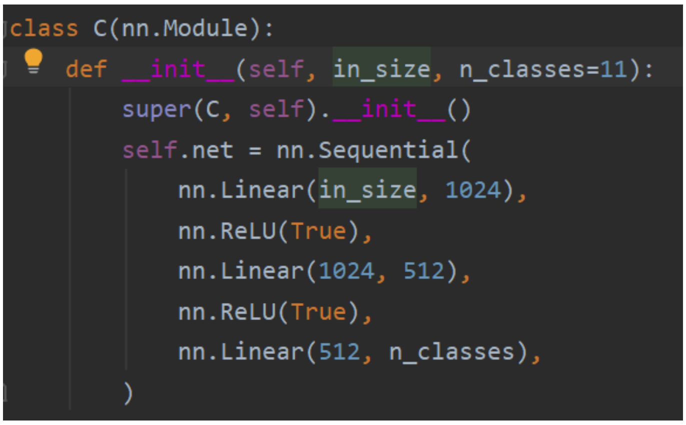
Problem 1: Trimmed Action Recognition w/o RNN

1. Architecture

(1) Extractor

I use Resnet50, pre-trained on ImageNet, as my feature extractor. The output feature has the size *(batch size, 2048)*.

(2) Classifier



```
class C(nn.Module):
    def __init__(self, in_size, n_classes=11):
        super(C, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(in_size, 1024),
            nn.ReLU(True),
            nn.Linear(1024, 512),
            nn.ReLU(True),
            nn.Linear(512, n_classes),
        )
```

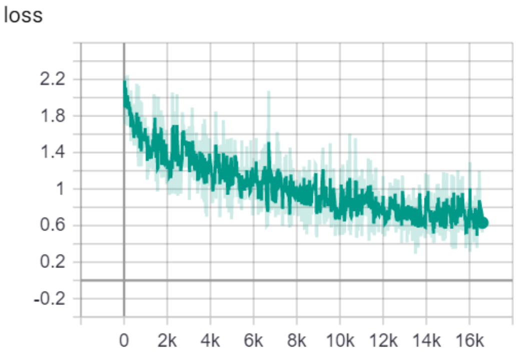
Above is my classifier for Problem 1, which consist of 3 fully-connected layers.

(3) Other Details

1) **Learning Rate Policy:** The learning rate is decrease every epoch using the "poly" policy.
2) **Data Processing:** For the purpose of batch-training, each video is randomly selected 2 frames as its representation. After the features are extracted separately, features of the same video will be concatenated, resulting in features of size 4096 which will then be classified.

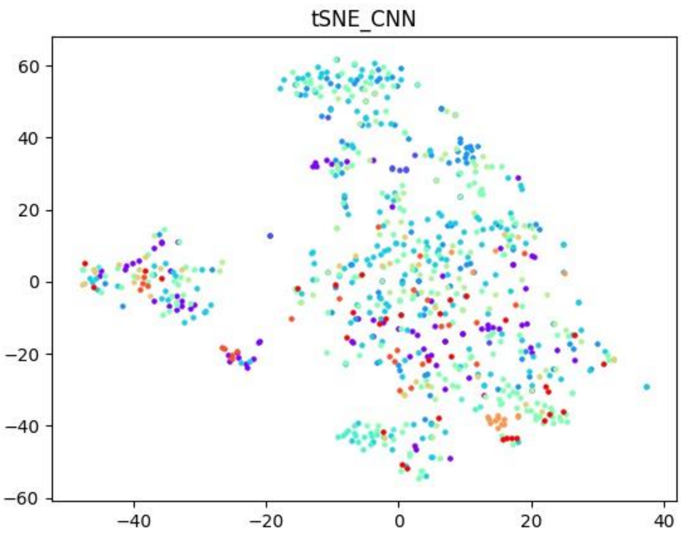
2. Result

(1) Learning Curve



(2) Validation Accuracy: 37.71%

(3) t-SNE



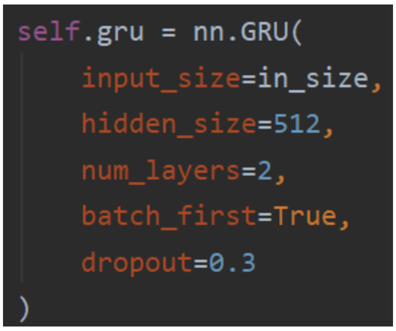
Problem 2: Trimmed Action Recognition w/ RNN

