CSC449 Final Project Report

Team Name: team
Members: Guo Li Zhenfei Ji

Task 1: Multi-Label Actor-Action Classification

Problem Statement

Goal of this task is to train a model that takes in frames of video which may contain multiple actors performing multiple actions and classifies which actor and action it is belongs to.
By default, there are seven classes of actors performing eight different actions. A single action-class can be performed by multiple actors, but none of the actors can perform all eight actions. This may raise a question: do we decouple this classification problem into two independent classification problems or should we regard each actor-action sample as a unique category classification? After some consideration, we think it is better to treat each pair of samples as a unique category. Actions are performed by specific actors, and hence there exist causality between actor and action, we should not separate them into two independent classes. For example, *adult-flying* or *ball-running* should never be considered. However, once we consider each actor-action pair as a whole, the dataset may become sparser. Specifically, given a batch of samples, one actor-action pair may appear only once and most of time it doesn't present. Therefore, in order to achieve better accuracy, the problem of unbalance has to be solved.

Model Description

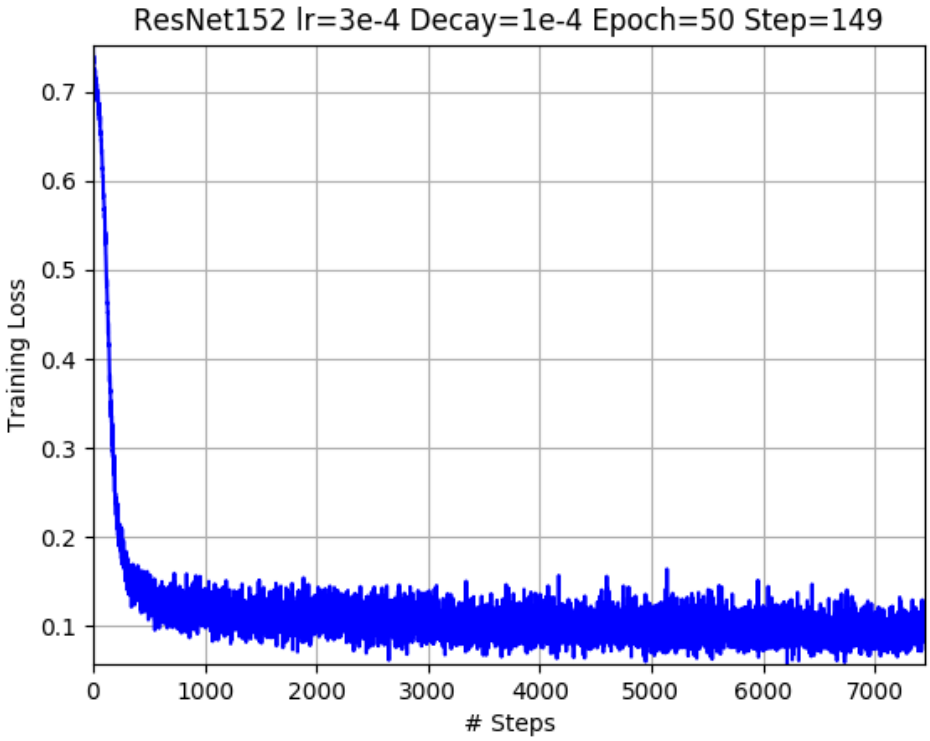
After experimenting with different backbone models, we finally decide to use the DenseNet-161 pretrained on ImageNet as our backbone model. Specifically, DenseNet-161 is used as a feature extractor – we replace the final fully connected layer of the original model with layers defined by ourselves and freeze all other layers. Finetuning the model is achieved by updating the parameters of newly added layers according to the model's performance on validation set. In order to achieve the classification goal on A2D dataset, we change the out features of the final fc layer to the number of classes we receive from parser.

