Computer Vision Assignment 2

22580530

May 2023

Question 1

I construct a model based on SqueezeNet[2] with pre-trained weights. SqueezeNet is a neural network architecture which makes use of "squeeze layers", with 1x1 convolution filters, to reduce the number of network parameters while maintaining accuracy. The network manages to achieve comparable accuracy to AlexNet with 50× fewer parameters. I choose SqueezeNet for its impressive efficiency.

I load SqueezeNet version 1.1 and its weights through Pytorch. SqueezeNet is trained on ImageNet and requires images in a specific format. The images in the CIFAR-10 dataset are cropped to size 224×224 and the pixel values are normalized. The SqueezeNet architecture consists of 8 "fire modules", which use convolution layers to create a latent representation of the image, and a final "classification" module which uses a convolution layer and adaptive average pooling to generate values for 1000 ImageNet classes. I replace the classification module with an adaptive average pooling layer (to reduce the dimensionality of the latent representation), a flatten layer and a fully connected layer with a softmax activation function to generate a confidence score for each of the 10 CIFAR-10 classes.

All training was done on my Acer Aspire 5 laptop (no GPU). First, I freeze all the weights except for those in my new classifier module. I train the network using stochastic gradient descent with a learning rate of 0.001 for two epochs. The training takes about 908 seconds per epoch. This yields a test accuracy 75%. Next I freeze only the first 4 fire modules and tune the others. For the tuning I use a lower learning rate of 0.0001 and only train for one epoch, which takes 1135 seconds. This achieves a test accuracy of 84%. Finally, I retrain the model with all the weights unfrozen, again with a learning rate of 0.0001 and for one epoch, which takes 1914 seconds and results in a test accuracy of 88%.

These are all a massive improvement over the model from the first assignment, which scored a test accuracy of 62%. This illustrates the power of an advanced architecture like SqueezeNet to learn a latent representation for arbitrary images, which can then be used for different kinds of image classification without changing the weights. However, the model in the first assignment only took 90 seconds to train. This shows that even when only adjusting a small subset of weights, partially training a large network can be far more computationally expensive than training a small network from scratch.

Tuning more of the weights of the original model for a specialized classification task can further improve accuracy, but also increases training time per epoch. Also, training layers further back in the network seems to give diminishing returns on accuracy. The best approach seems to be to tune weights from the back of a pre-trained network up to a layer which gives a good balance between training time and accuracy.

Question 2

For this question I evaluate the Faster R-CNN[3] network for image detection. Faster R-CNN is an extension of Fast R-CNN, which is a unified network for object detection. In Faster R-CNN, this unified network is combined with a fully convolutional "Region Proposal Network" to quickly propose rectangular object regions. Thus, Faster R-CNN first creates a latent representation of the image, proposes object regions within the image and then classifies these regions with some class and confidence value.

I load the Faster R-CNN network with a ResNet-50 back-end (for latent image representations) and pre-trained weights using Pytorch. For the test set, I use a set of 100 images containing cats and dogs from the COCO 2017 validation dataset. This data is loaded using the FiftyOne library. The class labels used by the network are indexed differently from those in the test set, so the detection labels are converted to the correct indices. Some of the model's detections on a handful of images are shown in Figure 1. The model is clearly impressive, if imperfect.

The mAP (mean average precision) is a popular metric for evaluating the effectiveness of object detection models[1]. The AP is calculated per object class in an image, and the mean over all classes is used. There are three parts to calculating the AP for a class:

- Precision: the proportion of model detections which are correct.
- Recall: the proportion of objects in the image detected by the model.
- IoU (intersection over union): The ratio between the area of intersection and the combined area of two bounding boxes.

The IoU is used to determine whether a detection made by the model is correct: for a correct detection, the IoU between the model's bounding box and the ground truth box must exceed some threshold.

For a given image, the model makes multiple detections. These detections are sorted in decreasing order of confidence. For each detection, the precision and recall are computed. The recall will monotonically increase down the list as more objects are detected, while the precision decreases for every incorrect detection. The idea behind AP is to calculate the area under the precision-recall curve, where recall is on the x-axis and precision is on the y-axis. Since the precision fluctuates between correct and incorrect detections, the curve is