1. Architecture

(1) Extractor

Same as Problem 1, I used the pre-trained Resnet50 as my extractor.

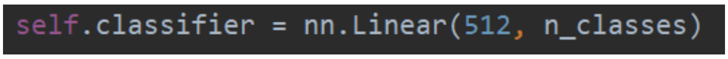
(2) RNN



```
self.gru = nn.GRU(
    input_size=in_size,
    hidden_size=512,
    num_layers=2,
    batch_first=True,
    dropout=0.3
)
```

For my RNN, I used GRU with 2 layers and Dropout to stabilize training.

(3) Classifier



```
self.classifier = nn.Linear(512, n_classes)
```

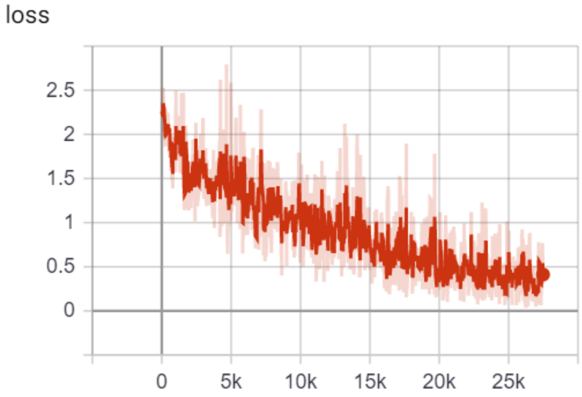
For the sake of simplicity, I reduced my classifier to 1 fully-connected layer, which is basically a linear combination of the RNN features.

(4) Other Details

1) **Learning Rate Policy:** “poly”, same as Problem 1.
2) **Data Processing:** In order to feed sequential input into RNN, I selected 15 frames from each video, with the frames selected evenly throughout the video length.

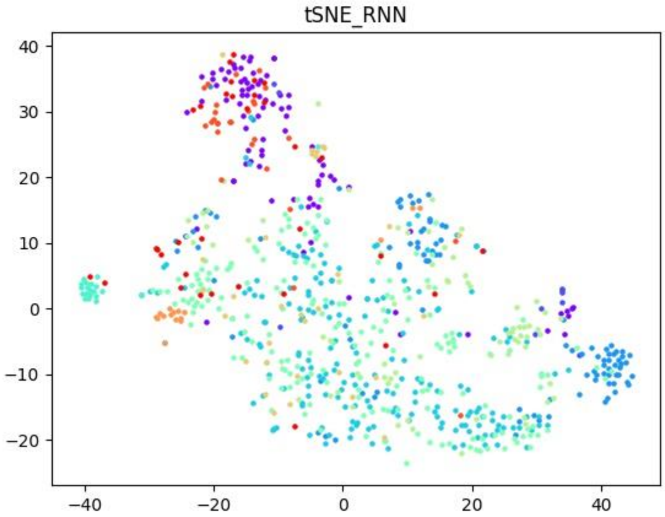
2. Result

(1) Learning Curve



(2) Validation Accuracy: 52.54%

(3) t-SNE



3. Discussion

RNN can achieve significantly better performance than CNN, since it considers the temporal relations between the frames. In comparison, fully-connected layers would process the frame features equally, resulting in a loss of information.

Problem 3: Temporal Action Segmentation

1. Architecture

(1) Model

I applied the same model of Problem 2, with the exception of increasing the Dropout probability.