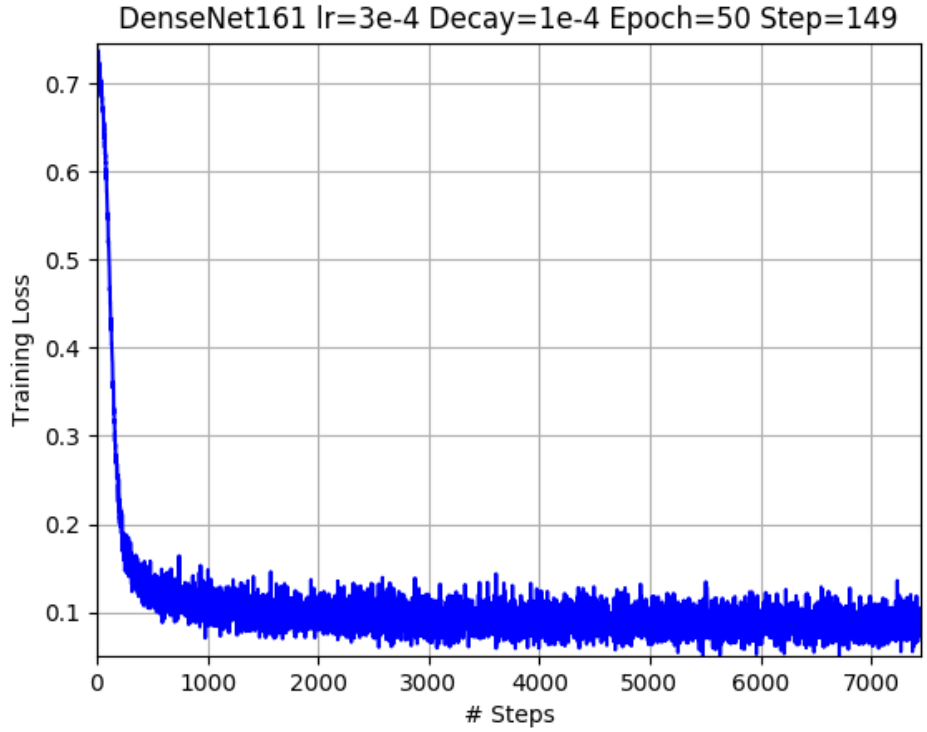
Loss

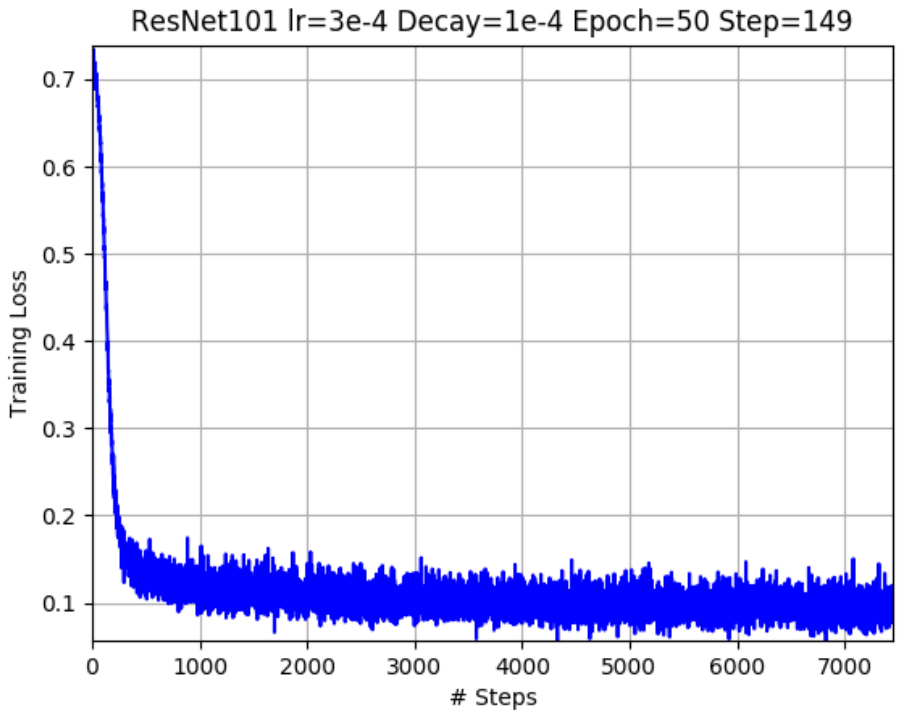
We adopt the BCELoss (*Binary-Cross-Entropy Loss*) as our loss function, experimented with per-class weights.
The loss can be described as:
$\ell(\mathrm{x}, \mathrm{y}) = L = \{l_1, \dots, l_N\}^T$, $l_n = -w_n[\mathrm{y_n} \cdot \log x_n + (1 - \mathrm{y_n}) \cdot \log(1 - x_n)]$, where N is the batch size.

