CMSC 491/691: Computer Vision **Fall 2023**

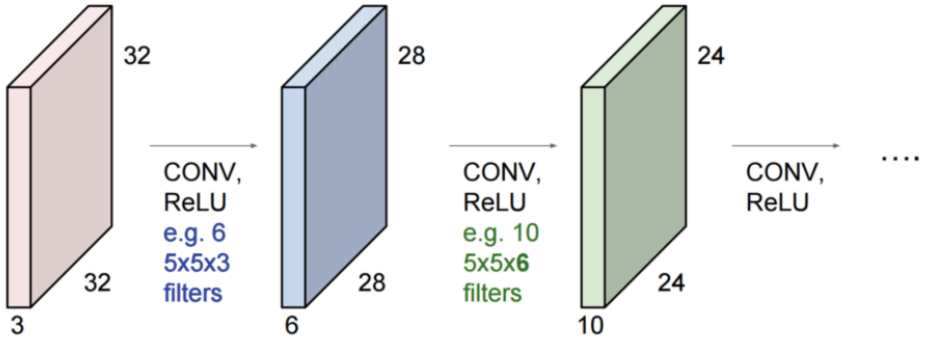
Lecture 10: 2024-03-06 Visual Recognition

Lecturer: Tejas Gokhale *Scribe: Arjun Kundu*

The notes below are an example of what I am expecting. They were taken from a random graduate class. They illustrate some uses of various LaTeX macros. Take a look at this and imitate.

10.1 Recap of last Lecture

A ConvNet is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types). This process can be visualized as moving the filter spatially over the image and calculating a dot product at each position. The operation is represented as: $w^T x + b$ where w represents the filter weights, x is a patch of the input image, and b is a bias term.



10.1.1 Stride and Weighted Sums

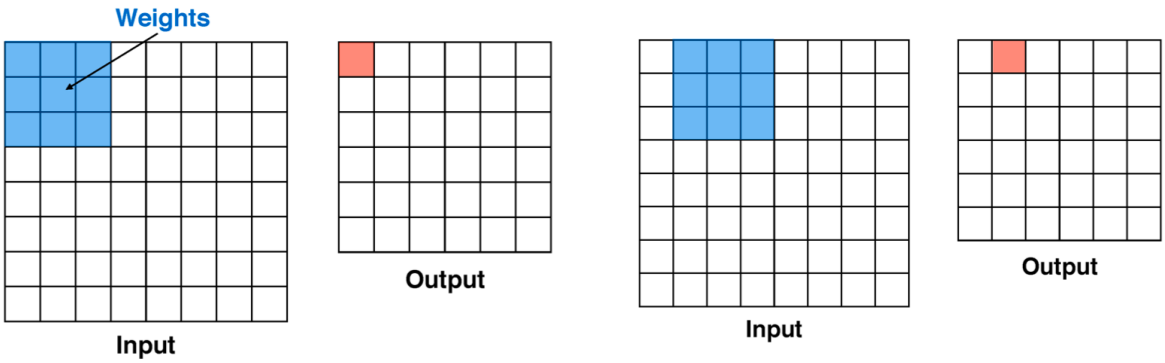
The concept of stride in a convolution layer refers to the step size the filter takes as it moves across the input. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 or more increases the step size accordingly. **Recall that at each position, we are doing a 3D sum** This is expressed in the following weighted sum for a given position r:

$$h^r = \sum_{ijk} x^r_{ijk} W_{ijk} + b \tag{10.1}$$

In this equation, h^r represents the output at a particular location r after the filter has convolved the input.

The variables x^r_{ijk} and W_{ijk} denote the input values and the weights of the filter at the corresponding positions, respectively, and b is the bias term.