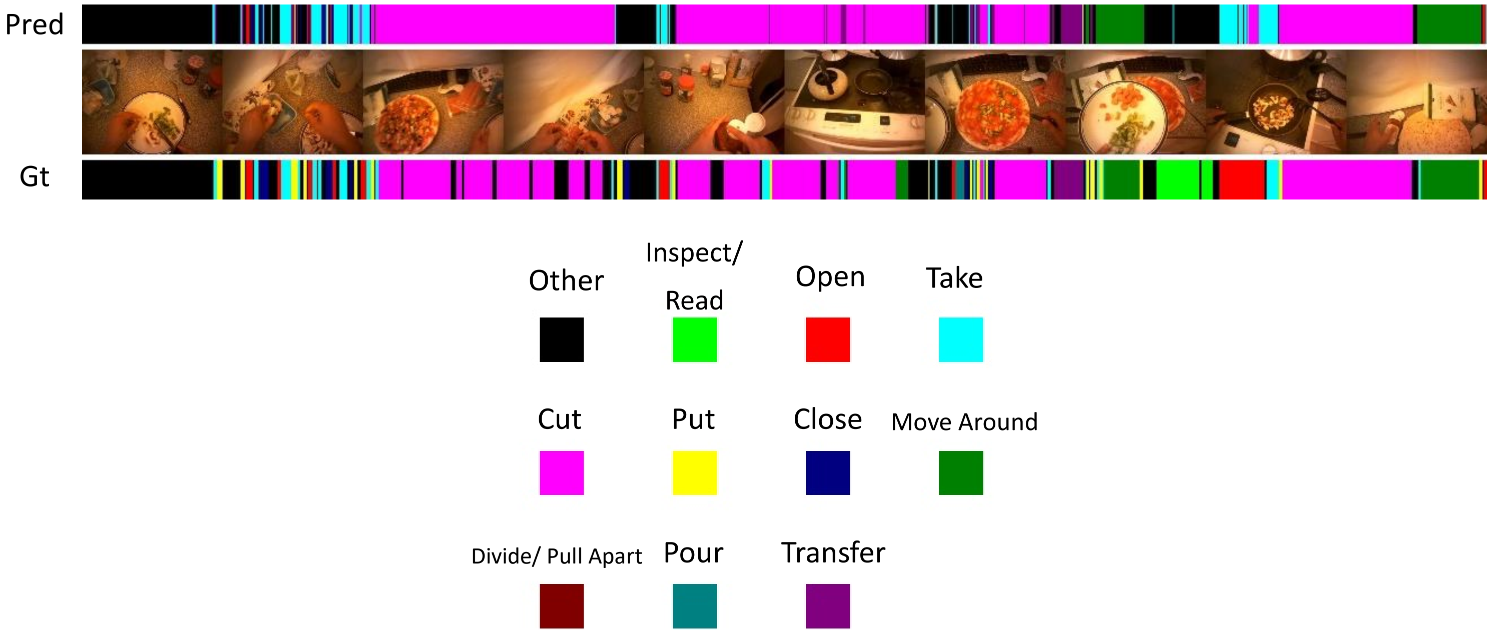
(2) Other Details

1) **Learning Rate Policy:** "poly", same as Problem 1.
2) **Data processing:** Since the size of videos are too big to fit on GPU, some down-sampling is necessary. I down-sample each videos to 256 frames, and only use 1 video to update gradient for each iteration (23 iterations in total for each epoch).
3) **Initialization:** The model is initialized with the best model of Problem 2.

2. Results

(1) Validation Accuracy: 57.55%

(2) Visualization of "OP01-R07-Pizza"



Above is the visualization of the temporal action label classification of video "OP01-R07-Pizza". As can be observed, my model tends to predict more continuous blocks of action labels for some time segments. My model can also roughly capture the transition between different actions (e.g. Cut -> Transfer -> Move Around) using the extracted features and the temporal information of the frames.