Levente Szabo
12/20/2019
Machine Learning for Computer Vision
Final Project

Mask RCNN:

This algorithm serves to provide state of the art image segmentation. For a given class it yields both a bounding box and a binary mask for each instance [1]. To better understand the practical implementation of Mask-RCNN we provide the results obtained on a novel dataset using a pretrained model. The EgoHands dataset [2] from Indiana University is used, it contains 48 videos of 100 frames each with bounding boxes and binary masks for each hand. The implementation of Mask-RCNN is in the pytorch torchvision package and closely follows the following tutorial [3].

For training our model we construct a pytorch dataset with __getitem__ method that yields image and features (boxes,masks,labels,area) in a round robin fashion by selecting the first frame from the first video, then the first from the second and so on. Selection of training data in this fashion increases the diversity of the images seen and allows the algorithm to learn from a training set more closely resembling the true distribution of hand images. We used a pre-trained resnet-50 backbone (pre-trained on the COCO dataset [4]) with 256 hidden layers. The optimizer used was SGD with a learning rate of 0.005, a momentum of 0.9 and a weight decay of 0.0005. The learning rate scheduling was not used due to performance issues. All training and evaluation was performed on the NYU Prince HPC cluster using 2 gpus, 2 cpus and 24 GB of memory. Final training and testing results were done with a split of using 24 videos (2400 frames) for training and 2 videos (200 frames) for testing.

Metrics

Learning Rate	Loss Classifier	Loss Objectness	Loss Mask	Loss Box Reg	Loss Rpn box reg	Loss
0.005	0.1081	0.0119	0.0371	0.1093	0.0187	0.2861

The errors encountered through training involved memory issues, unforseen data and high computational demands. When a batch size larger than 10 was used the the gpu would often run out of memory, so a batch size of 10 was used for all training. Often the kernel would crash and training would stop when longer jobs were placed, so a maximum of 2400 frames was used for training. In certain frames there would be no hands at all and to avoid any miscalibration these frames were avoided altogether.

To construct a video out of the evaluated images (all 100 frames from the 25th video) the mask was first turned into RGB with inside values as red, then this was added the original image to create an image with a mask overlay. After this the most significant (highest 50% scoring) boxes were also added as blue rectangles. Due to the lack of model training the images do not perfectly segment the hands and there is an excess of boxes. However the evaluated images and the video on the training and test set indicate a reasonable working model. The model always has several boxes around each hand and very rarely mistakes foreign objects (i.e faces) for hands. Once the masks, boxes and original image are pasted together they are turned into a .avi file (see train_video.avi and test_video.avi) using VirtualDub [5] an open source video editing software.

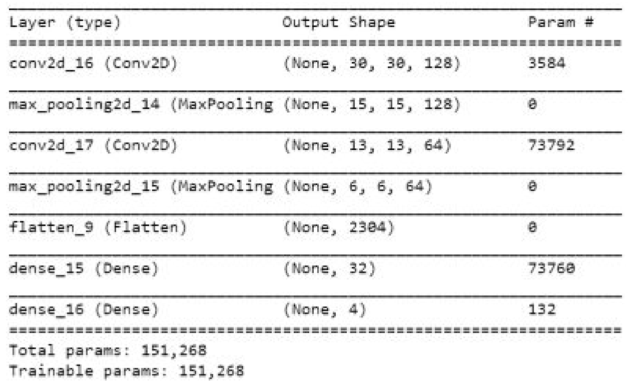
Overall I believe this model to be usable for hand segmentation and if provided more training data (or epochs) and memory then the metrics and the video quality would greatly improve.

Activity Classification:

The EgoHands dataset contains 4 different categories of activities (card playing, chess, jenga and puzzles). To classify the activity taking place in each video we apply a 2D convolutional network on a region of interest for each frame and take the class with the maximum average prediction to be the classification for the video. The keras package is used for all machine learning algorithms.

The dataset has 12 videos (1200 frames) for each category. In order to maximize our available training data we take the first 11 videos of each category as training data and the last as a testing set. The label for each image is encoded in one-hot encoding ([1,0,0,0] indicates a label of 0 or “card playing”). Finally, since we have the bounding box information available we utilize it to get a region of interest for the image. Over all boxes we select the minimal bounding box to be [min(min_x), min(min_y), max(max_x), max(max_y)]. These coordinates give us a large box which is then resized to be 32 by 32 pixels.

Finally in keras we construct a 2D convolutional neural network with architecture given below. The optimizer and learning rate was critical to making this model work. Initially the adam optimizer was used but it was too aggressive and the model ended up classifying all images as being of one class. When the SGD optimizer was used with a specified learning rate of 0.00001 the model yielded usable results.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 13, 13, 64)        73792
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_15 (Dense)             (None, 32)                73760
_________________________________________________________________
dense_16 (Dense)             (None, 4)                 132
=================================================================
Total params: 151,268
Trainable params: 151,268
```

We used categorical cross entropy as our loss function and ran the model for 10 epochs. Finally we had an accuracy of 0.75, this was determined by counting the number of correctly classified images in each video and if more than half were correctly classified then the video was as well. Only 1 out of 4 testing videos (puzzle playing) was incorrectly classified. Better results could be obtained if more finetuning was used such a dropout layers, data generation through horizontal/vertical flips, and of course more epochs. Fortunately the activities had a large deviation between them, higher resolution imagery and better segmentation would be required if we wished to be more discerning in our classification. Overall the region of interest method extraction combined with a simple 2D convolutional neural network yields reasonable results for the EgoHands dataset when attempting to classify activities.

[1] Kaiming He et al. Mask R-CNN, Facebook AI Research, January 2018
https://arxiv.org/pdf/1703.06870.pdf

[2] Bambach Sven et al. EgoHands: A Dataset for Hands in Complex Egocentric Interactions, December 2015
http://vision.soic.indiana.edu/projects/egohands/

[3] Finetuning Instance Segmentation.
https://colab.research.google.com/github/pytorch/vision/blob/temp-tutorial/tutorials/torchvision_finetuning_instance_segmentation.ipynb

[4] Coco Dataset
http://cocodataset.org/#download

[5]
http://www.virtualdub.org/