Multi-label Actor-action Classification with Per Class Detection Methods

Team name: VisualCortex
Team members: Bin Yang (leader), Ruitao Lin, Zhexin (Brian) Xu

1. Models and Performance

To solve this multi-label classification problem, our basic idea is to detect the presence of each class independently. The simplest idea would be to train 43 different models for each class independently, but it would be too costly. Therefore, we make use of well-established deep CNNs to extract a rich feature representation of the input, and then build 43 independent detectors on top of that to predict the presence of each class.

We first implemented and trained three different models as described in following subsections, and then we combined them together to form an ensemble as our final working model.

1.1 Per Class Detection with ResNet (PCD-ResNet):

Architecture

For this model, we first obtain a feature vector (2048×1) from the last average-pooling layer of a ResNet152, and then build 43 detectors on top of it for each of the 43 classes. Each detector (43 detectors in total) is composed of a fully-connected layer (2048×2) followed by a softmax operation, and gives the probability distribution (2 probabilities over $[0,1]$) of the detection of one specific class. The output of the whole model is then a vector (43×1) containing the detection probabilities of all 43 classes (Fig. 1).

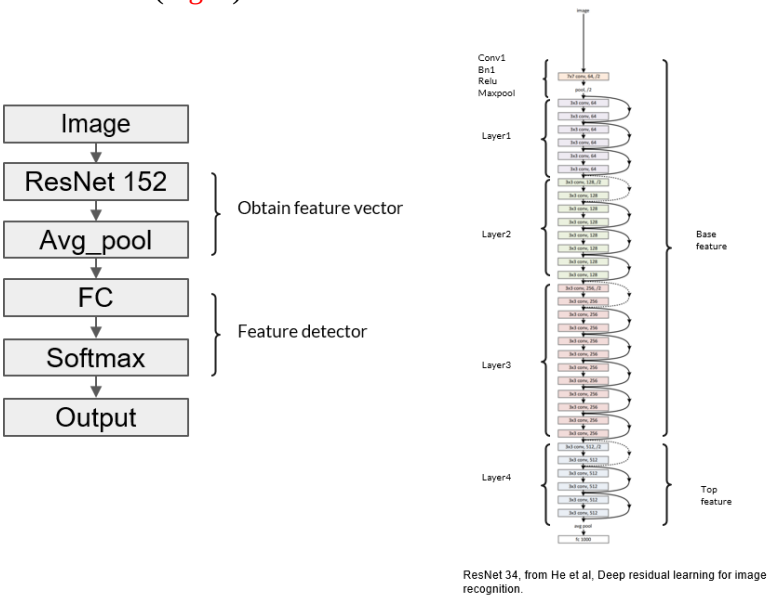

Figure 1 Architecture of PCD-ResNet

Training and Performance on Validation Dataset

- Remove random crop operation for preparing training data

  For preparation of the training dataset, we kept random rotation, random blur and random flip operations initially implemented in A2DDataloader, but removed the random crop operation. All these manipulations are meant to increase the variability of the training dataset hence to reduce the over fitting problem, however, we proposed that the random crop operation could actually be harmful to the training. The reason is that it reduced the number of positive training samples (reduced 26.8% on training dataset and 28.1% on validation dataset) for each class by cutting off actors from images (Fig. 2A). And in deed, the performance of the model dropped substantially when the random crop was applied (Fig. 2B, red VS magenta).

- The model was initialed with a ResNet152 model pre-trained on ImageNet.
- Log loss was used for training:

$$L = -\sum_{i=0}^{42}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

  where y_i and p_i are the ground truth and predicted labels respectively for the i-th class.
- Optimization method is SGD, with lr=0.00001, momentum=0.9 and weight decay = 0.00004.
- As shown in Fig. 2B (red line), the F1 score reached close to 60 after 24 epochs, and then fluctuated below 60 and maximized to 59.4 at 70 epochs.

