Image Classification with CNNs

Abdurhman Fayad ABDURAHMAN.FAYAD@GMAIL.COM

1. Model Description

As the problem to be solved has change in domain, I decided to use a pre-trained Faster R-CNN model with a ResNet-50 backbone that is included in the PyTorch torchvision module and fine-tune it. The model is created by the following components, Faster R-CNN, ResNet-50 backbone and FPN (Feature Pyramid Network).

Faster R-CNN, an extension of Fast R-CNN, is a state-of-the-art framework for object detection. The primary contribution of Faster R-CNN is its introduction of a Region Proposal Network (RPN). The RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals.

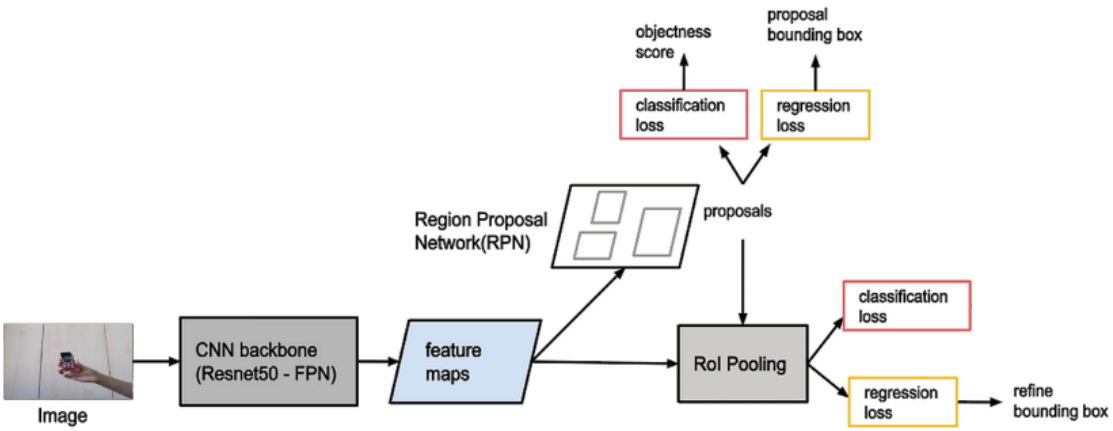


Figure 1: Network Architecture

The Feature Pyramid Network (FPN) based Faster R-CNN fine-tuned with surgical instrument locations. The network receives an input of an image of arbitrary size. The backbone network is a Resnet50-FPN CNN which is connected to a Region Proposal Network (RPN) that shares its convolutional layers with the detection network. The RPN is a fully convolutional network which generates region proposals which are highly likely to contain an object. The detection network pools features out of these region proposals and sends them to the final classification and bounding box regression networks. The final output is a bounding box for each detected instrument and a class label (grasper or clipper) with its confidence score.

The architecture of Fast R-CNN is simpler and more streamlined than its predecessors. The model takes an entire input image and a set of object proposals. The image is processed through several convolutional and max pooling layers to produce a convolutional feature map. Each object proposal is then mapped onto this feature map. A RoI (Region of Interest) pooling layer is then used to convert the features inside each RoI into a fixed-size feature map. These fixed-size feature maps are fed into a sequence of fully connected layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K+1 object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the K object classes.

2. Dataset

The dataset used for this project consist of:

- 8 different categories
- 1600 images for training
- 800 images for evaluation
- While in the train set images are characterized by different backgrounds, in test set all the images are collected with the same scenario.

Figure 2: Example of Train Images

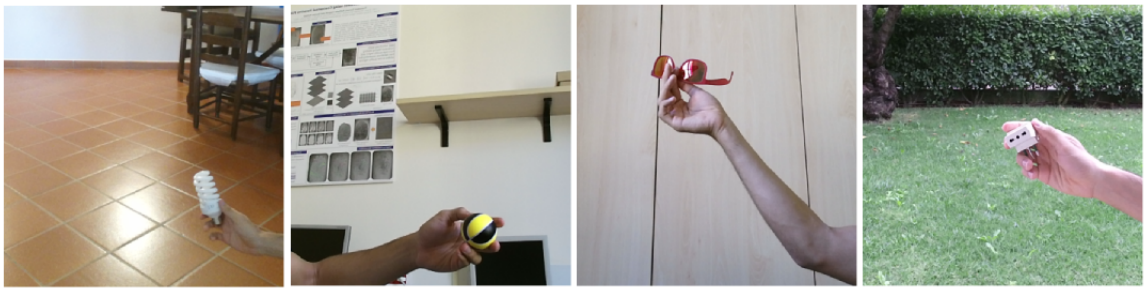
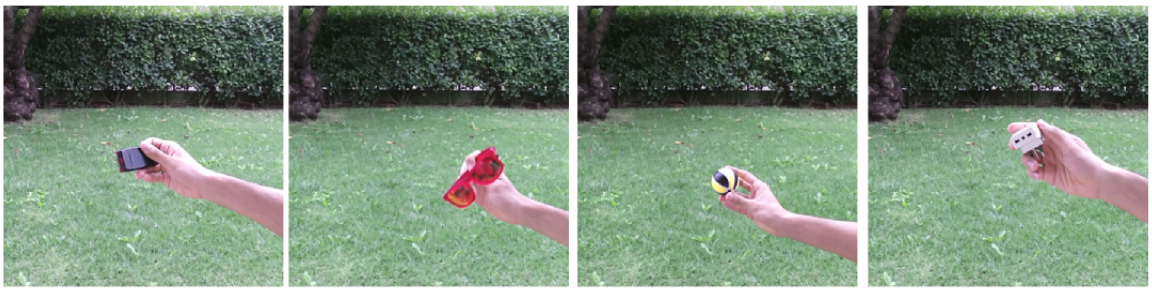