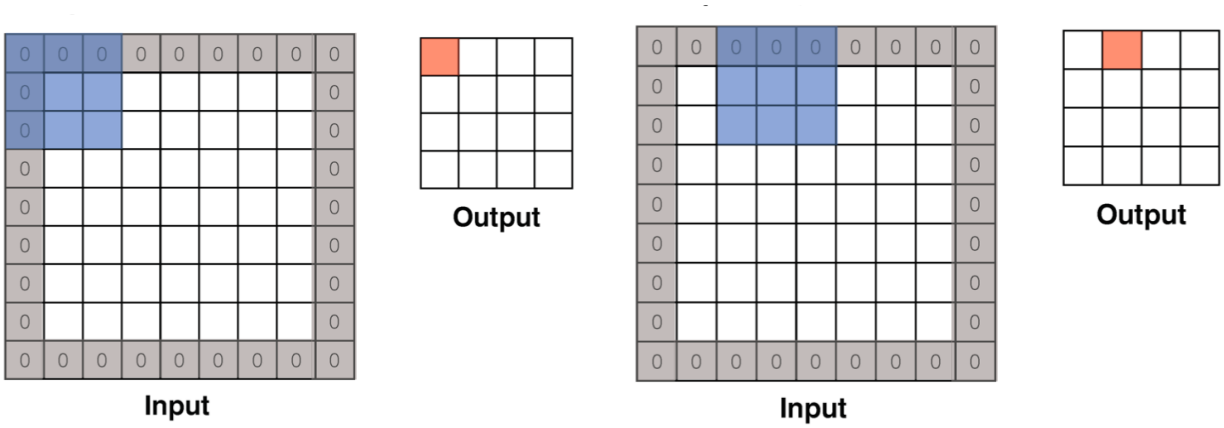
This 3D sum iterates over all the spatial and depth dimensions of the input volume and the filter, computing a weighted sum that captures the local spatial features at each step determined by the stride.



10.1.2 Padding

To process the entire input image including its edges, we can pad the input image with zeros. Padding allows the filter to slide over the border pixels of the image. For instance, a padding of 1 adds one layer of zeros around the image, ensuring that the filter can process the edges effectively.

For Example, here, **pad = 1, strid = 2**



Output Size Formula The size of the output feature map is determined by the formula:

$$\text{Output size} = \left(\frac{W - K + 2P}{S}\right) + 1 \tag{10.2}$$

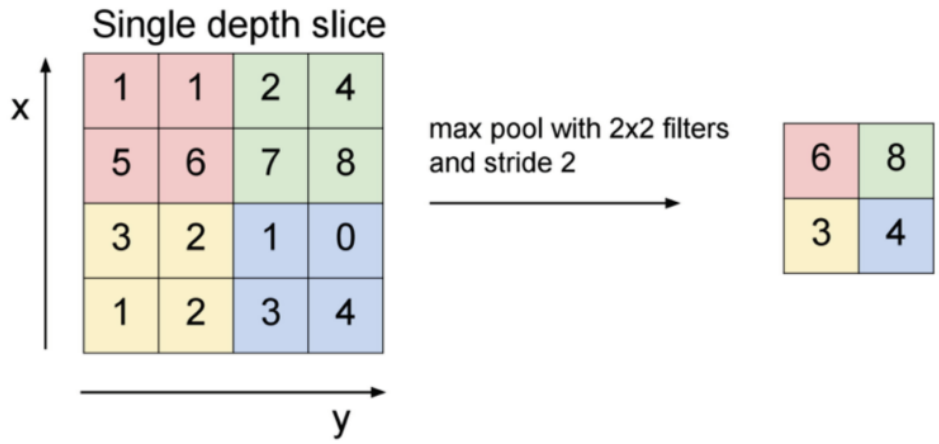
where W is the input dimension, K is the filter size, P is the padding, and S is the stride. This equation allows for the calculation of the output dimensions given any input size, filter size, padding, and stride.

10.1.3 Pooling

Pooling layers follow convolutional layers to reduce spatial dimensions and condense the extracted features. The most common pooling operation is max pooling, which selects the maximum value. In summary, it makes the representations smaller and more manageable, it operates over each activation map independently

10.1.4 Max Pooling

In max pooling, during the forward pass, the index of the maximum value within each pooling region is recorded. During backpropagation, gradients are only passed to these locations. This ensures that the network learns to focus on the most significant features.



10.2 Training ConvNets

To train ConvNets, you would generally do the following:

1. Find a ConvNet architecture suitable for your task.
2. Gather and preprocess your data.
3. Initialize the weights of the ConvNet.
4. Find a suitable learning rate and regularization strength.
5. Minimize the loss function and monitor the training progress.
6. Adjust the model parameters ("fiddle with knobs") as necessary based on performance and validation data.

10.2.1 Mini-batch Gradient Descent Loop

Training a ConvNet typically involves a loop that looks like the following:

1. Sample a batch of training data (e.g., 100 images).
2. Forward pass: Compute the loss (average over the batch).
3. Backward pass: Compute the gradient of the loss function with respect to the parameters.
4. Update all parameters using the gradients computed in the backward pass.