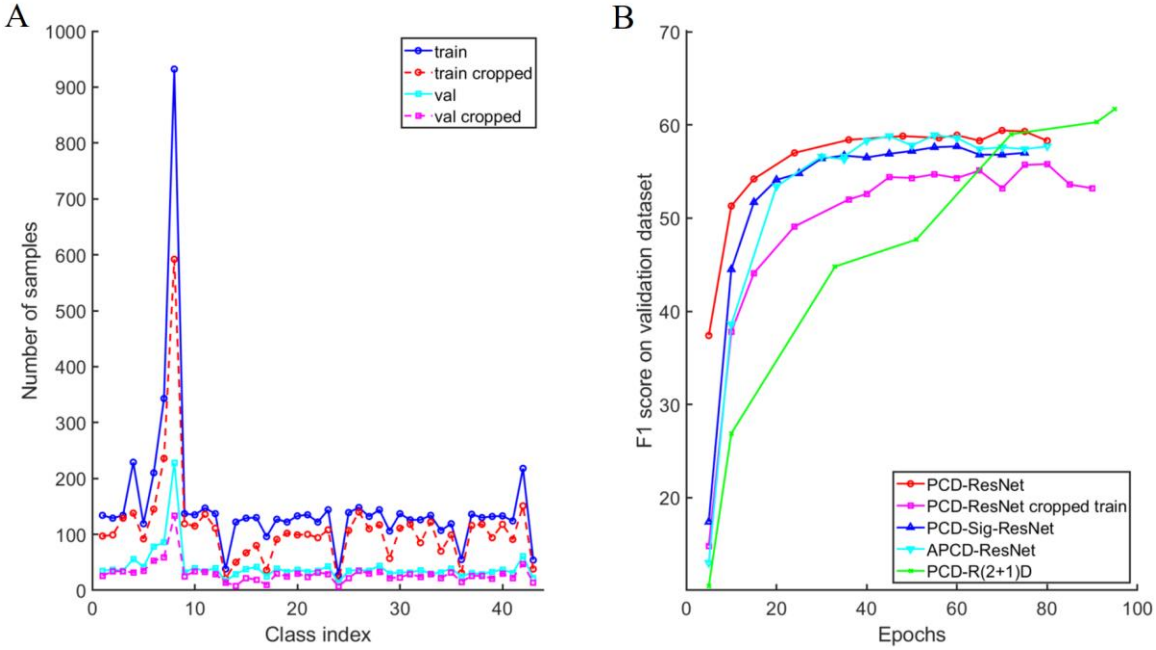

Figure 2. A, Dataset distribution. B, performance of all models.

Alternative Architecture

Each detector in the PCD-ResNet produces a probability distribution over [0,1] by applying a softmax function to two values generated by the detector's fully-connected layer. Then we might want the fully-connected layer to generate only one value and then let this value go through a sigmoid function, with which we can save $2048 \times 43 + 43$ parameters. Therefore, we also implemented this version, which we call PCD-Sig-ResNet. As we can see from Fig. 2B (blue line), its performance is slightly lower than the PCD-ResNet model.

1.2 Attended Per Class Detection with ResNet (APCD-ResNet):

Architecture

In last section, we see that although the PCD-Sig-ResNet model has less complexity compared to the PCD-ResNet, but its performance is also impaired. Therefore, we introduce an attention map to make the model focus on specific regions and then generate one single predictor for each class based on those regions. In this way, we hope that the reduction in number of parameters will not impair its performance. We hack into the ResNet and extract the feature map ($1024 \times 14 \times 14$) from its layer3, and then obtain an attention map from this feature map with a convolution layer (kernel=3, stride=1, padding=1) followed by a sigmoid operation. After that, the extracted feature map is weighted by this attention map and then fed into the rest layers of ResNet152. We replace the last fully-connected layer of ResNet152 with 43 class detectors, each of them is composed with a fully-connected layer (2048×1) followed by a sigmoid function. Then the outputs from all 43 detectors give the final prediction vector (43×1) (Fig. 3).