Figure 3: Example of Train Images



3. Training procedure

The training of the model was done by performing the following steps:

Dataset Class

The first part is to create a custom dataset class that supports two data splits: "train" and "test". During the "train" split, the dataset reads and processes images along with bounding box annotations recovered from mask data, and it allows for applying user-defined transformations. The "test" split is for evaluating the model on unseen data.

Dataloaders

The next step is to implement data transformations and data loading for training images using the custom dataset class, while defining two transformation functions for train and test which specify the desired augmentations and conversions to tensors using the albumentations library, and setting the batch size=8. Finally, the DataLoader model is used to create a data loader for the training dataset, facilitating efficient batched loading and processing of images. This approach provides a flexible and efficient pipeline for training image data with customizable augmentations and transformations.

Faster R-CNN

Here we initializes a Faster R-CNN model with a ResNet-50 backbone, pre-trained on a large dataset, for object detection tasks, then we modify the model to accommodate our custom dataset with a specific number of classes. By retrieving the input feature size of the classification layer, a new FastRCNNPredictor is created with the desired number of output classes. This customized predictor is assigned to the model's region of interest (ROI) heads, enabling the model to generate class predictions for the specified number of classes during object detection.

Training

Finally we start training the model for object detection tasks for 10 epochs. The optimizer used was SGD with learning rate=0.005, momentum=0=0.9 and weight decay=0.0001. We utilizes a training loop to iterate over the dataset, computing losses and updating the model's parameters. The loss values are tracked and stored to be analyzed. The model is put in training mode, and the optimizer is used to perform backpropagation and update the gradients. We also periodically print of loss values. At the end of training, the elapsed time is calculated and displayed.

4. Experimental Results

After training the model for 10 epochs, we can plot the graph of the loss values through the training.

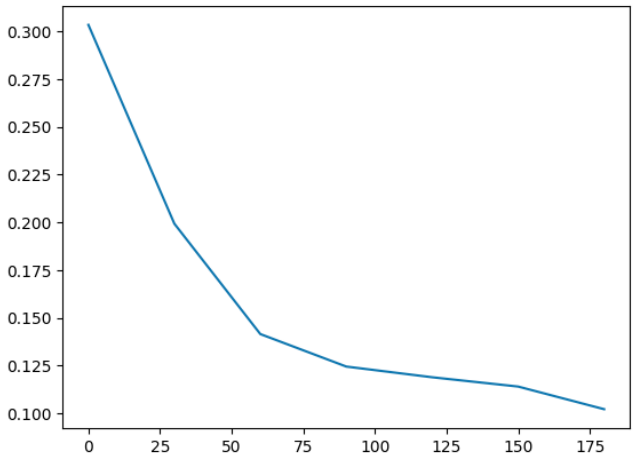


Figure 4: Loss values through epochs

We can test the model's ability to predict the bounding box of the object:

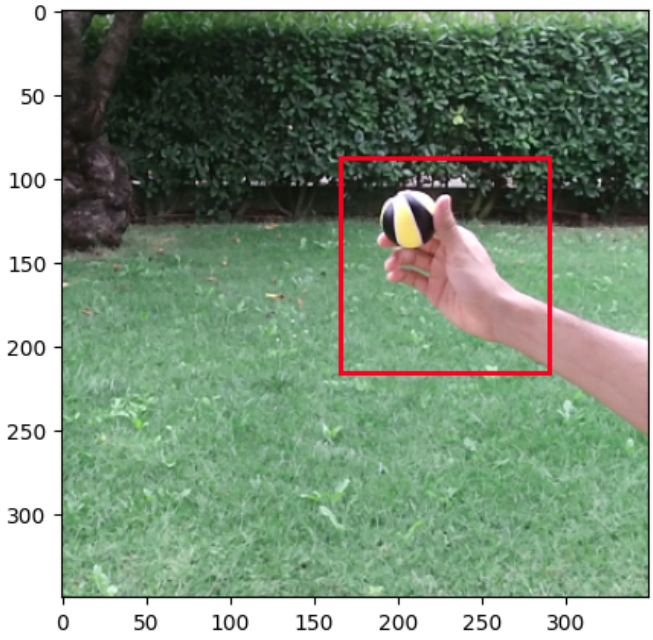


Figure 5: Object detection on test image

As we can see, the model performs very well for predicting the bounding box, and the accuracy on the complete test set was **0.95** in the kaggle competition.