CMSC 491/691: Computer Vision **Spring 2024**

Lecture 10: 2024-03-6 Visual Recognition

Lecturer: Tejas Gokhale *Scribe: Ben Lagnese*

10.1 Recap of Last Lecture

In the last lecture, we covered the motivation behind Convolutional Neural Networks (CNNs or ConvNets). A convolutional layer takes in a 3D layer. For our purposes, these are the dimensions of an image: width x height x channels (3 channels for RGB). The output layer is generated by convolving a filter with N x M x channels dimensions over the image (N and M are arbitrary, but cannot exceed image dimensions). This convolution generates a layer of depth 1. Additional filters can be applied, increasing the depth of the output by 1 for each filter added.

Additionally, we can pad the input image with p-deep zeroes or apply a stride s to the convolution (e.g. moving the convolution filter s pixels over at a time rather than one). With convolution kernal size k, i.e. kxk dimensions, the final size of the output is:

$$w_{out} = \lfloor \frac{w_{in} + 2p - k}{s} \rfloor + 1 \tag{10.1}$$

Convolution is often followed by pooling, a technique that convolves the hidden layer to make a smaller layer. Pooling uses filters that preserve important information. A common method is max pooling, which applies a max filter. To backpropagate the max pool, all unchosen indices have their gradients set to 0 and are not trained for that iteration.

10.2 Training ConvNets

To train ConvNets, the steps can largely be broken down into 3 steps:

1. Gather labeled data. The data can be split into training and the testing data as well.
2. Find a ConvNet architecture. Hyperparameters also need to be set, including initial weights, the learning rate, and regularization strength.
3. Minimize loss. Monitor the progress and adjust hyperparameters as desired.

However, the training dataset may be large. To improve the speed, Mini-batch Gradient Descent may be used. Instead of using the entire training set for each iteration, a batch is sampled from the dataset and used instead.

Models may run into issues with overfitting or underfitting. A model overfits if it tries to fit the data too tightly, modeling the noise rather than the general pattern. Conversely, a model underfits if it insufficiently models the pattern. Bigger, more complex models tend to overfit.

Regularization is a technique that curbs overfitting. It adds an additional loss function that penalizes large weights, keeping the weights closer to a linear state. The function chosen and the strength of the regularization can be customized. Examples include:

$$\text{L2 regularization} \qquad L_{reg} = \lambda \frac{1}{2} \left|\left|W\right|\right|_2^2 \tag{10.2}$$

$$\text{L1 regularization} \qquad L_{reg} = \lambda \left|\left|W\right|\right|_1 \tag{10.3}$$

$$\text{Elastic Net} \qquad L_{reg} = \lambda_1 \left|\left|W\right|\right|_1 + \lambda \left|\left|W\right|\right|_2^2 \tag{10.4}$$

$$\text{Max norm} \qquad L_{reg} = \left|\left|W\right|\right|_2^2 \leq c \tag{10.5}$$

Regularization may also be called "Weight decay", since the back propogation of the loss function is merely a function of the weight, e.g.

$$L_{reg} = \lambda \frac{1}{2} \left|\left|W\right|\right|_2^2 \longrightarrow \frac{\partial L}{\partial W} = \lambda W \tag{10.6}$$

Another techique to reduce overfitting is Dropout. When training the neural network, some layers can have neurons activate with a probability - i.e. only some neurons can be used to predict an output for any given iteration. This prevents each neuron from becoming too specialized, since the other neurons that could counteract extreme factors won't always be active.

10.3 Introduction to Scene Understanding

We discussed several ways to classify scenes.

1. Classify each pixel in the scene as a category.
2. Classify the scene itself.
3. Captioning the image with human language.
4. Captioning multiple parts of the image.

It is challenging to accomplish this.

- Object categories don't encapsulate all of the information an image.
- How fine-grained the categories should be is ambiguous.
- There is information associated with what the objects can functionally do that isn't present in their name.
- The objects are 3D, but images are 2D.
- Objects may be used for different purposes than they are intended for in the real world.
- Context matters. Parts of an image may be hard to determine in isolation, but can be identified when considered in tandem with other pieces.

10.4 Object Detection Models

Ponder This 10.1 *We note that for detecting objects, we can bound their location in a box and identify them. How do we pick these bounding boxes?*

1. Regress Bounding Box: Create an initial bounding box, then regress to minimize distance loss and cross entropy loss from the true bounding box. Doesn't scale well to multiple objects.
2. Sliding Window: Divide the image into smaller windows and classify each. Difficult to choose box sizes for various scenarios.
3. Selective Search: Reduce the number of possible windows to evaluate by selecting likely areas of interest. Heuristic approach: detect edges, group them into contours, rank windows by number of contours within, then only investigate the higher ranked windows.

10.4.1 R-CNN: Region proposals + CNN features

Combining selective search with CNNs to classify multiple objects in the image. Testing process:

1. 2k region proposals are extracted. These proposals may be further refined, e.g. using bounding box regression.
2. For each region proposal, the region is cropped and scaled to the CNN input size, then sent to the CNN.
3. The final layer outputs a 4096-dimensional, fully-connected feature vector, which can then be inputted to a linear classifier for the final classification.

This approach has two main issues: it requires a CNN for each window, making it very slow, and the mechanism for region proposal may be suboptimal.

Alternative: "Fast" R-CNN: Reuse features between proposals by creating them on a convolutional layer and using RoI Pooling (Region of Interest).

1. Forward the entire image through a ConvNet.
2. Make region proposals on a convoluted feature map of the image.
3. Send regions through a RoI Pooling layer before sending them to fully-connected layers.
4. Classify the final output from the FC layer.

Ponder This 10.2 *How do we crop from a feature map?*

1. Resize boxes to account for subsampling.
2. Snap to feature map grid.
3. Overlay a new grid of fixed size.
4. Take max in each cell.

This produces an output of fixed size for all regions which can be mapped back to features.