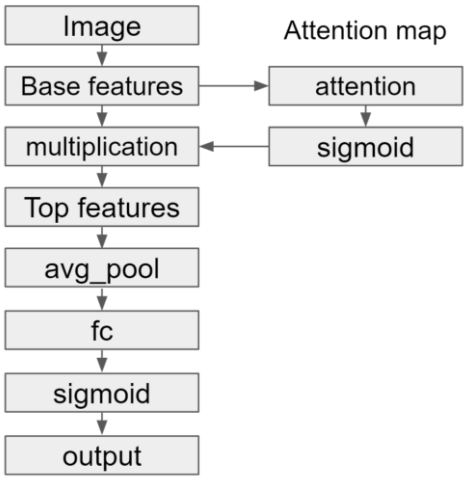

Figure 3. Architecture of APCD-ResNet

Training and Performance on Validation Dataset

- The training parameters were the same as for PCD-ResNet.
- As shown in Fig. 2B (cyan line), compared to the PCD-ResNet model, it reached convergence more lowly, however, it was able to achieve comparable maximum F1 score (58.9 at 55 epochs). Remember that this model has fewer parameters compared to the PCD-ResNet.
- Both the APCD-ResNet and the PCD-Sig-ResNet have the same fully-connected layer in their detectors, however, the attention mechanism makes the APCD-ResNet perform better than the PCD-Sig-ResNet (Fig. 2B, cyan VS blue).
- Fig. 4 shows some example attention maps generated by the APCD-ResNet on the validation dataset. We can see that it generally focuses on regions where actor-actions are.

Figure 4. Example attention maps generated by the APCD-ResNet on the validation dataset. The attention maps allow the network to localize regions of interest while performing actor-action detection.

1.3 Per Class Detection with ResNet(2+1)D (PCD-R(2+1)D):

Architecture

The previous two models all focus on single image, yet the task is actor-action detection, so it would be helpful to exploit a series of frames next to the target image. To achieve this goal, we replace the ResNet152 model in PCD-ResNet with the ResNet(2+1)D model, which takes a series of consecutive frames (16 frames) and perform convolutions in both the spatial and temporal domain independently. The design of the 43 detectors are the same as those for PCD-ResNet (Fig. 5).

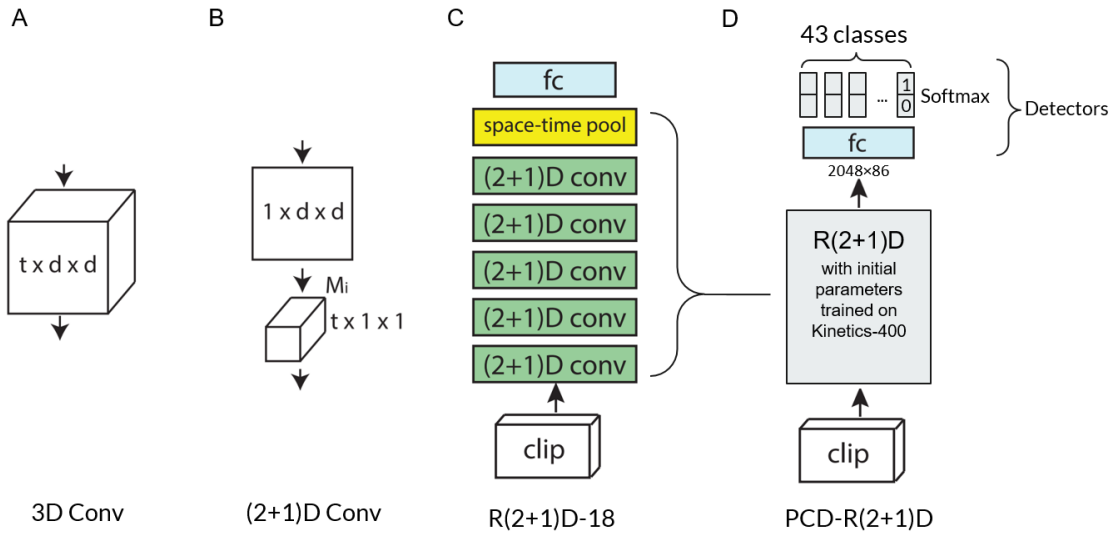

Figure 5. Architecture of PCD-R(2+1)D and its components. **A**, a traditional 3D convolution layer; **B**, (2+1)D layer used in the ResNet-(2+1)D model, where convolutions are perform sequentially in pixel space and in temporal space; **C**, the architecture of ResNet-(2+1)D from Tran et al. **D**, Architecture of PCD-R(2+1)D. This model contains a pre-trained R(2+1)D module, followed by a fully connected layer and the softmax layer for per-class detection.