Experiments

We select some pretrained models: ResNet-152, DenseNet-161, and ResNet-101 as backbone to do finetuning separately. The loss trend graphs of each model are as follows:







Optimization Function

We use the torch. Adam as optimization method to optimize our network with mini-batch size of 32, learning rate as $3e-4$ and weight decay as $1e-4$.

Number of Epochs of Convergence

From the loss trend shown on the plot, we can find that at about 2000 steps, roughly 15 epochs, all test models are converged.

Novelty of Method

Due to the very unbalanced nature of the dataset, the model may prone to only fit the positive class or negative class in some cases. Adding per-class weights to BCE loss function helped. Specifically, given a batch of samples, for each class, we calculate the weight of positive class to the total number of this class, and assign $1 - \ weight\ of\ positive\ class$ to positive class, $weight$ to negative class, which will alleviate the imbalance between negative and positive classes.

Performance on Validation Set

The fine-tuned DenseNet-161 model can reach Precision: 35.1 Recall: 46.0 F1: 37.7 on validation set.

The fine-tuned ResNet-152 model can reach Precision: 34.2 Recall: 42.6 F1: 35.6 on validation set.

The fine-tuned ResNet-101 model can reach Precision: 33.9 Recall: 35.0 F1: 33.0 on validation set

Task 2: Actor-Action Detection

Method Description

In this part, we need to use given detectron model to analysis actor-action among videos. In another word, we need to train a model and use this model to recognize what actors and actions in a piece of video. The backbone given in this part is detectron, which is a Facebook research software system for object detection. And this model is built on Mask-R-CNN and use Feature Pyramid Networks for Object Detection.

The data loader part of this part is already given, and the result data consists of two components, RGB image data and optical flow. From the instruction, we know that the optical flow is calculated by Gunnar-Farneback optical flow estimation algorithm. The input data in this part has five dimensions, b, s, c, h and w, corresponding to batch_size, sequence, channel, height, width.

The backbone model we chose for this project is ResNet-101. Based on the backbone code, firstly we load the pretrained model on MS-COCO, which can bring significance improvement to our task. Then we change the hyperparameters of the model and trained to see the performance of current model. After some training, we find out that the backbone model uploaded with pretrained network can produce relatively good performance upon actor classifications but poor performance upon actions. And we find out that ResNet-101 can bring us the highest map performance. We have tried several pretrained model with different hyperparameters such as ResNet-50, ResNet-101 and 152. And from what has been proved from papers, the batch_size of this kind of task should not be too small (at least larger than 8) and the segment size, which is the continuous frames of a piece of video, should be larger than 10. But for these rules, even we apply four GPUs cannot satisfied the training requirement. So, as the best of what we could do, we choose batch_size as 8 and segment as default. From the beginning of the training step, we can set the nw (number of works) to be 2 and this can increase the speed of uploading data from dataset. But after some steps (like 4500 steps), we need to decrease it due to the memory allocation error.

Our final model is about 6300 steps due to the limited time. It is not converged enough but can pass the baseline. The loss function and optimization method we keep the same as default.