Note: This process is usually referred to as "stochastic gradient descent" (SGD) even though SGD traditionally uses a batch size of 1.

10.2.2 Regularization

Regularization is a technique used to reduce overfitting, which can be accomplished through various methods:

$$L = L_{\text{data}} + L_{\text{reg}} \tag{10.3}$$

$$L_{\text{reg}} = \frac{\lambda}{2}\|W\|_2^2 \tag{10.4}$$

10.2.3 Overfitting

Overfitting: occurs when a model learns the noise in the training set rather than the underlying relationship.
Underfitting: happens when the model does not adequately learn the relationship in the training set.
Generally, models with more capacity or complexity are more likely to overfit.

10.2.4 Summary of Adjustments

When training ConvNets, there are several parameters and settings you may need to adjust:

- **Network architecture:** The structure of the neural network (number and types of layers).
- **Learning rate and its decay schedule**
- **Regularization:** Methods applied to reduce overfitting.
- **Loss function:** The function used to measure the discrepancy between the predicted outputs and the actual labels.
- **Weight initialization:** The method used to initially set the weights of the network before training begins.

10.2.5 Weight Decay

Weight Decay: Regularization is also called "weight decay" because the weights "decay" each iteration:

Gradient Descent Step

$$W \leftarrow W - \alpha\Delta W - \frac{\partial L_{\text{data}}}{\partial W} \tag{10.5}$$

Weights always decay by a certain amount each iteration.

10.2.6 Dropout

Dropout: A simple but powerful technique to reduce overfitting:

Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models. Without dropout, networks may exhibit substantial overfitting.

10.2.7 Summary

In summary, to prevent overfitting and improve the performance of a ConvNet, one should:

- Preprocess the data (subtract the mean, use sub-crops, etc.)
- Initialize weights carefully to avoid symmetry-breaking problems.
- Use Dropout to artificially reduce the number of active neurons during training.
- Use Stochastic Gradient Descent (SGD) with Momentum to improve convergence.
- Fine-tune from pre-trained networks, such as those trained on ImageNet.
- Babysit the network as it trains to ensure optimal performance.

10.2.8 Common Architectures

10.2.8.1 VGG

VGG: VGG is one of the most common CNN architectures used and is also typically employed for feature extraction.

10.2.8.2 ResNet

ResNet: Deeper networks with more layers do not always yield better performance. ResNet addresses this with residual blocks and skip connections.

10.2.9 Object Detection and Scene Understanding

In today's discussion on scene understanding, we will cover:

- Object detection models and their evaluation.
- The importance of recognition and the concept of "categories" in communication.
- Visual challenges with categories, such as the 3D nature of objects versus the 2D nature of images, and the highly varied world

10.2.10 Object Detection Models

We will explore several ideas central to object detection:

- **Idea 1:** Regress bounding box coordinates directly from image pixels.
- **Idea 2:** Use a sliding window approach to detect objects.
- **Idea 3:** Employ selective search to generate and evaluate only a few hundred region proposals, focusing on areas "likely" to contain an object of interest.

10.3 R-CNN: Region proposals + CNN Features

R-CNN, or Region-based Convolutional Neural Networks, combine region proposals with CNN features. The process involves:

- Classifying regions with a linear classifier after forwarding each region through a CNN.
- Utilizing warped image regions.
- Generating region proposals through selective search, roughly 2,000 rectangles that are likely to contain objects.

10.3.1 Problems with R-CNN

Despite its successes, R-CNN faced several issues:

- The approach is slow, as it requires running the CNN for each window.
- May be suboptimal for region proposal

10.3.2 Fast R-CNN

To address the limitations of R-CNN, Fast R-CNN was developed, introducing several improvements:

- Reusing features between proposals to speed up processing.
- Incorporating bounding box regressors and fully connected layers.
- Introducing ROI (Region of Interest) Pooling layer to extract features.

The Fast R-CNN architecture operates by forwarding the whole image through the ConvNet, then using the Conv5 feature map of the image.

10.3.3 ROI Pooling

ROI Pooling is a crucial component of Fast R-CNN that allows for cropping from a feature map. The steps are as follows:

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

This process can be further improved with bilinear sampling, allowing for more precise spatial information to be retained.