Training and Performance on Validation Dataset

- For the data preparation, we modified A2DDataset, so that it returns a given number of consecutive frames with the target image at the center. Other preparations are the same as for previous models.
- Training parameters like optimization function, learning rate, momentum, weight decay and loss function were the same as for previous models.
- Since this model is much more complicated than previous models, it consumes 20 times more time to train it than for previous models. In order to speed up the training, we started the training with 4 frames for each image, and increased it to 8 frames and finally 16 frames. In addition, in order to acquire enough time span, for the 4 frames case, we skipped every other frame.
- As shown in Fig. 2B (green line), this model evolved much slower than all other models, however, it outperformed the others with a maximum F1 score of 61.7 at 95 epochs. Note that we only trained 95 epochs due to time constraint, so its performance could be higher than 61.7.

1.4 Ensemble: PCD-ResNet + APCD-ResNet + PCD-R(2+1)D

Since we already built and trained three different models independently, we want to obtain extra performance improvement by combining them together. Specifically, first we calculate three weights for them based on their F1 scores, then we obtain a weighted voting score and a weighted average score from their individual predictions, and calculate the mean of the two scores as the final output:

$$w_1 = F1_1/(F1_1 + F1_2 + F1_3)$$

$$w_2 = F1_2/(F1_1 + F1_2 + F1_3)$$
$$w_3 = F1_3/(F1_1 + F1_2 + F1_3)$$
$$v = w_1[p_1 > 0] + w_2[p_2 > 0] + w_3[p_3 > 0]$$
$$m = w_1p_1 + w_2p_2 + w_3p_3$$
$$p = (v + m)/2$$

where $F1_1, F1_2, F1_3$, w_1, w_2, w_3 and p_1, p_2, p_3 are F1 scores, weights and predictions of each model, v and m are the weighted voting and average scores respectively, and p is the final output.

On the training dataset, the performance is shown in the following table. Unfortunately, due to the time-consuming nature in running this network, we are not able to obtain the scores for this method in a limited time period. We will continue examining this method and will present the results in our final presentation. The submitted test result was obtained from the PCD-R(2+1)D model.

Model	**F1 score**	**Precision**	**Recall**
PCD-ResNet	59.4	57.9	66.3
APCD-ResNet	58.9	57.4	65.6
PCD-R(2+1)D	61.7	60.9	68.6
Ensemble	(unable to run due to time limit)		

2. Novelty

1. **Convert the multi-label classification problem into a problem of independent detection of each possible class.**

   For a multi-label classification problem, people usually make use of both ground truth class labels and ground truth bounding boxes or semantic segmentation maps, so they are able to perform single-label classification in single regions of interest (e.g., RCNN, YOLO). Yet in our task, ground-truth bounding boxes or semantic segmentation maps are not available, therefore we cannot directly adopt those models for our task.

   On the other hand, typical classification networks are usually built for single-label classification, which output a score vector which sums up to 1 and represents the predicted probability distribution of the target over all possible classes. Obviously, this could not perform well if applied directly to our multi-label classification task.

   However, the problem can be easily solved by determining the presence of each class independently. Moreover, instead of training 43 different models for each class independently, which is too costly, we let all 43 detectors share the same feature vector generated by a deep CNN.

2. Ensemble 3 different models. By assembling all three models proposed in our work, the classification performance was increased.

3. **Implementing an attention mechanism.**

   We hack into a deep CNN and extract a rich feature map from a hidden layer, then pass the feature map to an attention layer, which consists of a convolution layer (kernel size of 3×3, depth of 1 or 43) followed by a sigmoid layer, and finally obtain a feature map. Then before the extracted feature map goes through the rest layers of the CNN, it will be filtered by a multiplication with the attention map.

   This attention mechanism could potentially extract features from locations most relevant to the actor-action detection, and

4. Remove the random crop manipulation on the training dataset in the provided a2d_dataset:

   a. It skewed the distribution of the training dataset over all classes (Fig. 2A)
   b. Experiments show that the crop manipulation impair the performance on the evaluation dataset.

5. **Method to speed up the training of PCD-R(2+1)D**

   Because the PCD-R(2+1)D model performs a series of 3D convolutions, it is much slower than our 2D CNNs (a magnitude of about 20). In order to speed up the training, we start from a small number of frames for each image, and gradually increase it when the performance improvement per epoch gets smaller, and we do that repeatedly until reach to 16 frames for each image. Moreover, when the number of frames per image is small, we down sample in time by skipping frames so that the frames we take for each input image still spans enough in time. Taken together, manipulating the total number of frames and the sampling rate allows us to speed up the training processing as well as preventing the model from overfitting.