Novelty

We have tried to use different models such as ResNet-50, 101 and 152. And we find that the best performance model upon this task is ResNet-101. In order to satisfy the batch_size as 8, we applied 4 k80 GPUs. We also tried different hyperparameters to increase the performance.

Apart from doing those adjust things, we tried to use the same model as what is implemented in code to extract features from RGB frames to extract optical flow from the input. As we did at the first time, we just run out of memory at the beginning of the training. And we tried to apply more and set the batch_size as 2 instead of 4. But it turns out what affect the performance of action classification is likely to be segment size. Due to the limited memory space of google cloud and remaining time we could do to finish the task, we cannot obtain a very good performance by extract RGB and optical flow at the same time.

Besides, we also tried to build our own network to extract flow features from actions. What we do is to use frozen ResNet-50 without last fc layer to extract from input flow and use SPP net at the end of it to gain a definite dimensional output no matter what the input dimension is. And then we can use a fc layer to transform those features into the same dimension of what we have extracted from RGB frames and fuse them together to compute (like two-stream model).

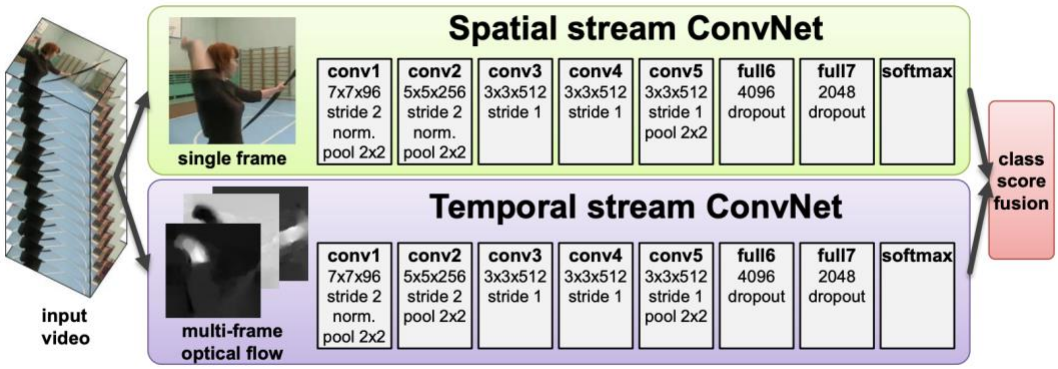
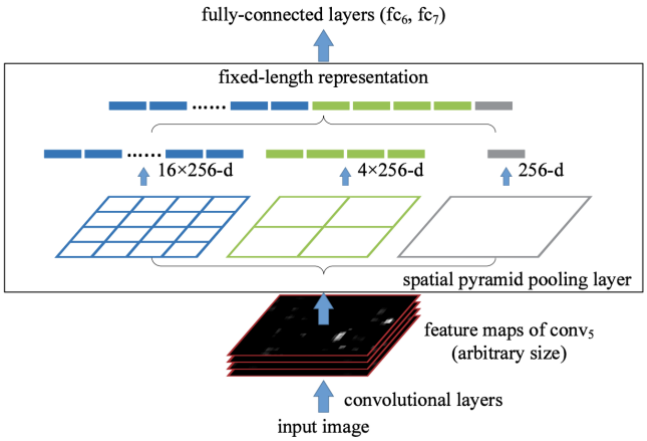


Figure 1: **Two-stream architecture for video classification.**

But, unfortunately, we failed because of the unfamiliar with the backbone model.

SPP Network:



Anyway, we have tried several methods, but the final submission one is implemented with different hyperparameters and ResNet-101 pretrained model.

Performance

The mean ap of actor-action is about 31.7%.
The mean ap of actor is about 73.7%.
The mean ap of action is about 30.8%.

Note

Please note that the path to the result of the generate test pkl is located at:
/csc_249_final_proj_a2d_det/train_output/test_result.pkl