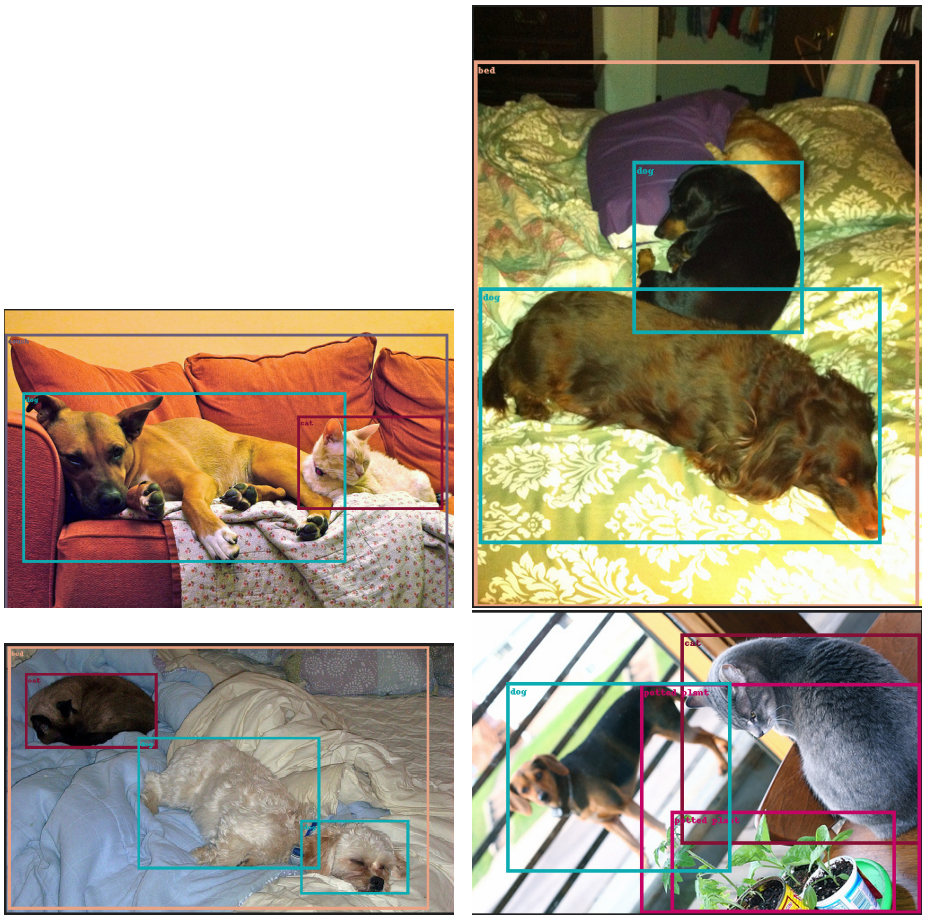

Figure 1: Faster R-CNN detections

jagged and in practice the area is approximated by some method, which differs between implementations.

For this test, the mAP is computed using the TorchMetrics library. This implementation approximates the area under the precision-recall curve by calculating a 101-point interpolated AP. This means that the curve is divided into 101 equally spaced points along the recall axis, and the AP is the average of the precision at each of these points.

Using an IoU threshold of 0.5 for correct detections, the mAP for this test set is equal to 0.6882. This indicates a solid object detection performance by Faster R-CNN on these images. This is expected, as Faster R-CNN is a complex neural network which likely had examples of all the object classes in this simple test set in its training data.

(a) Original Image (b) Noisy Image (c) Neural Network Output

Figure 2: Denoising network on example images

Question 3

I design a neural network for denoising images. The network is fully convolutional and uses an architecture similar to that of auto-encoders. The "encoder" consists of four convolution layers with $16, 32, 64$ and 128 channels respectively. The first three of convolution layers are followed by 2×2 pooling layers. The "decoder" has its own four convolution layers which mirror the ones in the "encoder", and similarly three matching unpooling layers. All convolution layers in the network use RelU activation and 3×3 filters.

In order to train the network, I create a dataset which has ten different noisy images for each image in the LFWcrop color dataset. I then split this set into a training, validation and test set in a 80%/10%/10% split. During training, the network takes a noisy image as input. The loss function is the mean squared error between the output of the network and the original image corresponding

to the noisy input image. I use a batch size of 5 and the Adam optimizer with a learning rate of 0.001. Since this dataset is ten times as big as the LFWcrop dataset, I only train the network for one epoch on the training set. The final network achieves an average MSE of 0.00167 on the test set (where pixel values are in the range $[0, 1]$).

Figure 2 illustrates the output of the trained neural network. It is able to effectively remove noise from an image. The image becomes noticeably blurry and some details are lost. However, this is not too bad considering the severity of the noise, which can obfuscate substantial portions of an image. The overall blurriness may be due to the pooling layers, which condense the dimensions of the image representation in the network. The network is overall quite small and may lack the capacity to accurately reconstruct the original images.

References

[1] Jonathan Hui. *mAP (mean Average Precision) for Object Detection.* https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. Mar. 2018.

[2] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[3] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).